Universal Windows Platform 编程学习笔记

Roger Young

2017 年 9 月 25 日

目录

目录 **3**

第一章 Get started with the Universal Windows Platform **5**

1.1 What's a Universal Windows Platform (UWP) app? 5

1.1.1 So, what exactly is a UWP app? 5

1.1.2 Use a language you already know 6

1.1.3 UWP apps come to life on Windows 6

1.1.4 Monetize your app 7

1.1.5 Let's get started 7

第二章 了解应用程序项目 **9**

第三章 XMAL 界面原理和语法 **11**

第四章 应用导航 **13**

第五章 页面排版 **15**

第六章 控件 **17**

第七章 数据视图 **19**

第八章 图形 **21**

8.1 Working with Brushes and Content –XAML and Visual Layer Interop 21

8.1.1 Using XamlCompositionBrushBase 21

8.1.2 Building a Custom Composition Brush 22

8.1.3 Loading images with LoadedImageSurface 29

8.1.4 Using LoadedImageSurface with a Composition Island 30

8.1.5 Wrapping up 31

8.1.6 Lighting UI with XamlLights 32

8.1.7 Using CompositionPropertySets 37

8.1.8 Using ScrollViewerManipulationPropertySet 37

8.1.9 Using PointerPositionPropertySet (new!) 37

8.1.10 Enabling Translation Property –Animating a XAML Element’s Offset using Composition Animations . . . 38
8.1.11 Wrapping up . . . 39
8.2 How to Restart your App Programmatically . . . 39

第九章 动画 **43**

第十章 图像 **45**

第十一章 多媒体 **47**

第十二章 启动与激活 **49**

第十三章 文件和数据 **51**
13.1 文件与目录 . . . 51
13.2 XML 数据处理 . . . 51
13.2.1 XML 格式简介 . . . 51
13.2.2 读写 XML . . . 51
13.2.3 相关类简介 . . . 53
13.3 JSON 数据处理 . . . 53

第十四章 应用设置和数据 **55**

第十五章 数据库编程 **57**
15.1 SQLite 数据库的使用 . . . 57
15.1.1 C# Example . . . 57
15.1.2 SQLite C# API Wrappers . . . 58
15.1.3 Visual Studio set-up . . . 58
15.1.4 Code . . . 59
15.1.5 What about Entity Framework? . . . 63
15.1.6 Conclusion . . . 64

第十六章 网络通信 **65**

第十七章 传感器与地理信息 **67**

第十八章 语言识别技术 **69**
18.1 元素周期表 . . . 71
18.2 密码子表 . . .

第一章　Get started with the Universal Windows Platform

1.1 What's a Universal Windows Platform (UWP) app?

The Universal Windows Platform (UWP) is the app platform for Windows 10. You can develop apps for UWP with just one API set, one app package, and one store to reach all Windows 10 devices – PC, tablet, phone, Xbox, HoloLens, Surface Hub and more. It' s easier to support a number of screen sizes, and also a variety of interaction models, whether it be touch, mouse and keyboard, a game controller, or a pen. At the core of UWP apps is the idea that users want their experiences to be mobile across ALL their devices, and they want to use whatever device is most convenient or productive for the task at hand.

UWP is also flexible: you don't have to use C# and XAML if you don't want to. Do you like developing in Unity or MonoGame? Prefer JavaScript? Not a problem, use them all you want. Have a C++ desktop app that you want to extend with UWP features and sell in the store? That's okay, too.

The bottom line: You can spend your time working with familiar programming languages, frameworks and APIs, all in single project, and have the very same code run on the huge range of Windows hardware that exists today. Once you've written your UWP app, you can then publish it to the store for the world to see.

1.1.1 So, what exactly is a UWP app?

What makes a UWP app special? Here are some of the characteristics that make UWP apps on Windows 10 different.

- There's a common API surface across all devices.

  The Universal Windows Platform (UWP) core APIs are the same for all classes of Windows device. If your app uses only the core APIs, it will run on any Windows 10 device, no matter if you are targeting a desktop PC, an Xbox or a Mixed Reality headset.

- Extension SDKs let your app do cool stuff on specific device types.

  Extension SDKs add specialized APIs for each device class. For example, if your UWP app targets HoloLens, you can add HoloLens features in addition to the normal UWP core APIs. If

you target the universal APIs, your app package can run on all devices that run Windows 10. But if you want your UWP app to take advantage of device specific APIs in the event it is running on a particular class of device, you can check at run-time if an API exists before calling it.

- Apps are packaged using the .AppX packaging format and distributed from the Store.

  All UWP apps are distributed as an AppX package. This provides a trustworthy installation mechanism and ensures that your apps can be deployed and updated seamlessly.

- There's one store for all devices.

  After you register as an app developer, you can submit your app to the store and make it available on all types device, or only those you choose. You submit and manage all your apps for Windows devices in one place.

- Apps support adaptive controls and input

  UI elements use effective pixels (see Responsive design 101 for UWP apps), so they can respond with a layout that works based on the number of screen pixels available on the device. And they work well with multiple types of input such as keyboard, mouse, touch, pen, and Xbox One controllers. If you need to further tailor your UI to a specific screen size or device, new layout panels and tooling help you adapt your UI to the devices your app may run on.

1.1.2 Use a language you already know

UWP apps use the Windows Runtime, a native API built into the operating system. This API is implemented in C++ and supported in C#, Visual Basic, C++, and JavaScript. Some options for writing apps in UWP include:

- XAML UI and a C#, VB, or C++ backend
- DirectX UI and a C++ backend
- JavaScript and HTML

Microsoft Visual Studio 2017 provides a UWP app template for each language that lets you create a single project for all devices. When your work is finished, you can produce an app package and submit it to the Windows Store from within Visual Studio to get your app out to customers on any Windows 10 device.

1.1.3 UWP apps come to life on Windows

On Windows, your app can deliver relevant, real-time info to your users and keep them coming back for more. In the modern app economy, your app has to be engaging to stay at the front of your users' lives. Windows provides you with lots of resources to help keep your users returning to your app:

- Live tiles and the lock screen show contextually relevant and timely info at a glance.
- Push notifications bring real-time, breaking alerts to your user' s attention when they're needed.
- The Action Center is a place where you can organize and display notifications and content that users need to take action on.
- Background execution and triggers bring your app to life just when the user needs it.
- Your app can use voice and Bluetooth LE devices to help users interact with the world around them.
- Support for rich, digital ink and the innovative Dial.
- Cortana adds personality to your software.
- XAML provides you with the tools to create smooth, animated user interfaces.

Finally, you can use roaming data and the Windows Credential Locker to enable a consistent roaming experience across all of the Windows screens where users run your app. Roaming data gives you an easy way to store a user' s preferences and settings in the cloud, without having to build your own sync infrastructure. And you can store user credentials in the Credential Locker, where security and reliability are the top priority.

1.1.4 Monetize your app

On Windows, you can choose how you'll monetize your app—across phones, tablets, PCs, and other devices. We give you a number of ways to make money with your app and the services it delivers. All you need to do is choose the one that works best for you:

- A paid download is the simplest option. Just name your price.
- Trials let users try your app before buying it, providing easier discoverability and
- conversion than the more traditional "freemium" options.
- Use sale prices for apps and add-ons.
- In-app purchases and ads are also available.

1.1.5 Let's get started

For a more detailed look at the UWP, read the Guide to Universal Windows Platform apps. Then, check out Get set up to download the tools you need to start creating apps, and then write your first app!

第二章　了解应用程序项目

第三章　XMAL 界面原理和语法

第四章　应用导航

第五章　页面排版

第六章　控件

第七章　数据视图

第八章　图形

8.1 Working with Brushes and Content -XAML and Visual Layer Interop

https://blogs.windows.com/buildingapps/2017/07/19/new-lights-propertyset-interop-xaml-visual-l
amp/

The Composition APIs empower Universal Windows Platform (UWP) developers to do beautiful and powerful things when they access the Visual Layer. In the Windows 10 Creators Update, we made working with the Visual Layer much easier with new, powerful APIs.

In this blog series, we' ll cover some of these improvements in the Creators Update and take a look at the following APIs:

- XamlCompositionBrushBase -easily paint a XAML UIElement with a CompositionBrush
- LoadedImageSurface -load an image easily and use with Composition APIs
- XamlLights -apply lights to your XAML UI with a single line of XAML
- PointerPositionPropertySet -create 60 FPS animations using pointer position, off the UI thread!
- Enabling the Translation property -animate a XAML UI Element using Composition animation

If you' d like to review the previously available ElementCompositionPreview APIs, for example working with "hand-in" and "hand-out" Visuals, you can quickly catch up here.

8.1.1 Using XamlCompositionBrushBase

One of the benefits of the new Composition and XAML interop APIs is the ability to use a CompositionBrush to directly paint a XAML UIElement rather than being limited to XAML brushes only. For example, you can create a CompositionEffectBrush that applies a tinted blur to the content beneath and use the brush to paint a XAML rectangle that can be included in the XAML markup

This is accomplished by using the new abstract class XamlCompositionBrushBase available in the Creators Update. To use it, you subclass XamlCompositionBrushBase to create your own XAML Brush that can be used in your markup. As seen the example code below, the XamlCompositionBrushBase

exposes a CompositionBrush property that you set with your effect (or any CompositionBrush) and it will be applied to the XAML element.

This effectively replaces the need to manually create SpriteVisuals with SetElementChild for most effect scenarios. In addition to needing less code to create and add an effect to the UI, using a Brush means you get the following added benefits for free:

- Theming and Styling
- Binding
- Resource and Lifetime management
- Layout aware
- PointerEvents
- HitTesting and other XAML-based advantages

Microsoft, as part of the Fluent Design System, has included a few Brushes in the Creators Update that leverage the features of XamlCompositionBrushBase:

- AcrylicBrush
- RevealBrush
- RevealBorderBrush
- RevealBackgroundBrush

8.1.2 Building a Custom Composition Brush

Let's create a XamlCompositionBrush of our own to see how simple this can be. Here's what we'll create:

To start, let's create a very simple Brush that applies an InvertEffect to content under it. First, we'll need to make a public sealed class that inherits from XamlCompositionBrushBase and override two methods:

- OnConnected
- OnDisconnected

Let's dive into the code. First, create your Brush class, which inherits from XamlCompositionBrushBase:

```
1 public class InvertBrush : XamlCompositionBrushBase
2
3 {
4
5     protected override void OnConnected()
```

```
6
7       {
8
9           if (CompositionBrush == null)
10
11          {
12
13              // 1 - Get the BackdropBrush, this gets what is behind the UI element
14
15              var backdrop = Window.Current.Compositor.CreateBackdropBrush();
16
17
18
19              // CompositionCapabilities: Are effects supported? If not, return.
20
21              if (!CompositionCapabilities.GetForCurrentView().AreEffectsSupported())
22
23              {
24
25                  return;
26
27              }
28
29
30
31              // 2 - Create your Effect
32
33              // New-up a Win2D InvertEffect and use the BackdropBrush as its Source
34
35              // Note - To use InvertEffect, you'll need to add the Win2D NuGet package to your
                    project (search NuGet for &amp;quot;Win2D.uwp&amp;quot;)
36
37              var invertEffect = new InvertEffect
38
39              {
40
41                  Source = new CompositionEffectSourceParameter(&amp;quot;backdrop&amp;quot;)
42
43              };
44
45
46
47              // 3 - Set up the EffectFactory
48
49              var effectFactory = Window.Current.Compositor.CreateEffectFactory(invertEffect);
50
51
52
53              // 4 - Finally, instantiate the CompositionEffectBrush
54
55              var effectBrush = effectFactory.CreateBrush();
56
57
58
59              // and set the backdrop as the original source
```

```
60
61              effectBrush.SetSourceParameter(&amp;quot;backdrop&amp;quot;, backdrop);
62
63
64
65              // 5 - Finally, assign your CompositionEffectBrush to the XCBB's CompositionBrush
                    property
66
67              CompositionBrush = effectBrush;
68
69          }
70
71      }
72
73
74
75      protected override void OnDisconnected()
76
77      {
78
79          // Clean up
80
81          CompositionBrush?.Dispose();
82
83          CompositionBrush = null;
84
85      }
86
87  }
```

There are a few things to call out in the code above.

- In the OnConnected method, we get a CompositionBackdropBrush. This allows you to easily get the pixels behind the UIElement.
- We use fallback protection. If the user’s device doesn’t have support for the effect(s), then just return.
- Next, we create the InvertEffect and use the backdropBrush for the Effect’s Source.
- Then, we pass the finished InvertEffect to the CompositionEffectFactory.
- Finally, we get an EffectBrush from the factory and set the XamlCompositionBrushBase.CompositionBrush property with our newly created effectBrush.

Now you can use it in your XAML. For example, let’s apply it to a Grid on top of another Grid with a background image:

```
1
2   <Grid>
3
4   <Grid.Background>
5
```

```
<ImageBrush ImageSource="ms-appx:///Images/Background.png"/>

</Grid.Background>

<Grid Width="300" Height="300"

HorizontalAlignment="Center"

VerticalAlignment="Center">

<Grid.Background>

<brushes:InvertBrush />

</Grid.Background>

</Grid>

</Grid>
```

Now that you know the basics of creating a brush, let' s build an animated effect brush next. Creating a Brush with Animating Effects Now that you see how simple it is to create a CompositionBrush, let' s create a brush that applies a TemeratureAndTint effect to an image and animate the Temperature value:

We start the same way we did with the simple InvertBrush, but this time we' ll add a DependencyProperty, ImageUriString, so that we can load an image using LoadedImageSurface in the OnConnected method.

```
public sealed class ImageEffectBrush : XamlCompositionBrushBase

{

    private LoadedImageSurface _surface;

    private CompositionSurfaceBrush _surfaceBrush;



    public static readonly DependencyProperty ImageUriStringProperty = DependencyProperty.Register(

    &amp;quot;ImageUri&amp;quot;,

    typeof(string),

    typeof(ImageEffectBrush),

    new PropertyMetadata(string.Empty, OnImageUriStringChanged)

    );



    public string ImageUriString
```

```
26
27      {
28
29          get =&gt; (String)GetValue(ImageUriStringProperty);
30
31          set =&gt; SetValue(ImageUriStringProperty, value);
32
33      }
34
35
36
37      private static void OnImageUriStringChanged(DependencyObject d,
            DependencyPropertyChangedEventArgs e)
38
39      {
40
41          var brush = (ImageEffectBrush)d;
42
43          // Unbox and update surface if CompositionBrush exists
44
45          if (brush._surfaceBrush != null)
46
47          {
48
49              var newSurface = LoadedImageSurface.StartLoadFromUri(new Uri((String)e.NewValue));
50
51              brush._surface = newSurface;
52
53              brush._surfaceBrush.Surface = newSurface;
54
55          }
56
57      }
58
59
60
61      protected override void OnConnected()
62
63      {
64
65          // return if Uri String is null or empty
66
67          if (string.IsNullOrEmpty(ImageUriString))
68
69          return;
70
71
72
73          // Get a reference to the Compositor
74
75          Compositor compositor = Window.Current.Compositor;
76
77
78
79          // Use LoadedImageSurface API to get ICompositionSurface from image uri provided
```

```
80
81          _surface = LoadedImageSurface.StartLoadFromUri(new Uri(ImageUriString));
82
83
84
85          // Load Surface onto SurfaceBrush
86
87          _surfaceBrush = compositor.CreateSurfaceBrush(_surface);
88
89          _surfaceBrush.Stretch = CompositionStretch.UniformToFill;
90
91
92
93          // CompositionCapabilities: Are Tint+Temperature and Saturation supported?
94
95          bool usingFallback = !CompositionCapabilities.GetForCurrentView().AreEffectsSupported();
96
97          if (usingFallback)
98
99          {
100
101             // If Effects are not supported, Fallback to image without effects
102
103             CompositionBrush = _surfaceBrush;
104
105             return;
106
107         }
108
109
110
111         // Define Effect graph (add the Win2D.uwp NuGet package to get this effect)
112
113         IGraphicsEffect graphicsEffect = new SaturationEffect
114
115         {
116
117             Name = &amp;quot;Saturation&amp;quot;,
118
119             Saturation = 0.3f,
120
121             Source = new TemperatureAndTintEffect
122
123             {
124
125                 Name = &amp;quot;TempAndTint&amp;quot;,
126
127                 Temperature = 0,
128
129                 Source = new CompositionEffectSourceParameter(&amp;quot;Surface&amp;quot;),
130
131             }
132
133         };
134
```

```
135
136
137         // Create EffectFactory and EffectBrush
138
139         CompositionEffectFactory effectFactory = compositor.CreateEffectFactory(graphicsEffect, new
                [] { &amp;quot;TempAndTint.Temperature&amp;quot; });
140
141         CompositionEffectBrush effectBrush = effectFactory.CreateBrush();
142
143         effectBrush.SetSourceParameter(&amp;quot;Surface&amp;quot;, _surfaceBrush);
144
145
146
147         // Set EffectBrush to paint Xaml UIElement
148
149         CompositionBrush = effectBrush;
150
151
152
153         // Trivial looping animation to demonstrate animated effect
154
155         ScalarKeyFrameAnimation tempAnim = compositor.CreateScalarKeyFrameAnimation();
156
157         tempAnim.InsertKeyFrame(0, 0);
158
159         tempAnim.InsertKeyFrame(0.5f, 1f);
160
161         tempAnim.InsertKeyFrame(1, 0);
162
163         tempAnim.Duration = TimeSpan.FromSeconds(5);
164
165         tempAnim.IterationBehavior = AnimationIterationBehavior.Count;
166
167         tempAnim.IterationCount = 10;
168
169         effectBrush.Properties.StartAnimation(&amp;quot;TempAndTint.Temperature&amp;quot;, tempAnim);
170
171     }
172
173
174
175     protected override void OnDisconnected()
176
177     {
178
179         // Dispose Surface and CompositionBrushes if XamlCompBrushBase is removed from tree
180
181         _surface?.Dispose();
182
183         _surface = null;
184
185
186
187         CompositionBrush?.Dispose();
188
```

```
189         CompositionBrush = null;
190
191     }
192
193 }
```

There are some new things here to call out that are different from the InvertBrush: We use the new LoadedImageSurface API to easily load an image in the OnConnected method, but also when the ImageUriString value changes. Prior to Creators Update, this required a hand-in Visual (a SpriteVisual, painted with an EffectBrush, which was handed back into the XAML Visual Tree). See the LoadedImageSurface section later in this article for more details.

Notice that we gave the effects a Name value. In particular, TemperatureAndTintEffect, uses the name "TempAndTint." This is required to animate properties as it is used for the reference to the effect in the AnimatableProperties array that is passed to the effect factory. Otherwise, you' ll encounter a "Malformed animated property name" error. After we assign the CompositionBrush property, we created a simple looping animation to oscillate the value of the TempAndTint from 0 to 1 and back every 5 seconds. Let' s take a look at an instance of this Brush in markup:

```
1 <Grid>
2
3 <Grid.Background>
4
5 <brushes:ImageEffectBrush ImageUriString="ms-appx:///Images/Background.png"/>
6
7 </Grid.Background>
8
9 </Grid>
```

For more information on using XamlCompositionBrushBase, see here. Now, let' s take a closer look at how easy it is now to bring in images to the Visual layer using LoadedImageSurface

8.1.3 Loading images with LoadedImageSurface

With the new LoadedImageSurface class, it' s never been easier to load an image and work with it in the visual layer. The class has the same codec support that Windows 10 has via the Windows Imaging Component (see full list here), thus it supports the following image file types:

- Joint Photographic Experts Group (JPEG)
- Portable Network Graphics (PNG)
- Bitmap (BMP)
- Graphics Interchange Format (GIF)
- Tagged Image File Format (TIFF)
- JPEG XR

- Icons (ICO)

NOTE: When using an animated GIF, only the first frame will be used for the Visual, as animation is not supported in this scenario.

To load in an image, you can use one of the four factory methods:

- StartLoadFromUri(Uri)
- StartLoadFromUri(Uri, Size)
- StartLoadFromStream(IRandomAccessStream)
- StartLoadFromStream(IRandomAccessStream, Size)

As you can see there are two ways to load an image: with a Uri or a Stream. Additionally, you have an option to use an overload to set the size of the image (if you don't pass in a Size, it will decode to the natural size).

```
CompositionSurfaceBrush imageBrush = compositor.CreateSurfaceBrush();

LoadedImageSurface loadedSurface = LoadedImageSurface.StartLoadFromUri(new Uri(&quot;ms-appx:///
    Images/Photo.jpg&quot;), new Size(200.0, 200.0));

imageBrush.Surface = loadedSurface;
```

This is very helpful when you need to load an image that will be used for your CompositionBrush (e.g. CompositionEffectBrush) or SceneLightingEffect (e.g. NormalMap for textures) as you no longer need to manually create a hand-in Visual (a SpriteVisual painted with an EffectBrush). In an upcoming post in this series, we will explore this further using NormalMap images with to create advanced lighting to create unique and compelling materials.

8.1.4 Using LoadedImageSurface with a Composition Island

LoadedImageSurface is also useful when loading an image onto a SpriteVisual inserted in XAML UI using ElementCompositionPreview. For this scenario, you can use the Loaded event to adjust the visual's properties after the image has finished loading. Here is an example of using LoadedImageSurface for a CompositionSurfaceBrush, then updating the SpriteVisual's size with the image's DecodedSize when the image is loaded:

```
private SpriteVisual spriteVisual;

private void LoadImage(Uri imageUri)

{

    CompositionSurfaceBrush surfaceBrush = Window.Current.Compositor.CreateSurfaceBrush();



    // You can load an image directly and set a SurfaceBrush's Surface property with it
```

```
12
13      var loadedImageSurface = LoadedImageSurface.StartLoadFromUri(imageUri);
14
15      loadedImageSurface.LoadCompleted += Load_Completed;
16
17      surfaceBrush.Surface = loadedImageSurface;
18
19
20
21      // We'll use a SpriteVisual for the hand-in visual
22
23      spriteVisual = Window.Current.Compositor.CreateSpriteVisual();
24
25      spriteVisual.Brush = surfaceBrush;
26
27
28
29      ElementCompositionPreview.SetElementChildVisual(MyCanvas, spriteVisual);
30
31  }
32
33
34
35  private void Load_Completed(LoadedImageSurface sender, LoadedImageSourceLoadCompletedEventArgs args)
36
37  {
38
39      if (args.Status == LoadedImageSourceLoadStatus.Success)
40
41      {
42
43          Size decodedSize = sender.DecodedSize;
44
45          spriteVisual.Size = new Vector2((float)decodedSize.Width, (float)decodedSize.Height);
46
47      }
48
49  }
```

There are some things you should be aware before getting started with LoadedImageSurface. This class makes working with images a lot easier, however you should understand the lifecycle and when images get decoded/sized. We recommend that you take a couple minutes and read the documentation before getting started.

8.1.5 Wrapping up

Using Composition features in your XAML markup is easier than ever. From painting your UIElements with CompositionBrushes and applying lighting, to smooth off-UIThread animations, the power of the Composition API is more accessible than ever.

In the next post, we' ll explore more new APIs like the new Translation property, using XamlLights in your XAML markup and how to create a custom light using the new PointerPositionPropertySet.

8.1.6 Lighting UI with XamlLights

A powerful new feature in the Creators Update is the ability to set and use a Lighting effect directly in XAML by leveraging the abstract XamlLight class.

Creating a XamlLight starts just like a XamlCompositionBrushBase does, with an OnConnected and OnDisconnected method (see Part One here), but this time you inherit from the XamlLight subclass to create your own unique lighting that can be used directly in XAML. Microsoft uses this with the Reveal effect that comes with the Creators Update.

To see this in action, let’ s build a demo that creates the animated GIF you see at the top of this post. It combines everything you learned about XamlCompositionBrushBase and LoadedImageSurface in the last post, but this example has two XamlLights: a HoverLight and an AmbientLight.

Let’ s begin with creating the AmbientLight first. To get started, we begin similarly to the XamlCompositionBrushBase with an OnConnected and OnDisconnected method. However, for a XamlLight we set the CompositionLight property of the XamlLight subclass.

```
1  public class AmbLight : XamlLight
2
3  {
4
5  protected override void OnConnected(UIElement newElement)
6
7  {
8
9  Compositor compositor = Window.Current.Compositor;
10
11
12
13 // Create AmbientLight and set its properties
14
15 AmbientLight ambientLight = compositor.CreateAmbientLight();
16
17 ambientLight.Color = Colors.White;
18
19
20
21 // Associate CompositionLight with XamlLight
22
23 CompositionLight = ambientLight;
24
25
26
27 // Add UIElement to the Light's Targets
28
29 AmbLight.AddTargetElement(GetId(), newElement);
30
31 }
32
33
34
35 protected override void OnDisconnected(UIElement oldElement)
36
```

```
{

// Dispose Light when it is removed from the tree

AmbLight.RemoveTargetElement(GetId(), oldElement);

CompositionLight.Dispose();

}



protected override string GetId() => typeof(AmbLight).FullName;

}
```

With ambient lighting done, let' s build the SpotLight XamlLight. One of the main things we want the SpotLight to do is follow the user' s pointer. To do this, we can now use GetPointerPositionPropertySet method of ElementCompositionPreview to get a CompositionPropertySet we can use with a Composition ExpressionAnimation (PointerPositionPropertySet is explained in more detail in the PropertySets section below).

Here is the finished XamlLight implementation that creates that animated spotlight. Read the code comments to see the main parts of the effects, particularly how the resting position and animated offset position are used to create the lighting.

```
public class HoverLight : XamlLight

{

private ExpressionAnimation _lightPositionExpression;

private Vector3KeyFrameAnimation _offsetAnimation;



protected override void OnConnected(UIElement targetElement)

{

Compositor compositor = Window.Current.Compositor;



// Create SpotLight and set its properties

SpotLight spotLight = compositor.CreateSpotLight();

spotLight.InnerConeAngleInDegrees = 50f;

spotLight.InnerConeColor = Colors.FloralWhite;

spotLight.OuterConeAngleInDegrees = 0f;

```

```
spotLight.ConstantAttenuation = 1f;

spotLight.LinearAttenuation = 0.253f;

spotLight.QuadraticAttenuation = 0.58f;



// Associate CompositionLight with XamlLight

this.CompositionLight = spotLight;



// Define resting position Animation

Vector3 restingPosition = new Vector3(200, 200, 400);

CubicBezierEasingFunction cbEasing = compositor.CreateCubicBezierEasingFunction( new Vector2(0.3f, 0.7f), new Vector2(0.9f, 0.5f));

_offsetAnimation = compositor.CreateVector3KeyFrameAnimation();

_offsetAnimation.InsertKeyFrame(1, restingPosition, cbEasing);

_offsetAnimation.Duration = TimeSpan.FromSeconds(0.5f);



spotLight.Offset = restingPosition;



// Define expression animation that relates light's offset to pointer position

CompositionPropertySet hoverPosition = ElementCompositionPreview.GetPointerPositionPropertySet(targetElement);

_lightPositionExpression = compositor.CreateExpressionAnimation("Vector3(hover.Position.X, hover.Position.Y, height)");

_lightPositionExpression.SetReferenceParameter("hover", hoverPosition);

_lightPositionExpression.SetScalarParameter("height", 15.0f);



// Configure pointer entered/ exited events

targetElement.PointerMoved += TargetElement_PointerMoved;

targetElement.PointerExited += TargetElement_PointerExited;



```

```
// Add UIElement to the Light's Targets

HoverLight.AddTargetElement(GetId(), targetElement);

}



private void MoveToRestingPosition()

{

// Start animation on SpotLight's Offset

CompositionLight?.StartAnimation("Offset", _offsetAnimation);

}



private void TargetElement_PointerMoved(object sender, PointerRoutedEventArgs e)

{

if (CompositionLight == null) return;



// touch input is still UI thread-bound as of the Creators Update

if (e.Pointer.PointerDeviceType == Windows.Devices.Input.PointerDeviceType.Touch)

{

Vector2 offset = e.GetCurrentPoint((UIElement)sender).Position.ToVector2();

(CompositionLight as SpotLight).Offset = new Vector3(offset.X, offset.Y, 15);

}

else

{

// Get the pointer's current position from the property and bind the SpotLight's X-Y Offset

CompositionLight.StartAnimation("Offset", _lightPositionExpression);

}

}



private void TargetElement_PointerExited(object sender, PointerRoutedEventArgs e)
```

```
{
// Move to resting state when pointer leaves targeted UIElement
MoveToRestingPosition();
}


protected override void OnDisconnected(UIElement oldElement)
{
// Dispose Light and Composition resources when it is removed from the tree
HoverLight.RemoveTargetElement(GetId(), oldElement);
CompositionLight.Dispose();
_lightPositionExpression.Dispose();
_offsetAnimation.Dispose();
}


protected override string GetId() => typeof(HoverLight).FullName;
}
```

Now, with the HoverLight class done, we can add both the AmbLight and the HoverLight to previous ImageEffectBrush (find ImageEffectBrush in the last post):

```
<Grid>
<Grid.Background>
<brushes:ImageEffectBrush ImageUriString="ms-appx:///Images/Background.png" />
</Grid.Background>


<Grid.Lights>
<lights:HoverLight/>
<lights:AmbLight/>
</Grid.Lights>
</Grid>
```

Note: To add a XamlLight in markup, your Min SDK version must be set to Creators Update, otherwise you can set it in the code behind.

For more information, go here to read more about using XamlLight and here to see the Lighting documentation.

8.1.7 Using CompositionPropertySets

When you want to use the values of the ScrollViewer' s Offset or the Pointer' s X and Y position (e.g. mouse cursor) to do things like animate effects, you can use ElementCompositionPreview to retrieve their PropertySets. This allows you to create amazingly smooth, 60 FPS animations that are not tied to the UI thread. These methods let you get the values from user interaction for things like animations and lighting.

8.1.8 Using ScrollViewerManipulationPropertySet

This PropertySet is useful for animating things like Parallax, Translation and Opacity.

```
1 // Gets the manipulation <ScrollViewer x:Name="MyScrollViewer"/>
2
3 CompositionPropertySet scrollViewerManipulationPropertySet =
4
5 ElementCompositionPreview.GetScrollViewerManipulationPropertySet(MyScrollViewer);
```

To see an example, go to the Smooth Interaction and Motion blog post in this series. There is a section devoted to using the ScrollViewerManipulationPropertySet to drive the animation.

8.1.9 Using PointerPositionPropertySet (new!)

New in the Creators Update, is the PointerPositionPropertySet. This PropertySet is useful for creating animations for lighting and tilt. Like ScrollViewerManipulationPropertySet, PointerPosition-PropertySet enables fast, smooth and UI thread independent animations.

A great example of this is the animation mechanism behind Fluent Design' s RevealBrush, where you see lighting effects on the edges on the UIElements. This effect is created by a CompositionLight, which has an Offset property animated by an ExpressionAnimation using the values obtained from the PointerPositionPropertySet.

```
1 // Useful for creating an ExpressionAnimation
2
3 CompositionPropertySet pointerPositionPropertySet = ElementCompositionPreview.
      GetPointerPositionPropertySet(targetElement);
4
5 ExpressionAnimation expressionAnimation = compositor.CreateExpressionAnimation("Vector3(param.
      Position.X, param.Position.Y, height)");
6
7 expressionAnimation.SetReferenceParameter("param", pointerPositionPropertySet);
```

To get a better understanding of how you can use this to power animations in your app, let's explore XamlLights and create a demo that uses the PointerPositionPropertySet to animate a SpotLight.

8.1.10 Enabling Translation Property -Animating a XAML Element's Offset using Composition Animations

As discussed in our previous blog post, property sharing between the Framework Layer and the Visual Layer used to be tricky prior to the Creators Update. The following Visual properties are shared between UIElements and their backing Visuals:

- Offset
- Scale
- Opacity
- TransformMatrix
- InsetClip
- CompositeMode

Prior to the Creators update, Scale and Offset were especially tricky because, as mentioned before, a UIElement isn't aware of changes to the property values on the hand-out Visual, even though the hand-out Visual is aware of changes to the UIElement. Consequently, if you change the value of the hand-out Visual's Offset or Size property and the UIElement's position changes due to a page resize, the UIElement's previous position values will stomp all over your hand-out Visual's values.

Now with the Creators Update, this has become much easier to deal with as you can prevent Scale and Offset stomping by enabling the new Translation property on your element, by way of the ElementCompositionPreview object.

```
ElementCompositionPreview.SetIsTranslationEnabled(Rectangle1, true);



//Now initialize the value of Translation in the PropertySet to zero for first use to avoid timing
    issues. This ensures that the property is ready for use immediately.



var rect1VisualPropertySet = ElementCompositionPreview.GetElementVisual(Rectangle1).Properties;

rect1VisualPropertySet.InsertVector3("Translation", Vector3.Zero);
```

Then, animate the visual's Translation property where previously you would have animated its Offset property.

```
// Old way, subject to property stomping:

visual.StartAnimation("Offset.Y", animation);

// New way, available in the Creators Update

visual.StartAnimation("Translation.Y", animation);
```

By animating a different property from the one affected during layout passes, you avoid any unwanted offset stomping coming from the XAML layer.

8.1.11 Wrapping up

In the past couple posts, we explored some of the new features of XAML and Composition Interop and how using Composition features in your XAML markup is easier than ever. From painting your UIElements with CompositionBrushes and applying lighting, to smooth off-UIThread animations, the power of the Composition API is more accessible than ever.

In the next post, we' ll dive deeper into how you can chain Composition effects to create amazing materials and help drive the evolution of Fluent Design.

8.2 How to Restart your App Programmatically

For some apps (especially games) it is not uncommon for the app to get into a state where it needs to restart –perhaps after a license update, after installing downloadable content, its caches have become corrupt or unwieldy, or for any other reason where the app needs to refresh state from scratch. In earlier releases, your only option would have been to prompt the user to close and relaunch, or to call CoreApplication.Exit –and both options provide sub-optimal user experience.

We have therefore introduced a new API that enables an app to request immediate termination and restart, and to pass arbitrary arguments into the fresh instance. In this post, we' ll look at how this works and how you can build it into your app. This is available now in Insider builds from Build 16226 onwards, along with the corresponding SDK.

Here' s a sample app, called TestRestart.

The app provides a ListView of cities on the left, the currently-selected city on the right and a TextBox for providing arguments to the app when it is restarted. When the user taps the Request Restart button, the app will terminate and restart itself, passing in the supplied arguments. The new API, RequestRestartAsync, is exposed as a static method on the CoreApplication object. It takes a string parameter, which can be any string value you like –including input from the user or another external entity. If you do choose to accept input in this way, it is your responsibility to validate it correctly to make sure it conforms to whatever constraints you choose to impose. You should do this validation on input, before passing it to RequestRestartAsync. In this sample app, we' re expecting the user to type in the name of a city.

```
async private void DoRestartRequest()
{
    bool isValidPayload = false;
    string payload = restartArgs.Text;
    if (!string.IsNullOrEmpty(payload))
    {
        foreach (ImageViewModel imageItem in imageListView.Items)
        {
            if (imageItem.Name == payload)
            {
```

```
                isValidPayload = true;
                break;
            }
        }
    }

    if (isValidPayload)
    {
        AppRestartFailureReason result =

        await CoreApplication.RequestRestartAsync(payload);

        if (result == AppRestartFailureReason.NotInForeground ||

        result == AppRestartFailureReason.RestartPending ||

        result == AppRestartFailureReason.Other)

        {

            Debug.WriteLine("RequestRestartAsync failed: {0}", result);

        }

    }

}
```

To mitigate privacy concerns, an app is only permitted to restart itself if it is in the foreground at the time it makes the request. When the app restarts, it restarts with normal UI -that is, as a normal foreground window. If we were to permit a background task or minimized app to restart, the result would be unexpected to the user. This is why the API is framed as a request. If the request is denied, the app would need to handle the failure -perhaps by waiting until it is in the foreground and trying again. If you were to request a restart and then through some twist of logic managed to request it again before the system started the operation, then you' d get the RestartPending result, although this is an edge case. You' re unlikely to ever get the other result -unless something goes wrong in the platform.

Note that this is the only significant constraint, but you should use this API carefully. For example, you probably should not use it if your app was not originally launched by the user -for example, if it was launched as the result of a share or picker operation. Restarting in the middle of one of those contract operations would certainly confuse the user.

If the request is granted, the app is terminated and then restarted. There are many different ways to activate an app: in addition to a regular launch activation, apps can choose to support file activation, protocol activation, share or picker activation and so on. The list is documented here. For the restart case, the app will be activated as a regular launch -just as if the user had closed the app manually and tapped its tile to launch it again -but including the arbitrary arguments supplied earlier (if any).

In your App class, you should handle this by overriding the OnActivated method. Test the ActivationKind, and if it' s ActivationKind.Launch, then the incoming IActivatedEventArgs will be a LaunchActivatedEventArgs. From this, you can get hold of the incoming activation arguments. For a regular user-initiated launch, the Arguments will be empty, so if it' s not empty you could simply infer that this is a restart activation. You can also check the PreviousExecutionState, which for a restart operation will be set to Terminated.

Although the arguments might have originated from an untrusted source (eg, the user), you should have done the validation before requesting restart. If so, you can consider them trustworthy when you receive them in OnActivated.

```
protected override void OnActivated(IActivatedEventArgs args)

{

    switch (args.Kind)

    {

        case ActivationKind.Launch:

        LaunchActivatedEventArgs launchArgs = args as LaunchActivatedEventArgs;

        string argString = launchArgs.Arguments;



        Frame rootFrame = Window.Current.Content as Frame;

        if (rootFrame == null)

        {

            rootFrame = new Frame();

            Window.Current.Content = rootFrame;

        }

        rootFrame.Navigate(typeof(MainPage), argString);

        Window.Current.Activate();

        break;

    }

}
```

What you do with the incoming arguments is entirely up to you. In this app, we' re simply passing them on to the MainPage. In the MainPage in turn, we have an override of OnNavigatedTo which uses the string to select an item in the ListView:

```
protected override void OnNavigatedTo(NavigationEventArgs e)



{

    string payload = e.Parameter as string;

    if (!string.IsNullOrEmpty(payload))

    {

        foreach (ImageViewModel imageItem in imageListView.Items)

        {

            if (imageItem.Name == payload)

            {

                imageListView.SelectedItem = imageItem;

                break;

            }

        }

    }

}
```

As you can see, the CoreApplication.RequestRestartAsync method is a simple API. You can use it to terminate your app immediately, and have it restart as if by user action, with the additional option of passing in arbitrary arguments on activation.

第九章　动画

第十章　图像

第十一章　多媒体

第十二章　启动与激活

第十三章　文件和数据

在应用程序的开发过程中，数据的读写操作是一项基本、必备的操作。需要熟练掌握相关的技术。

13.1　文件与目录

UWP 提供了对文件/目录操作的支持，使用 UWP 提供的 API，可以实现最常见的文件读写操作。

文件读写相关的主要类型，主要分布在 ***Windows.Storage*** 命名空间及该命名空间的子命名空间中。

表 13.1: 文件/目录操作相关的重要类型

类型名	说明
StorageFile	表示一个文件实例。
StorageFolder	表示一个目录。
KnownFolders	静态类型，可访问几个特殊的目录。
FileIO	针对 StorageFile 实例而设计，通过一系列静态方法来实现对文件的读写。
PathIO	针对表示文件（夹）的字符串而设计，通过一系列静态方法来实现对文件的读写。

13.2　XML 数据处理

13.2.1　XML 格式简介

13.2.2　读写 XML

Windows.Data.Xml.Dom 命名空间下提供了若干类型，可以帮助开发者对 XML 文档进行读取、写入、筛选等操作。其中 ***XmlDocument*** 类是比较核心的类型，它表示一个 XML 文档的实例，使用该类可以通过代码来构建 XML 文档。

在读取 XML 数据时，可以调用 ***LoadFromUriAsync***（从 URI 加载）方法或 ***LoadFromFileAsync***（从文件加载）两个静态成员，直接返回 ***XmlDocument*** 对象实例，随后，就可以对 XML 文档进行读取或编辑操作。如果要创建全新的 XML 文档，可在实例化 ***XmlDocument*** 类后，调用相应的方法来为文档创建各种类型的节点：

- ***CreateComment*** 方法：创建 XML 注释节点，返回类型为 ***XmlComment***；

- ***CreateElement*** 方法：创建 XML 元素，返回类型为 ***XmlElement***；
- ***CreateTextNode*** 方法：创建纯文本节点；
- ***CreateAttribute*** 方法：创建 XML 元素属性的 "$name = value$" 对，返回类型为 ***XmlAttribute***；
- ***CreateCDataSection*** 方法：创建 CData 节点，可以存储不被 XML 解析器分析的文本；

不管是 ***XmlDocument***，还是 ***XmlComment***、***XmlElement*** 等类型的节点，它们都实现了同一个接口 ***IXmlNode***，所以只需要调用 ***A*** 节点的 ***AppendChild*** 方法，并将 ***B*** 节点作为参数，就可以将 ***B*** 节点添加到 ***A*** 节点的子节点集合中。要从 ***A*** 节点的子节点集合中删除 ***B*** 节点，调用 ***RemoveChild*** 方法即可。

下面通过一个实例来展示如何创建一个 XML 文档和读取 XML 文档。

XmlSample 页面的 XAML 代码如下：

```
<Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">
    <Grid.RowDefinitions>
        <RowDefinition Height="*"/>
        <RowDefinition Height="auto"/>
    </Grid.RowDefinitions>

    <CommandBar Grid.Row="1">
        <AppBarButton x:Name="SaveXml"  Icon="Save" Label="创建" Click="SaveXml_Click"/>
        <AppBarButton x:Name="LoadXml" Icon="Read" Label="读取" Click="LoadXml_Click"/>
    </CommandBar>

    s<TextBlock Name="XmlTextBlock" TextWrapping="Wrap" Margin="5"/>
</Grid>
```

可用下面的代码创建并保存 XML 文档（函数 ***SaveXml_Click***）：

```
private async void SaveXml_Click(System.Object sender, RoutedEventArgs e)
{
    //创建XmlDocument实例
    XmlDocument doc = new XmlDocument();
    XmlElement root = doc.CreateElement("Books");
    doc.AppendChild(root);

    XmlElement book = doc.CreateElement("book");
    // 设置属性
    book.SetAttribute("ISBN", "978-0321563842");
    book.SetAttribute("Version", "4th Edition");
    XmlText text = doc.CreateTextNode("The C++ Programming Language");
    book.AppendChild(text);

    root.AppendChild(book);
    // 保持到文件
    StorageFolder local = ApplicationData.Current.LocalFolder;
    StorageFile xmlFile = await local.CreateFileAsync("sample.xml", CreationCollisionOption.
        ReplaceExisting);
    if (xmlFile != null)
    {
        await doc.SaveToFileAsync(xmlFile);
        await new MessageDialog("File Saved!").ShowAsync();
```

```
    }
}
```

可用下面的代码读取 XML 文档（函数 ***LoadXml_Click***）：

```
private async void LoadXml_Click(System.Object sender, RoutedEventArgs e)
{
    // 从本地文件中加载XML
    StorageFolder local = ApplicationData.Current.LocalFolder;
    StorageFile xmlFile = await local.GetFileAsync("sample.xml");
    if (xmlFile != null)
    {
        XmlDocument doc = await XmlDocument.LoadFromFileAsync(xmlFile);
        XmlTextBlock.Text = doc.GetXml();
    }
}
```

13.2.3 相关类简介

13.3 JSON 数据处理

JSON（JavaScript Object Notation）即 JavaScript 对象表示方法。可用简单的文本来描述 JavaScript 语言所能识别的对象。

Windows.Data.Json 命名空间下包含一些让开发者能够轻松访问和处理 JSON 格式数据的类型。

- ***IjsonValue*** 接口：对封装 JSON 值的类型进行界定规范。其中 ***Stringify*** 方法可以返回某个 JSON 值的字符串表示形式；
- ***JsonValue*** 类：表示一个简单的 JSON 值，如字符串、数值等。
- ***JsonArray*** 类：表示 JSON 中的数值；
- ***JsonObject*** 类：表示一个单独的 JSON 对象。

下面通过一个实例来展示如何创建一个 JSON 文档和读取 JSON 文档。

JsonSample 页面的 XAML 代码如下：

```
<Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">
    <Grid.RowDefinitions>
        <RowDefinition Height="*"/>
        <RowDefinition Height="*"/>
        <RowDefinition Height="auto"/>
    </Grid.RowDefinitions>

    <CommandBar Grid.Row="2">
        <AppBarButton x:Name="SaveJson"  Icon="Save" Label="创建" Click="SaveJson_Click"/>
        <AppBarButton x:Name="LoadJson" Icon="Read" Label="读取" Click="LoadJson_Click"/>
    </CommandBar>

    <TextBlock Name="SaveJsonTextBlock" Grid.Row="0" TextWrapping="Wrap" Margin="5"/>
```

```
    <TextBlock Name="LoadJsonTextBlock" Grid.Row="1" TextWrapping="Wrap" Margin="5"/>
</Grid>
```

可用下面的代码创建并保存 JSON（函数 ***SaveJson_Click***）

```
private async void SaveJson_Click(object sender, RoutedEventArgs e)
{
    JsonObject obj = new JsonObject();
    obj["Title"] = JsonValue.CreateStringValue("The C++ Programming Language");
    obj["ISBN"] = JsonValue.CreateStringValue("978-0321563842");
    obj["Version"] = JsonValue.CreateStringValue("4th Edition");

    SaveJsonTextBlock.Text = obj.Stringify();

    var local = ApplicationData.Current.LocalFolder;
    var jsonFile = await local.CreateFileAsync("sample.json", CreationCollisionOption.
        ReplaceExisting);
    if (jsonFile != null)
    {
        await FileIO.WriteTextAsync(jsonFile, obj.Stringify());
        await new MessageDialog("File Saved!").ShowAsync();
    }
}
```

可用下面的代码读取 JSON 文档（函数 ***LoadJson_Click***）：

```
private async void LoadJson_Click(object sender, RoutedEventArgs e)
{
    var local = ApplicationData.Current.LocalFolder;
    var jsonFile = await local.GetFileAsync("sample.json");
    if (jsonFile != null)
    {
        LoadJsonTextBlock.Text = await FileIO.ReadTextAsync(jsonFile);
    }
}
```

第十四章　应用设置和数据

应用设置是指 Windows Universal Platform 应用中可由用户自定义的部分，比如应用的颜色等。

应用数据是应用自身创建和管理的数据，主要包含运行时的状态、应用设置、参考内容以及其他设置等。

第十五章　数据库编程

15.1　SQLite 数据库的使用

https://blogs.windows.com/buildingapps/2017/02/06/using-sqlite-databases-uwp-apps/

For many developers, SQLite has become the preferred client-side technology for data storage. It is a server-less, embedded, open-source database engine that satisfies most local data access scenarios. There are numerous advantages that come with its use, many of which are explained in the SQLite about page.

Since the Windows 10 Anniversary Update (Build 14393), SQLite has also shipped as part of the Windows SDK. This means that when you are building your Universal Windows Platform (UWP) app that runs across the different Windows device form factors, you can take advantage of the SDK version of SQLite for local data storage. This comes with some advantages:

- Your application size reduces since you don't download your own SQLite binary and package it as part of your application
  - Note: ***Microsoft.Data.SQLite*** (used in the example below) currently has an issue where both SQLite3.dll and WinSQLite.dll are loaded in memory whenever a .NET Native version of your application is run. This is a tracked issue that will be addressed in subsequent updates of the library.
- You can depend on the Windows team to update the version of SQLite running on the operating system with every release of Windows.
- Application load time has the potential to be faster since the SDK version of SQLite will likely already be loaded in memory.

Below, we provided a quick coding example on how to consume the SDK version of SQLite in your C# application.

Note: Since the Windows SDK version of SQLite has only been available since the Windows 10 Anniversary Update, it can only be used for UWP apps targeting Build 14393 or higher.

15.1.1　C# Example

In this example, we will build a UWP application that will allow users to input text into an app local database. The goal is to provide developers with concise guidance on how to use the SQLite

图 15.1

binary that’s shipped as part of the Windows SDK. Therefore this code sample is meant to be as simple as possible, so as to provide a foundation that can be further built upon.

An example of the end product is shown below:

15.1.2 SQLite C# API Wrappers

As mentioned in the SQLite documentation, the API provided by SQLite is fairly low-level and can add an additional level of complexity for the developer. Because of this, many open-source libraries have been produced to act as wrappers around the core SQLite API. These libraries abstract away a lot of the core details behind SQLite, allowing developers to more directly deal with executing SQL statements and parsing the results. For SQLite consumption across Windows, we recommend the open-source Microsoft.Data.Sqlite library built by the ASP.NET team. It is actively being maintained and provides an intuitive wrapper around the SQLite API. The rest of the example assumes use of the Microsoft.Data.Sqlite library.

Alternative SQLite wrappers are also linked in the “Additional Resources” section below.

15.1.3 Visual Studio set-up

The packages used in this sample have a dependency on NuGet version 3.5 or greater. You can check your version of NuGet by going to Help About Microsoft Visual Studio and looking through the

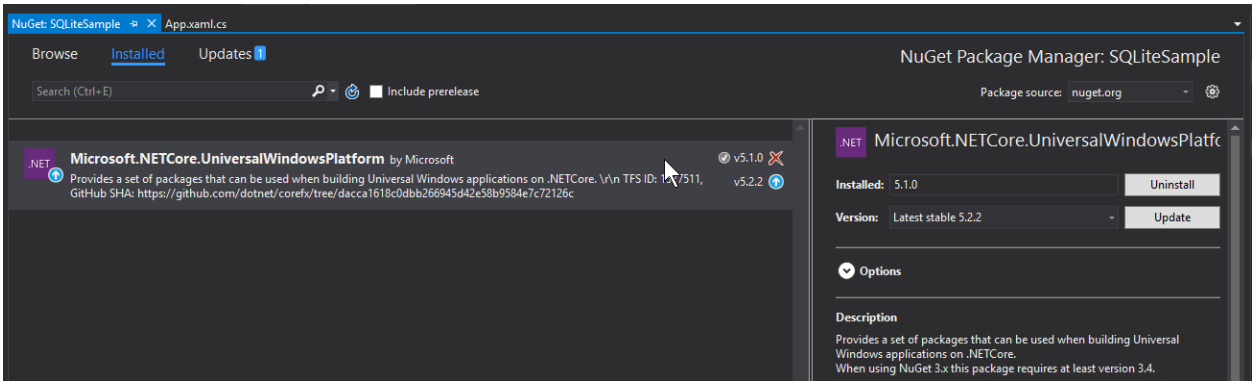

图 15.2

Installed Products for NuGet Package Manager. You can go to the NuGet download page and grab the version 3.5 VSIX update if you have a lower version.

Note: Visual Studio 2015 Update 3 is pre-installed with NuGet version 3.4, and will likely require an upgrade. Visual Studio 2017 RC is installed with NuGet version 4.0, which works fine for this sample.

15.1.3.1 Adding *Microsoft.Data.Sqlite* and upgrading the .NET Core template

The ***Microsoft.Data.Sqlite*** package relies on at least the 5.2.2 version of .NET Core for UWP, so we'll begin by upgrading this:

- Right click on ***References*** → ***Manage NuGet Packages***
- Under the ***Installed*** tab, look for the ***Microsoft.NETCore.UniversalWindowsPlatform*** package and check the version number on the right-hand side. If it's not up to date, you'll be able to update to version 5.2.2 or higher.

Note: Version 5.2.2 is the default for VS 2017 RC. Therefore, this step is not required if you are using this newest version of Visual Studio.

To add the Microsoft.Data.Sqlite NuGet package to your application, follow a similar pattern:

- Right-click on ***References*** → ***Manage NuGet Packages***
- Under the Browse tab, search for the Microsoft.Data.Sqlite package and install it.

15.1.4 Code

15.1.4.1 Application User Interface

We'll start off by making a simple UI for our application so we can see how to add and retrieve entries from our SQLite database.

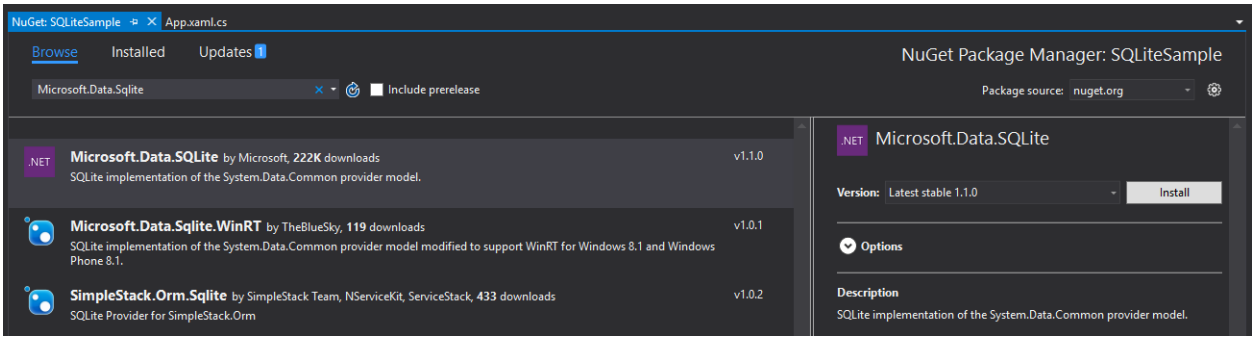

图 15.3: ***Microsoft.Data.Sqlite*** 安装界面

```
<Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">
    <StackPanel>
        <TextBox Name="Input_Box"></TextBox>
        <Button Click="Add_Text">Add</Button>
        <ListView Name="Output">
            <ListView.ItemTemplate>
                <DataTemplate>
                    <TextBlock Text="{Binding}"/>
                </DataTemplate>
            </ListView.ItemTemplate>
        </ListView>
    </StackPanel>
</Grid>
```

There are three important parts to our application’s interface:

1. A text box that allows us to take text from the user.
2. A button linked to an event for pulling the text and placing it in the SQLite database.
3. An ItemTemplate to show previous entries in the database.

15.1.4.2 Code Behind for Application

In the App.xaml.cs and MainPage.xaml.cs files generated by Visual Studio, we’ll start by importing the Microsoft.Data.Sqlite namespaces that we’ll be using.

```
using Microsoft.Data.Sqlite;
using Microsoft.Data.Sqlite.Internal;
```

Then as part of the app constructor, we’ll run a “CREATE TABLE IF NOT EXISTS” command to guarantee that the SQLite .db file and table are created the first time the application is launched.

```
public App()
{
    this.InitializeComponent();
    this.Suspending += OnSuspending;
    SqliteEngine.UseWinSqlite3(); //Configuring library to use SDK version of SQLite
    using (SqliteConnection db = new SqliteConnection("Filename=sqliteSample.db"))
```

```
    {
        db.Open();
        String tableCommand = @"CREATE TABLE IF NOT EXISTS MyTable (
                                    Primary_Key INTEGER PRIMARY KEY AUTOINCREMENT,
                                    Text_Entry NVARCHAR(2048) NULL)";
        SqliteCommand createTable = new SqliteCommand(tableCommand, db);
        try
        {
            createTable.ExecuteReader();
        }
        catch (SqliteException e)
        {
            //Do nothing
        }
    }
}
```

There are couple of points worth noting with this code:

1. We make a call to SqliteEngine.UseWinSqlite3() before making any other SQL calls, which guarantees that the Microsoft.Data.Sqlite framework will use the SDK version of SQLite as opposed to a local version.

2. We then open a connection to a SQLite .db file. The name of the file passed as a String is your choice, but should be consistent across all SqliteConnection objects. This file is created on the fly the first time it's called, and is stored in the application's local data store.

3. After establishing the connection to the database, we instantiate a SqliteCommand object passing in a String representing the specific command and the SqliteConnection instance, and call execute.

4. We place the ExecuteReader() call inside a try-catch block. This is because SQLite will always throw a SqliteException whenever it can't execute the SQL command. Not getting the error confirms that the command went through correctly.

Next, we'll add code in the View's code-behind file to handle the button-clicked event. This will take text from the text box and put it into our SQLite database.

```
private void Add_Text(object sender, RoutedEventArgs e)
{
    using (SqliteConnection db = new SqliteConnection("Filename=sqliteSample.db"))
    {
        db.Open();

        SqliteCommand insertCommand = new SqliteCommand();
        insertCommand.Connection = db;

        //Use parameterized query to prevent SQL injection attacks
        insertCommand.CommandText = "INSERT INTO MyTable VALUES (NULL, @Entry);";
        insertCommand.Parameters.AddWithValue("@Entry", Input_Box.Text);

        try
        {
```

```
            insertCommand.ExecuteReader();
        }
        catch (SqliteException error)
        {
            //Handle error
            return;
        }
        db.Close();
    }
    Output.ItemsSource = Grab_Entries();
}
```

As you can see, this isn't drastically different than the SQLite code explained in the app's constructor above. The only major deviation is the use of parameters in the query so as to prevent SQL injection attacks. You will find that commands that make changes to the database (i.e. creating tables, or inserting entries) will mostly follow the same logic.

Finally, we go to the implementation of the Grab_Entries() method, where we grab all the entries from the Text_Entry column and fill in the XAML template with this information.

```
private List<String> Grab_Entries()
{
    List<String> entries = new List<string>();
    using (SqliteConnection db = new SqliteConnection("Filename=sqliteSample.db"))
    {
        db.Open();
        SqliteCommand selectCommand = new SqliteCommand("SELECT Text_Entry from MyTable", db);
        SqliteDataReader query;

        try
        {
            query = selectCommand.ExecuteReader();
        }
        catch(SqliteException error)
        {
            //Handle error
            return entries;
        }

        while(query.Read())
        {
            entries.Add(query.GetString(0));
        }
        db.Close();
    }
    return entries;
}
```

Here, we take advantage of the SqliteDataReader object returned from the ExecuteReader() method to run through the results and add them to the List we eventually return. There are two methods worth pointing out:

1. The Read() method advances through the rows returned back from the executed SQLite command,

and returns a boolean based on whether you’ ve reached the end of the query or not (True if there are more rows left, and False if you’ ve reached the end).

2. The GetString() method returns the value of the specified column as a String. It takes in one parameter, an int that represents the zero-based column ordinal. There are similar methods like GetDataTime() and GetBoolean() that you can use based on the data type of the column that you are dealing with.

   (a) The ordinal parameter isn’ t as relevant in this example since we are selecting all the entries in a single column. However, in the case where multiple columns are part of the query, the ordinal represents the column you are pulling from. So if we selected both Primary_Key and Text_Entry, then GetString(0) would return the value of Primary_Key String and GetString(1) would return the value of Text_Entry as a String.

And that’ s it! You can now build your application and add any text you like into your SQLite database. You can even close and open your application to see that the data persists.

A link to the full code can be found at: `https://github.com/Microsoft/windows-developer-blog-samples/tree/master/Samples/SQLiteSample`

15.1.4.3 Moving Forward

There are plenty of additions that you can make to tailor this sample to your needs:

1. Adding more tables and more complicated queries.

2. Providing more sanitation over the text entries to prevent faulty user input.

3. Communicating with your database in the cloud to propagate information across devices.

4. And so much more!

15.1.5 What about Entity Framework?

For those developers looking to abstract away particular database details, Microsoft’ s Entity Framework provides a great model that lets you work at the “Object” layer as opposed to the database access layer. You can create models for your database using code, or visually define your model in the EF designer. Then Entity Framework makes it super-easy to generate a database from your defined object model. It’ s also possible to map your models to existing databases you may have already created.

SQLite is one of many database back-ends that Entity Framework is configured to work with. This documentation provides an example to work from.

15.1.6 Conclusion

From embedded applications for Windows 10 IoT Core to a cache for enterprise relations database server (RDBS) data, SQLite is the premier choice for any application that needs local storage support. SQLite’s server-less and self-contained architecture makes it compact and easy to manage, while its tried and tested API surface coupled with its massive community support provides additional ease of use. And since it ships as part of Windows 10, you can have peace of mind, knowing that you’re always using an up-to-date version of the binary.

As always, please leave any questions in the comments section, and we’ll try our best to answer them. Additional resources are also linked below.

第十六章　网络通信

第十七章　传感器与地理信息

第十八章　语言识别技术

附录

18.1 元素周期表

元 素 周 期 表

1 2.20 1s																	2 1s
H 氢 Hydrogen 1.00784–1.00811																	**He** 氖 Helium 4.002602(2)
3 0.98 2s	4 1.57 2s											5 2.04 2p	6 2.55 2p	7 3.04 2p	8 3.44 2p	9 3.98 2p	10 2p
Li 锂 Lithium 6.938–6.997	**Be** 铍 Beryllium 9.012183 1(5)											**B** 硼 Boron 10.806–10.821	**C** 碳 Carbon 12.0096–12.0116	**N** 氮 Nitrogen 14.00643–14.00728	**O** 氧 Oxygen 15.99903–15.99977	**F** 氟 Fluorine 18.998403163(6)	**Ne** 氦 Neon 20.1797(6)
11 0.93 3s	12 1.31 3s											13 1.61 3p	14 1.90 3p	15 2.19 3p	16 2.58 3p	17 3.16 3p	18 3p
Na 钠 Sodium 22.98976928(2)	**Mg** 镁 Magnesium 24.304–24.307											**Al** 铝 Aluminium 26.9815385(7)	**Si** 硅 Silicon 28.084–28.086	**P** 磷 Phosphorus 30.973761998(5)	**S** 硫 Sulphur 32.059–32.076	**Cl** 氯 Chlorine 35.446–35.457	**Ar** 氩 Argon 39.948(1)
19 0.82 4s	20 1.00 4s	21 1.36 3d	22 1.54 3d	23 1.63 3d	24 1.66 3d*	25 1.55 3d	26 1.83 3d	27 1.88 3d	28 1.91 3d	29 1.90 3d*	30 1.65 3d	31 1.81 4p	32 2.01 4p	33 2.18 4p	34 2.55 4p	35 2.96 4p	36 3.00 4p
K 钾 Potassium 39.0983(1)	**Ca** 钙 Calcium 40.078(4)	**Sc** 钪 Scandium 44.955908(5)	**Ti** 钛 Titanium 47.867(1)	**V** 钒 Vanadium 50.9415(1)	**Cr** 铬 Chromium 51.9961(6)	**Mn** 锰 Manganese 54.938044(3)	**Fe** 铁 Iron 55.845(2)	**Co** 钴 Cobalt 58.933194(4)	**Ni** 镍 Nickel 58.6934(4)	**Cu** 铜 Copper 63.546(3)	**Zn** 锌 Zinc 65.38(2)	**Ga** 镓 Gallium 69.723(1)	**Ge** 锗 Germanium 72.630(8)	**As** 砷 Arsenic 74.921595(6)	**Se** 硒 Selenium 78.971(8)	**Br** 溴 Bromine 79.901–79.907	**Kr** 氪 Krypton 83.798(2)
37 0.82 5s	38 0.95 5s	39 1.22 4d	40 1.33 4d	41 1.6 4d*	42 2.16 4d*	43 1.9 4d	44 2.2 4d*	45 2.28 4d*	46 2.20 4d*	47 1.93 4d*	48 1.69 4d	49 1.78 5p	50 1.96 5p	51 2.05 5p	52 2.1 5p	53 2.66 5p	54 2.60 5p
Rb 铷 Rubidium 85.4678(3)	**Sr** 锶 Strontium 87.62(1)	**Y** 钇 Yttrium 88.90584(2)	**Zr** 锆 Zirconium 91.224(2)	**Nb** 铌 Niobium 92.90637(2)	**Mo** 钼 Molybdenum 95.95(1)	**Tc** 锝 Technetium (98)	**Ru** 钌 Ruthenium 101.07(2)	**Rh** 铑 Rhodium 102.90550(2)	**Pd** 钯 Palladium 106.42(1)	**Ag** 银 Silver 107.8682(2)	**Cd** 镉 Cadmium 112.414(4)	**In** 铟 Indium 114.818(1)	**Sn** 锡 Tin 118.710(7)	**Sb** 锑 Antimony 121.760(1)	**Te** 碲 Tellurium 127.60(3)	**I** 碘 Iodine 126.90447(3)	**Xe** 氙 Xenon 131.293(6)
55 0.79 6s	56 0.89 6s	57-71	72 1.3 5d	73 1.5 5d	74 2.36 5d	75 1.9 5d	76 2.2 5d	77 2.20 5d	78 2.28 5d*	79 2.54 5d*	80 2.00 5d	81 1.62 6p	82 1.87 6p	83 2.02 6p	84 2.0 6p	85 2.2 6p	86 2.2 6p
Cs 铯 Cesium 132.9054 5196(6)	**Ba** 钡 Barium 137.327(7)	镧系 Lanthanides	**Hf** 铪 Halfnium 178.49(2)	**Ta** 钽 Tantalum 180.94788(2)	**W** 钨 Tungsten 183.84(1)	**Re** 铼 Rhenium 186.207(1)	**Os** 锇 Osmium 190.23(3)	**Ir** 铱 Iridium 192.217(3)	**Pt** 铂 Platinum 195.084(9)	**Au** 金 Gold 196.966569(5)	**Hg** 汞 Mercury 200.592(3)	**Tl** 铊 Thallium 204.382–204.385	**Pb** 铅 Lead 207.2(1)	**Bi** 铋 Bismuth 208.98040(1)	**Po** 钋 Polonium (209)	**At** 砹 Astatine (210)	**Rn** 氡 Radon (222)
87 0.7 7s	88 0.9 7s	89-103	104 6d	105 6d	106 6d	107 6d	108 6d	109 6d	110 6d	111 6d	112 6d	113 7p	114 7p	115 7p	116 7p	117 7p	118 7p
Fr 钫 Francium (223)	**Ra** 镭 Radium (226)	锕系 Actinides	**Rf** 𨧀 Rutherfordium (261)	**Db** □ Dubnium (268)	**Sg** □ Seaborgium (269)	**Bh** □ Bohrium (270)	**Hs** □ Hassium (269)	**Mt** 鿏 Meitnerium (278)	**Ds** 鐽 Darmstadtium (281)	**Rg** 錀 Roentgenium (282)	**Cn** 鿔 Copernicium (285)	**Nh** 鉨 Nihonium (286)	**Fl** 鈇 Flerovium (289)	**Mc** 镆 Moscovium (289)	**Lv** 鉝 Livermorium (293)	**Ts** Tennessine (294)	**Og** Oganesson (294)

镧系	57 1.1 5d*	58 1.12 4f*	59 1.13 4f	60 1.14 4f	61 1.13 4f	62 1.17 4f	63 1.2 4f	64 1.2 4f*	65 1.1 4f	66 1.22 4f	67 1.23 4f	68 1.24 4f	69 1.25 4f	70 1.1 4f	71 1.27 4f
	La 镧 Lanthanum 138.90547(7)	**Ce** 铈 Cerium 140.116(1)	**Pr** 镨 Praseodymium 140.90766(2)	**Nd** 钕 Neodymium 144.242(3)	**Pm** 钷 Promethium (145)	**Sm** 钐 Samarium 150.36(2)	**Eu** 铕 Europium 151.964(1)	**Gd** 钆 Gadolinium 157.25(3)	**Tb** 铽 Terbium 158.92535(2)	**Dy** 镝 Dysprosium 162.500(1)	**Ho** 钬 Holmium 164.93033(2)	**Er** 铒 Erbium 167.259(3)	**Tm** 铥 Thulium 168.93422(2)	**Yb** 镱 Ytterbium 173.045(10)	**Lu** 镥 Lutetium 174.9668(1)
锕系	89 1.1 6d*	90 1.3 5f*	91 1.5 5f*	92 1.38 5f*	93 1.36 5f*	94 1.28 5f	95 1.13 5f	96 1.28 5f*	97 1.3 5f	98 1.3 5f	99 1.3 5f	100 1.3 5f	101 1.3 5f	102 1.3 5f	103 1.3 5f
	Ac 锕 Actinium (227)	**Th** 钍 Thorium 232.0377(4)	**Pa** 镤 Protactinium 231.03588(2)	**U** 铀 Uranium 238.02891(3)	**Np** 镎 Neptunium (237)	**Pu** 钚 Plutonium (244)	**Am** 镅 Americium (243)	**Cm** 锔 Curium (247)	**Bk** 锫 Berkelium (247)	**Cf** 锎 Californium (251)	**Es** 锿 Einsteinium (252)	**Fm** 镄 Fermium (257)	**Md** 钔 Mendelevium (258)	**No** 锘 Nobelium (259)	**Lr** 铹 Lawrencium (266)

Z = atomic number; eneg = electronegativity; ss = subshell; Sy = Symbol, Name = element name, saw = standard atomic weight

Z eneg ss
Sy
Name
saw

相对原子质量来源：(http://ciaaw.org/atomic-weights.htm). .

An asterisk (*) next to a subshell indicates an anomalous (Aufbau rule-breaking) ground state electron configuration.

18.2　密码子表

密码子表