WebView

Последнее обновление: 14.06.2016

WebView представляет собой мини-веббраузер, позволяющий просматривать различные типы контента:

- В первую очередь WebView позволяет загружать из среды интернет веб-сайты и взаимодействовать с ними. WebView имеет полную поддержку HTML, CSS и JavaScript
- Различные типы документов, например, документы pdf. Однако каждая платформа может поддерживать свой набор типов документов. К примеру, на iOS и Android мы сможем просмотреть файл pdf, а вот на Windows Phone/Windows Mobile нет
- Строки, которые содержат код html, при просмотре будут должным образом интерпретированы, как в обычном браузере
- Локальные файлы - мы можем добавить в проект локальные файлы html и запускать их в WebView

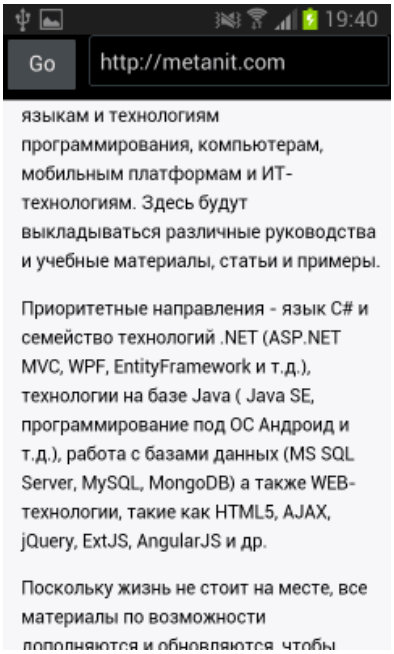
Просмотр данных из интернета

Для загрузки данных нам надо передать свойству **Source** элемента WebView определенный путь:

```
class MainPage : ContentPage
{
    WebView webView;
    Entry urlEntry;
    public MainPage()
    {
        urlEntry = new Entry { HorizontalOptions = LayoutOptions.FillAndExpand };
        Button button = new Button { Text = "Go" };
        button.Clicked += button_Clicked;
        StackLayout stack = new StackLayout
        {
            Orientation = StackOrientation.Horizontal,
            Children = { button, urlEntry}
        };
        webView = new WebView
        {
            Source = new UrlWebViewSource { Url = "http://blog.xamarin.com/"},
            // или так
            // Source = "http://blog.xamarin.com/",
            VerticalOptions = LayoutOptions.FillAndExpand
        };

        this.Content = new StackLayout { Children = {stack, webView }};
    }

    void button_Clicked(object sender, EventArgs e)
    {
        webView.Source = new UrlWebViewSource{Url=urlEntry.Text};
        // или так
        // webView.Source = urlEntry.Text;
    }
}
```



Аналог в xaml:

```
<StackLayout>
  <StackLayout Orientation="Horizontal">
    <Button Text="Go" Clicked="button_Clicked" />
    <Entry x:Name="urlEntry" HorizontalOptions="FillAndExpand" />
  </StackLayout>
  <WebView x:Name="webView" Source="http://blog.xamarin.com" VerticalOptions="FillAndExpand" />
</StackLayout>
```

Примечение для iOS

Начиная с версии iOS 9 система позволяет взаимодействовать только с теми серверами, которые применяют SSL. Для всех остальных серверов, которые не применяют SSL, надо вносить соответствующие значения в файл **Info.plist**.

```
<key>NSAppTransportSecurity</key>
    <dict>
        <key>NSExceptionDomains</key>
        <dict>
            <key>xamarin.com</key>
            <dict>
                <key>NSIncludesSubdomains</key>
                <true/>
                <key>NSTemporaryExceptionAllowsInsecureHTTPLoads</key>
                <true/>
                <key>NSTemporaryExceptionMinimumTLSVersion</key>
                <string>TLSv1.1</string>
            </dict>
        </dict>
    </dict>
```

В данном случае дается разрешение для работы с сайтом xamarin.com. Но если нам вообще надо снять ограничение, то мы можем использовать более универсальную, но менее безопасную настройку:

```
<key>NSAppTransportSecurity</key>
    <dict>
        <key>NSAllowsArbitraryLoads </key>
        <true/>
    </dict>
```

Чтобы внести изменения в файл, надо его открыть как xml-файл в обычном текстовом редакторе.

HTML-строки

```
class MainPage : ContentPage
{
    public MainPage()
    {
        WebView  webView = new WebView();
        var htmlSource = new HtmlWebViewSource();
        htmlSource.Html = @"
                        <h1>Xamarin.Forms</h1>
                        <p>Привет мир!</p>
```

```
                    ";
        webView.Source = htmlSource;
        this.Content = webView;
    }
}
```

Локальные html-файлы

WebView может запускать файлы HTML, CSS и Javascript, которые встроены в приложение. Например, пусть у нас есть такой html-файл *index.html*:

```
<html>
  <head>
    <title>Xamarin Forms</title>
    <meta charset="utf-8" />
    <link href="styles.css" rel="stylesheet">
  </head>
  <body>
    <h1>Xamarin.Forms</h1>
    <p>Привет мир</p>
    <img src="image2.png" />
  </body>
</html>
```

Этот файл использует таблицу стилей из файла styles.css и файл изображения image2.png. Также надо учесть, что данный файл имеет кодировку utf-8.

Теперь чтобы использовать этот и сопутствующие файлы, нам надо добавить их во все три проекта. Но перед добавлением настроеим главный проект. Допустим, у нас все загружается в классе страницы MainPage:

```
class MainPage : ContentPage
{
    public MainPage()
    {
        WebView  webView = new WebView();
        UrlWebViewSource urlSource = new UrlWebViewSource();
        urlSource.Url = System.IO.Path.Combine(DependencyService.Get<IBaseUrl>().Get(), "index.html");
        webView.Source = urlSource;
        this.Content = webView;
    }
}

public interface IBaseUrl { string Get(); }
```

Во-первых, зедсь определяется вспомогательный интерфейс IBaseUrl, через который, используя внедрение зависимостей, мы будем получать путь к фйлу.

Во-вторых, так как нам надо установить путь, мы используем объект **UrlWebViewSource**. А через вызов `DependencyService.Get<IBaseUrl>().Get()` мы будем получать от каждой отдельной платформы ее путь к файлу index.html.

Теперь настроим остальные проекты.

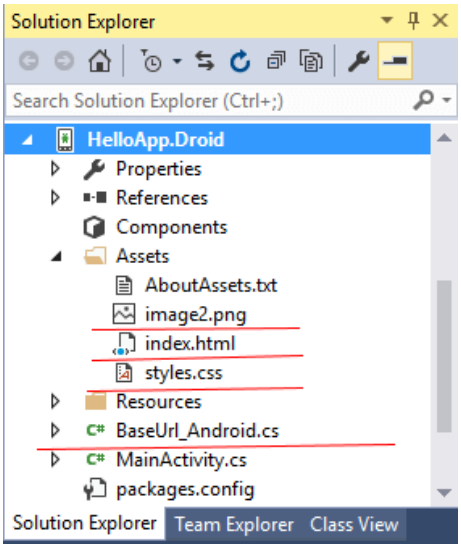
Android

В проект для Android в папку Assets добавим все необходимые файлы html, css, js, файлы изображений. А в сам этот проект добавим дополнительный класс **BaseUrl_Android**:

```
using Xamarin.Forms;
using HelloApp.Droid;

[assembly: Dependency(typeof(BaseUrl_Android))]
namespace HelloApp.Droid    // пространство имен может отличаться
{
    public class BaseUrl_Android : IBaseUrl
    {
        public string Get()
        {
            return "file:///android_asset/";
        }
    }
}
```

То есть здесь мы реализуем интерфейс IBaseUrl и возвращаем путь к папке с файлами.



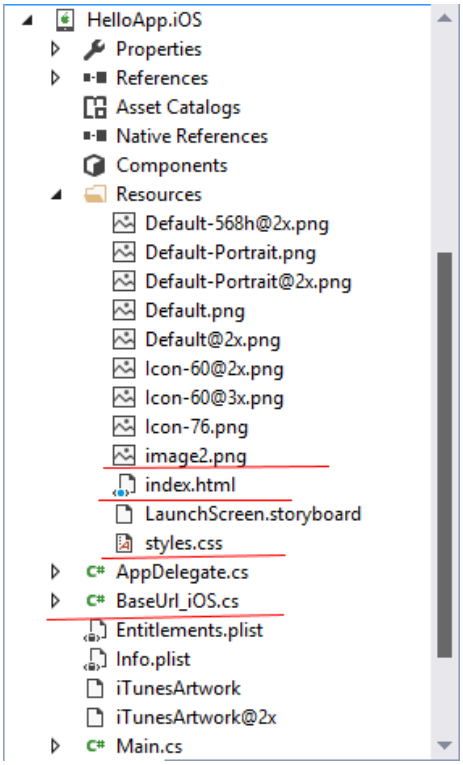
iOS

В проект для iOS аналогично добавляем все используемые файлы в папку Resources. И затем в проект добавляем класс :

```
using Foundation;
using HelloApp.iOS;
using Xamarin.Forms;

[assembly: Dependency(typeof(BaseUrl_iOS))]
namespace HelloApp.iOS
{
    public class BaseUrl_iOS : IBaseUrl
    {
        public string Get()
        {
            return NSBundle.MainBundle.BundlePath;
        }
    }
}
```

Опять же это реализация интерфейса IBaseUrl, которая возвращает путь к файлам.

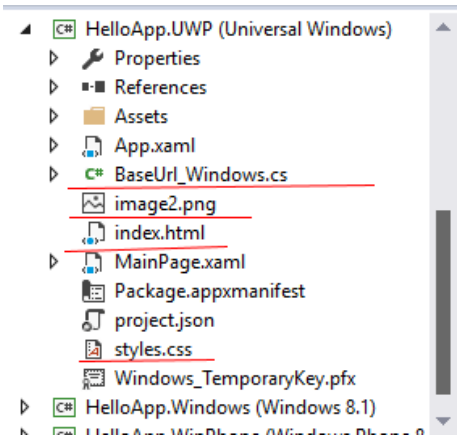


UWP

Далее в проект для UWP также добавляем все необходимые файлы, можно прямо в корень проекта, можно специально создать для них подкаталг. И затем добавляем в проект новый класс **BaseUrl_Windows**:

```
using HelloApp.UWP;
using Xamarin.Forms;

[assembly: Dependency(typeof(BaseUrl_Windows))]
namespace HelloApp.UWP
{
    public class BaseUrl_Windows : IBaseUrl
    {
        public string Get()
        {
            return "ms-appx-web:///";
        }
    }
}
```



Теперь вне зависимости от типа устройства главный проект будет получать корректный путь и должным образом устанавливать источник ресурса для WebView:

Назад Содержание Вперед