



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

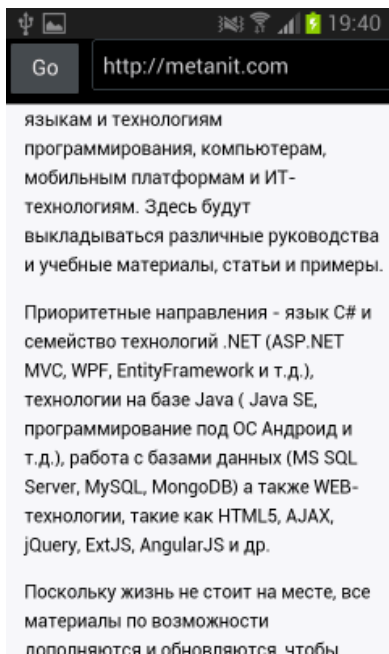
Просмотр данных из интернета

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>
```

Примечание 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;
    }
}

```



Xamarin.Forms

Привет мир!

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>
  
</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 настроим 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, здесь определяется вспомогательный интерфейс *IBaseUrl*, через который, используя внедрение зависимостей, мы будем получать путь к файлу.

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.

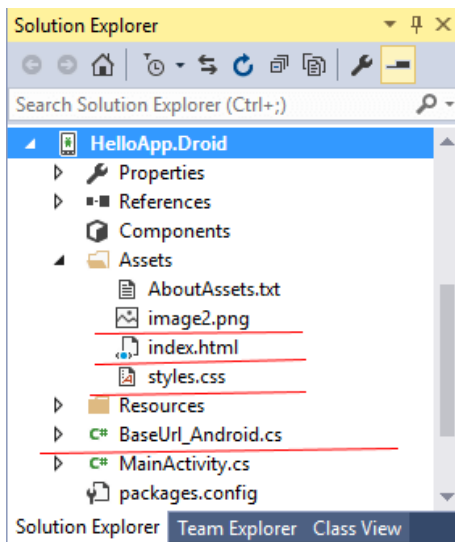
Android

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.



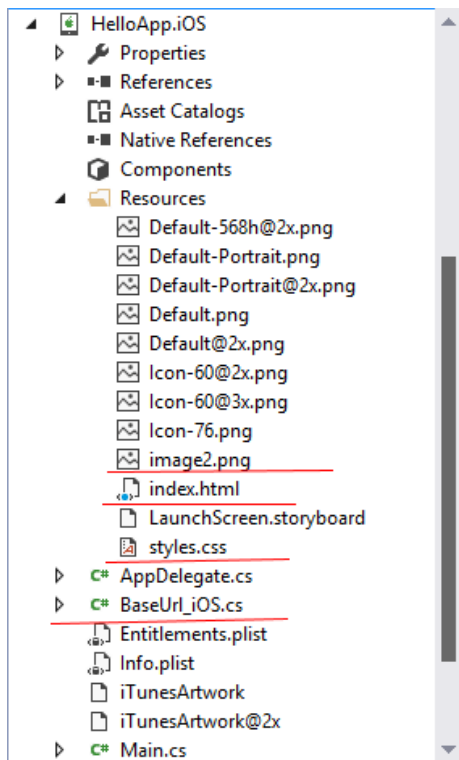
iOS

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.

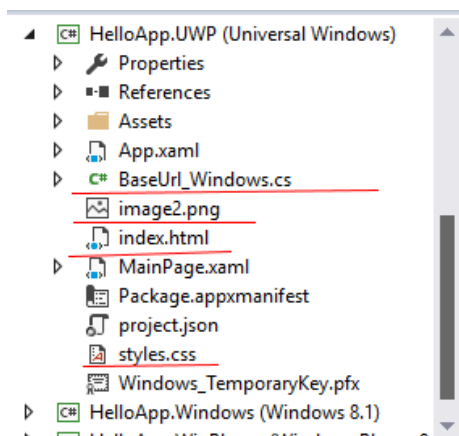


UWP

Далее в проект для UWP также добавляем все необходимые файлы, можно прямо в корень проекта, можно специально создать для них подкаталог. 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:



Xamarin.Forms

Привет мир



Нужен анализ контекста?

Аудит контекстной рекламы. Анализ настроек, рекомендации. От 150 рублей. Звоните! gusarov-group.by



[Назад](#) [Содержание](#) [Вперед](#)



Гибкий кабель-канал

 promtechnolog.ru



Яндекс.Директ

Сильный мороз в Минской области?

 yandex.by



Яндекс.Директ

5 Комментариев metanit.com

1 Войти ▾

♥ Рекомендовать  Поделиться

Лучшее в начале ▾



Присоединиться к обсуждению...

ВОЙТИ С ПОМОЩЬЮ

ИЛИ ЧЕРЕЗ DISQUS **Андрей Петров** • 9 месяцев назад

WebView использует возможности только встроенного в устройство браузера? т.е. если браузер в силу своих ограничений не способен открывать какие-либо страницы (касается WP 8.1) то и WebView не сможет этого сделать?

  • Ответить • Поделиться ›**Metanit** Модератор ➔ Андрей Петров • 9 месяцев назад

да, WebView использует встроенные возможности системы, отсюда и ограничения на различных платформах

  • Ответить • Поделиться ›**Андрей** • год назад

получаю ошибку что страница не доступна, файл по пути file:///android_asset/file.png не найден

  • Ответить • Поделиться ›**Сергій Степанович** ➔ Андрей • год назад

У Вас ошибка в написании android

  • Ответить • Поделиться ›**Volodymyr** • 2 года назад

Возник баг в ios. WebView использую для промотра документов pdf. Внизу разместилась клавиша для печати документа. Иногда документ показывается не на весь размер экрана, а в уменьшенном размере. Баг выскакивает случайно. Причину установить не получается.

  • Ответить • Поделиться ›

ТАКЖЕ НА METANIT.COM

C++ | Динамические массивы

4 комментария • 2 месяца назад



Metanit — в данном случае вы делаете ошибку, что переносите опыт работы с одним языком на другой. В разных языках может быть разная терминология. В C++

Kotlin | Руководство

2 комментария • месяц назад



Артур Голояд — Просто шикарно, может подкинете материалов по работе с сетями на Kotlin?

ASP.NET Core | SignalR Core. Первое приложение

25 комментариев • 4 месяца назад



Metanit — помещаете код из index.html в представление и

Аватарсе

Angular в ASP.NET Core | Первый проект

33 комментария • 2 месяца назад



Zaur YakubOff — Проблема вот в чем оказалось. Прошу прощения за занудство

```
app.UseWebpackDevMiddleware(new ...
```



[Вконтакте](#) | [Twitter](#) | [Google+](#) | [Канал сайта на youtube](#) | [Помощь сайту](#)

Контакты для связи: metanit22@mail.ru

Copyright © metanit.com, 2012-2017. Все права защищены.