DependencyService

Последнее обновление: 17.09.2016

Механизм **DependencyService** представляет еще один способ использования платформо-зависимых вызовов в кроссплатформенном приложении на Xamarin Forms. Иногда нам приходится задействовать те классы, которые доступны для определенной платформы. И в этом случае мы можем определить набор методов, реализация которых будет различаться на разных платформах. А затем с помощью DependencyService применить эти методы в Portable-проекте.

Чтобы начать использовать DependencyService, нам надо определить интерфейс в Portable-проекте. Этот интерфейс будет определять сигнатуру методов, реализация которых будет зависеть от конкретной платформы. К примеру, нам надо вывести номер версии операционной системы. Для этого добавим в Portable-проект следующий интерфейс:

```
public interface IDeviceInfo
{
    string GetInfo();
}
```

Также в Portable-проект добавим класс страницы, которую назовем MainPage:

```
using Xamarin.Forms;

namespace DIApp
{
    public class MainPage : ContentPage
    {
        public MainPage()
        {
            IDeviceInfo deviceInfo = DependencyService.Get<IDeviceInfo>();

            Label infoLabel = new Label();
            infoLabel.Text = deviceInfo.GetInfo();
            infoLabel.FontSize = Device.GetNamedSize(NamedSize.Large, typeof(Label));

            Content = new StackLayout
            {
                Children = { infoLabel}
            };
        }
    }
}
```

Для получения информации об устройстве здесь используется вызов:

```
IDeviceInfo deviceInfo = DependencyService.Get<IDeviceInfo>();
```

Здесь с помощью метода `Get()` мы получаем объект IDeviceInfo. При этом не важно, что в Portable-проекте нет конкретной реализации данного интерфейса. В качестве реализации будет использоваться объект класса, который будет определен в проекте для текущей платформы.

Но естественно, если мы сейчас запустим проект, то получим ошибку.

И теперь в каждом проекте для конкретной ОС нам надо определить реализацию этого интерфейса. Так, добавим в проект для Android новый класс **DeviceInfo**:

```
using Android.OS;
using Xamarin.Forms;
```

```
[assembly: Dependency(typeof(DIApp.Droid.DeviceInfo))]
namespace DIApp.Droid
{
    public class DeviceInfo : IDeviceInfo
    {
        public string GetInfo()
        {
            return $"Android {Build.VERSION.Release.ToString()}";
        }
    }
}
```

Для получения версии системы здесь применяется класс `Android.OS.Build`, который доступен нам только на Android.

Кроме того, над пространством имен нам надо указать специальный атрибут:

```
[assembly: Dependency(typeof(DIApp.Droid.DeviceInfo))]
```

Данный атрибут представлен встроенным классом **DependencyAttribute** из пространства имен `Xamarin.Forms`. Он используется в связке с классом DependencyService. В качестве параметра в конструктор атрибута передается тип объекта, который будет доступен для доступа из Portable-проекта, ну то есть класс DeviceInfo в нашем случае.

Подобным образом добавим новый класс DeviceInfo в проект для iOS:

```
using UIKit;
using Xamarin.Forms;

[assembly: Dependency(typeof(DIApp.iOS.DeviceInfo))]
namespace DIApp.iOS
{
    public class DeviceInfo : IDeviceInfo
    {
        public string GetInfo()
        {
            UIDevice device = new UIDevice();
            return $"{device.SystemName} {device.SystemVersion}";
        }
    }
}
```

Во многом аналогичный класс, как и на Android, только теперь для получения информации об устройстве применяется класс `UIKit.UIDevice`, который доступен только для iOS.

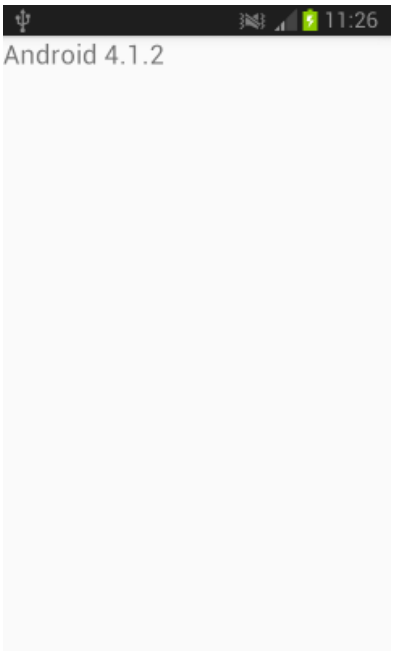
И далее в проект для UWP добавим также новый класс DeviceInfo:

```
using Windows.Security.ExchangeActiveSyncProvisioning;
using Windows.System.Profile;
using Xamarin.Forms;

[assembly: Dependency(typeof(DIApp.UWP.DeviceInfo))]
namespace DIApp.UWP
{
    public class DeviceInfo : IDeviceInfo
    {
        public string GetInfo()
        {
            EasClientDeviceInformation devInfo = new EasClientDeviceInformation();

            var deviceFamilyVersion = AnalyticsInfo.VersionInfo.DeviceFamilyVersion;
            var version = ulong.Parse(deviceFamilyVersion);
            var majorVersion = (version & 0xFFFF000000000000L) >> 48;
            var minorVersion = (version & 0x0000FFFF00000000L) >> 32;
            var buildVersion = (version & 0x00000000FFFF0000L) >> 16;
            var revisionVersion = (version & 0x000000000000FFFFL);
            var systemVersion = $"{majorVersion}.{minorVersion}.{buildVersion}.{revisionVersion}";

            return $"{devInfo.OperatingSystem} {systemVersion}";
        }
    }
}
```

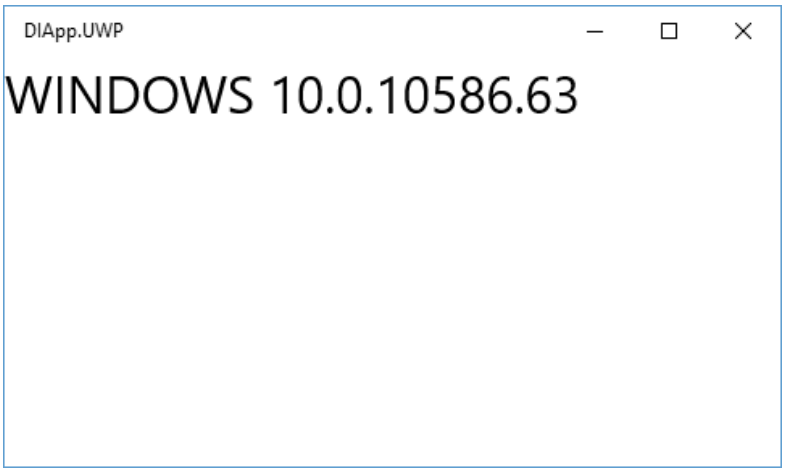
Для получения полной версии сборки системы здесь применяются более сложные преобразования и совсем другие классы.

Таким образом, чтобы получить элементарную информацию об системе, мы вынуждены применять разные классы на разных платформах. Однако DependencyService позволяет с помощью интерфейса IDeviceInfo привести все эти классы общему знаменателю.

И после определения всех классов мы можем запустить проект, и для разных платформ мы получим разные результаты. Например, для Android:



А для проекта UWP будут другие значения:



Назад **Содержание** **Вперед**

nepsy • 6 месяцев назад
Подскажите, о каких файлах идет речь при добавление интерфейсов?
• Ответить • Поделиться ›

Metanit Модератор nepsy • 6 месяцев назад
файлы называются по имени интерфейса или класса
• Ответить • Поделиться ›