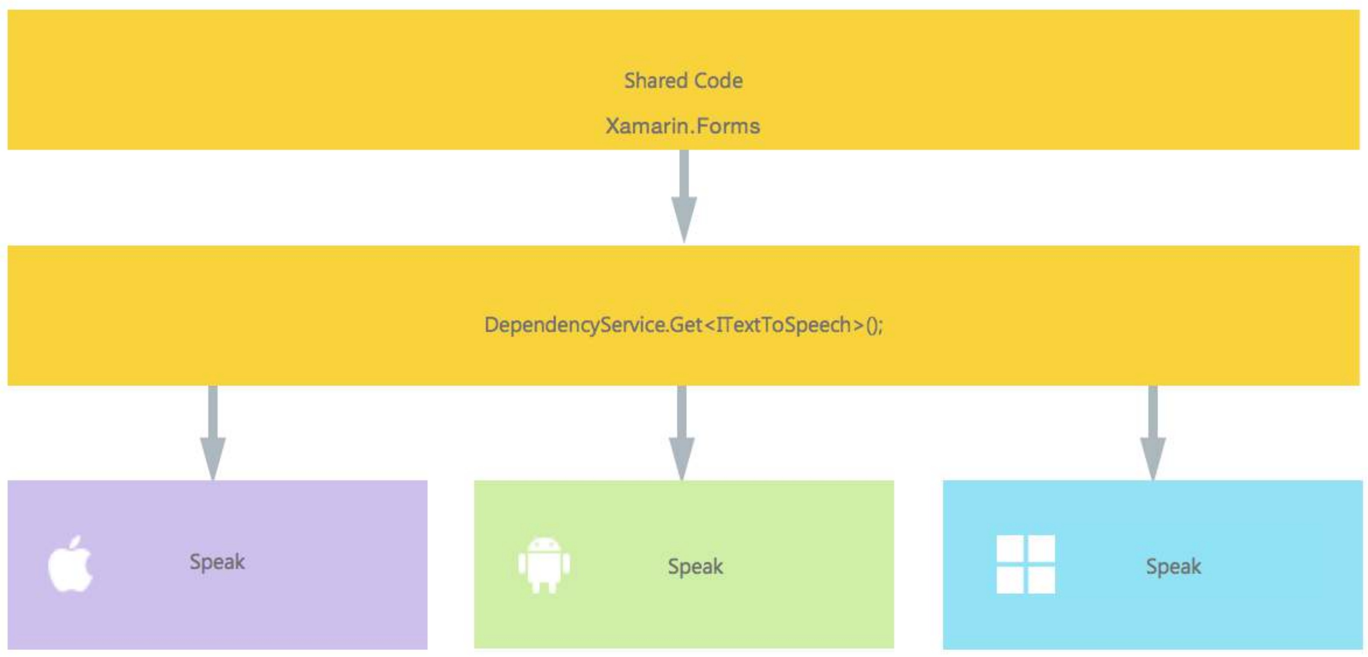
Chapter 11: Accessing native features with DependencyService

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"),
            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
}
```

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;
}
```

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>
        /// 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()
    }
}
```