Chapter 12: DependencyService

Section 12.1: Android implementation

The Android specific implementation is a bit more complex because it forces you to inherit from a native `Java.Lang.Object` and forces you to implement the `IOnInitListener` interface. Android requires you to provide a valid Android context for a lot of the SDK methods it exposes. Xamarin.Forms exposes a `Forms.Context` object that provides you with a Android context that you can use in such cases.

```
using Android.Speech.Tts;
using Xamarin.Forms;
using System.Collections.Generic;
using DependencyServiceSample.Droid;

public class TextToSpeechAndroid : Java.Lang.Object, ITextToSpeech, TextToSpeech.IOnInitListener
{
    TextToSpeech _speaker;

    public TextToSpeechAndroid () {}

    public void Speak (string whatToSay)
    {
        var ctx = Forms.Context;

        if (_speaker == null)
        {
            _speaker = new TextToSpeech (ctx, this);
        }
        else
        {
            var p = new Dictionary<string,string> ();
            _speaker.Speak (whatToSay, 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
}
```

When you've created your class you need to enable the `DependencyService` to discover it at run time. This is done by adding an `[assembly]` attribute above the class definition and outside of any namespace definitions.

```
using Android.Speech.Tts;
using Xamarin.Forms;
using System.Collections.Generic;
using DependencyServiceSample.Droid;
```

```
[assembly: Xamarin.Forms.Dependency (typeof (TextToSpeechAndroid))]
namespace DependencyServiceSample.Droid {
    ...
```

This attribute registers the class with the `DependencyService` so it can be used when an instance of the `ITextToSpeech` interface is needed.

Section 12.2: Interface

The interface defines the behaviour that you want to expose through the `DependencyService`. One example usage of a `DependencyService` is a Text-To-Speech service. There is currently no abstraction for this feature in Xamarin.Forms, so you need to create your own. Start off by defining an interface for the behaviour:

```
public interface ITextToSpeech
{
       void Speak (string whatToSay);
}
```

Because we define our interface we can code against it from our shared code.

Note: Classes that implement the interface need to have a parameterless constructor to work with the `DependencyService`.

Section 12.3: iOS implementation

The interface you defined needs to be implemented in every targeted platform. For iOS this is done through the `AVFoundation` framework. The following implementation of the `ITextToSpeech` interface handles speaking a given text in English.

```
using AVFoundation;

public class TextToSpeechiOS : ITextToSpeech
{
    public TextToSpeechiOS () {}

    public void Speak (string whatToSay)
    {
        var speechSynthesizer = new AVSpeechSynthesizer ();

        var speechUtterance = new AVSpeechUtterance (whatToSay) {
            Rate = AVSpeechUtterance.MaximumSpeechRate/4,
            Voice = AVSpeechSynthesisVoice.FromLanguage ("en-US"),
            Volume = 0.5f,
            PitchMultiplier = 1.0f
        };

        speechSynthesizer.SpeakUtterance (speechUtterance);
    }
}
```

When you've created your class you need to enable the `DependencyService` to discover it at run time. This is done by adding an `[assembly]` attribute above the class definition and outside of any namespace definitions.

```
using AVFoundation;
using DependencyServiceSample.iOS;

[assembly: Xamarin.Forms.Dependency (typeof (TextToSpeechiOS))]
```

```
namespace DependencyServiceSample.iOS {
    public class TextToSpeechiOS : ITextToSpeech
...
```

This attribute registers the class with the `DependencyService` so it can be used when an instance of the `ITextToSpeech` interface is needed.

Section 12.4: Shared code

After you've created and registered your platform-specific classes you can start hooking them up to your shared code. The following page contains a button that triggers the text-to-speech functionality using a pre-defined sentence. It uses `DependencyService` to retrieve a platform-specific implementation of `ITextToSpeech` at run time using the native SDKs.

```
public MainPage ()
{
    var speakButton = new Button {
        Text = "Talk to me baby!",
        VerticalOptions = LayoutOptions.CenterAndExpand,
        HorizontalOptions = LayoutOptions.CenterAndExpand,
    };

    speakButton.Clicked += (sender, e) => {
        DependencyService.Get<ITextToSpeech>().Speak("Xamarin Forms likes eating cake by the
ocean.");
    };

    Content = speakButton;
}
```

When you run this application on an iOS or Android device and tap the button you will hear the application speak the given sentence.