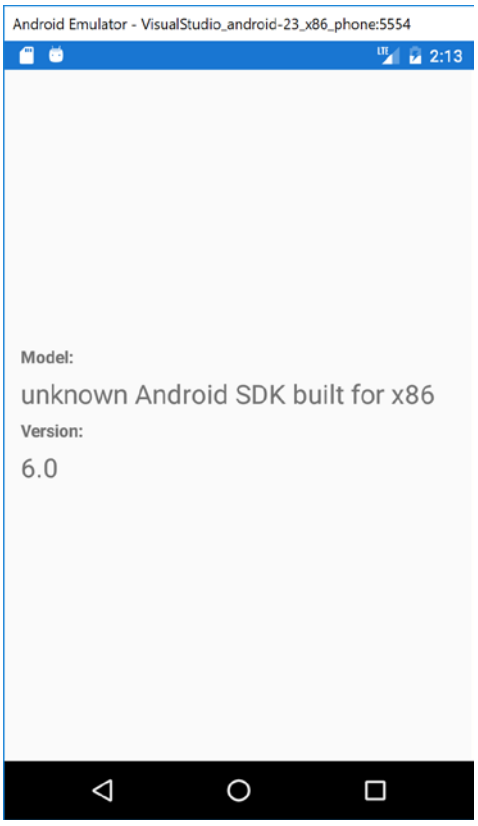
Demo's – Platform specific API calls

General

DEMO LOCATION: https://github.com/HowestNMCT/week8_Demo

.The goal of using Xamarin.Forms is obviously to make as much code common for all platforms (Android/iOS/UWP) as possible. However, sooner or later you might feel the need to make a small part platform specific, because there is no common way to do it in a common way using Xamarin.Forms.

In this demo, we will **ask for model and version** information of the device's OS the app is running on. The result looks like the screenshot below. While we can keep a common XAML in our PCL project, asking for the model and version will be different for each platform.



Android Emulator - VisualStudio_android-23_x86_phone:5554

2:13

Model:

unknown Android SDK built for x86

Version:

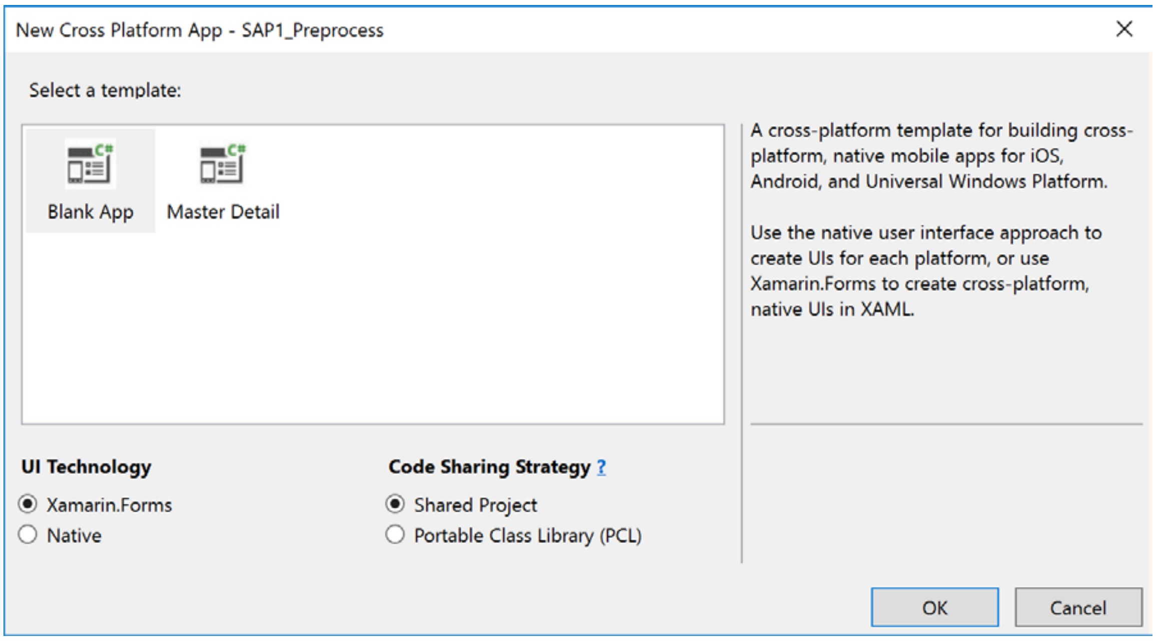
6.0

You will see 3 ways on how to do this. **During the labs of this course, we will only make use of the third option, being the use of a PCL project.**

Option 1: SAP project – preprocessing

DEMO PROJECT: SAP1_Preprocess

- We create a Xamarin.Forms project with SAP (Shared Project) as Code Sharing Strategy. *(see theory for more information on this)*



New Cross Platform App - SAP1_Preprocess

Select a template:

A cross-platform template for building cross-platform, native mobile apps for iOS, Android, and Universal Windows Platform.

Use the native user interface approach to create UIs for each platform, or use Xamarin.Forms to create cross-platform, native UIs in XAML.

UI Technology

(●) Xamarin.Forms
( ) Native

Code Sharing Strategy ?

(●) Shared Project
( ) Portable Class Library (PCL)

- Next, we can write platform specific code **inside our SAP project** using preprocessing:
  #if   #elif   #endif
  Note that the platform specific projects remain untouched.
- You can find example code in the demo in file 'MainPage.xaml.cs'

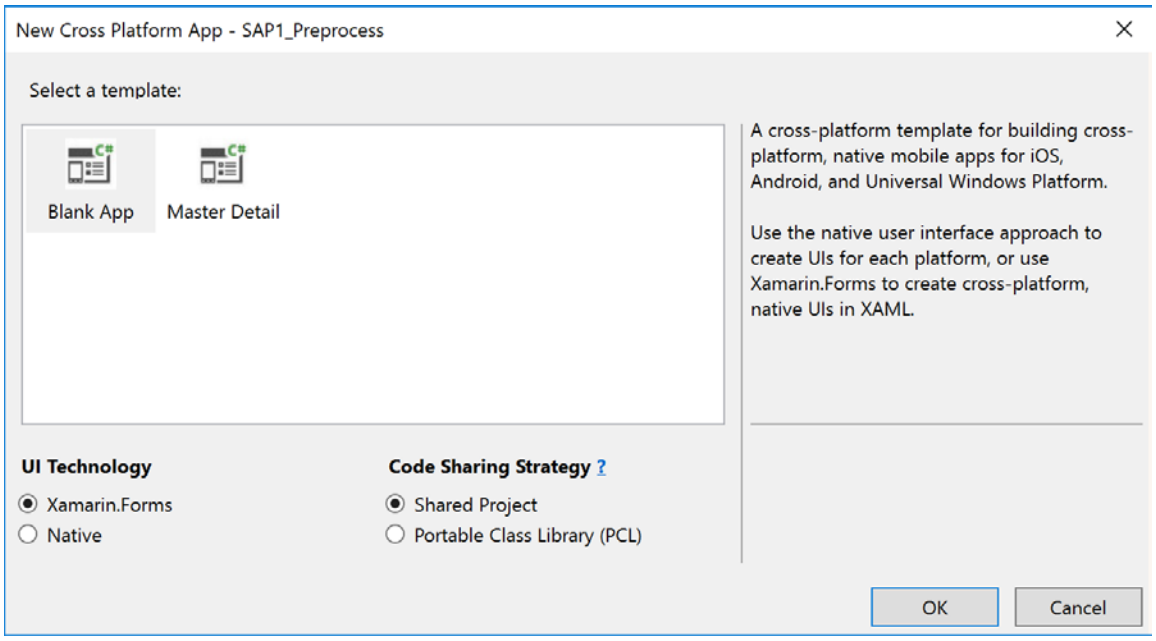
⇨ At first, this looks like the most straightforward way to do platform specific API calls in Xamarin.Forms. However, your code becomes unstructured and chaotic very fast. Also, we prefer a PCL project over an SAP project.

⇨ This is the least preferable option of all 3.

Option 2: SAP project – parallel classes

DEMO PROJECT: SAP2_ParallelClass

- We create a Xamarin.Forms project with SAP (Shared Project) as Code Sharing Strategy.
  (see theory for more information on this)



New Cross Platform App - SAP1_Preprocess

Select a template:

A cross-platform template for building cross-platform, native mobile apps for iOS, Android, and Universal Windows Platform.

Use the native user interface approach to create UIs for each platform, or use Xamarin.Forms to create cross-platform, native UIs in XAML.

UI Technology

● Xamarin.Forms
○ Native

Code Sharing Strategy ?

● Shared Project
○ Portable Class Library (PCL)

- Next, we create a class **inside each platform specific project** (.Android, .iOS, .UWP).
  This class must:
  - Have **exactly the same name** in all three projects
  - Preferably be in the **same namespace as your SAP project** in all three projects.
    As you know by now, the namespace when you create a class inside a platform specific project will be `<projectname>.<platform>`*, eg.:* `SAP2_ParallelClass.Droid`.
    You should change it so it matches the SAP project's namespace, eg. `SAP2_ParallelClass`
  - Have **exactly the same function headers**.
  - The class **musn't be present in the SAP project**.
- In the project, you can find the above in the following projects:
  - In each platform specific project, you can find the class PlatformInfo.
  - Note that there is **not** a PlatformInfo class inside the SAP project!
- After you create the situation as described above, you can simply use the class you created inside your SAP project, just as if it were actually present inside that very project.
  - Find example code in the demo for this in MainPage.xaml.cs

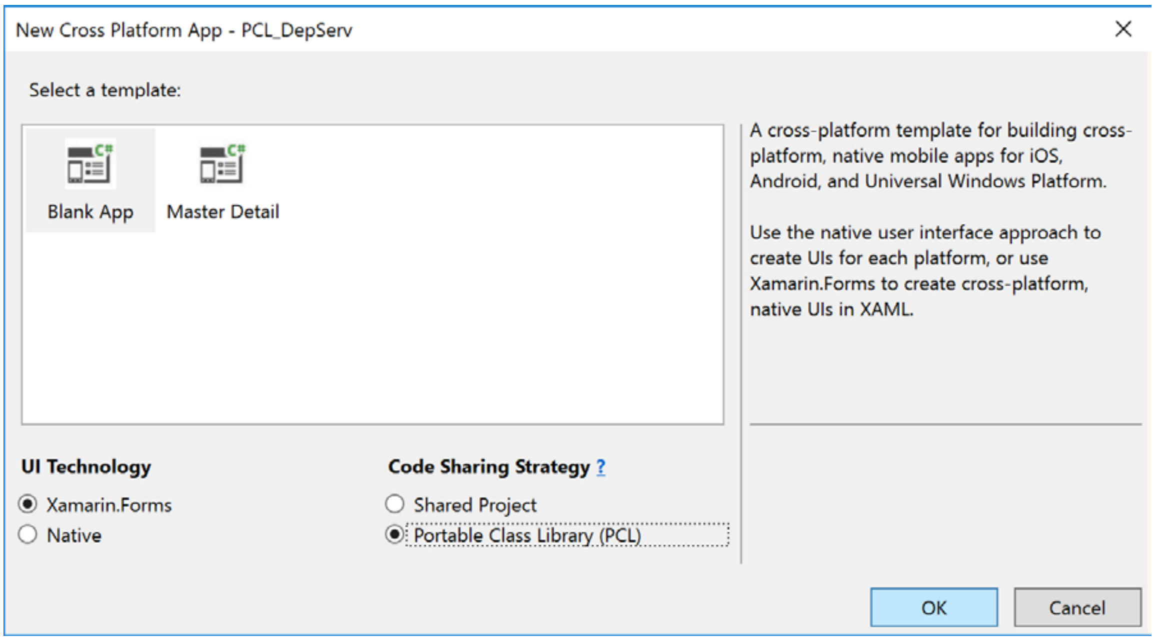
⇨ This looks a lot more structured than in the first option. However, there is a great chance that you make a mistake along the way in namespaces, class names and/or function headers. Also, we do still prefer a PCL project over an SAP project.

⇨ Of the two options in SAP Code Sharing Strategies, this is the best one.

Option 3: SAP project – parallel classes

DEMO PROJECT: PCL_DepServ

- We create a Xamarin.Forms project with PCL (Portable Class Library) as Code Sharing Strategy. *(see theory for more information on this)*


New Cross Platform App - PCL_DepServ

Select a template:

Blank App · Master Detail

A cross-platform template for building cross-platform, native mobile apps for iOS, Android, and Universal Windows Platform.

Use the native user interface approach to create UIs for each platform, or use Xamarin.Forms to create cross-platform, native UIs in XAML.

UI Technology: [✓ Xamarin.Forms] Native

Code Sharing Strategy ?: Shared Project [✓ Portable Class Library (PCL)]

OK · Cancel

- Now, we secretly hope that when reading option 2, you asked yourself the question “Hey? Why don’t we make an interface for that, to at least force the function headers to be the same?” . While this wasn’t possible in option 2 (see theory on why), we will be able to do this here.
- We create an **interface inside the PCL project**. (eg.: in this case we need two functions; one returning the model and another the version)
- We **implement this interface in each platform specific project**.
  - The **only requirement** is that each class **implements the interface, and**
  - **that you make the class available for the DependencyService**, by adding (inside class, above namespace)**:**
    `[assembly: Dependency(typeof(PCL_DepServ.Droid.DroidInfo))]`
  - ⇨ You can choose **whichever class name you like** for the class you create
  - ⇨ The class can **remain inside its own namespace** (you do **not** have to change the default created namespace, eg. PCL_DepServ.Droid)

- In the demo, you can find sample classes of the above:

IPlatformInfo	(inside PCL_DepServ)
DroidInfo	(inside PCL_DepServ.Android)
AppleInfo	(inside PCL_DepServ.iOS)
PlatformInfo	(inside PCL_DepServ.UWP)

- Next, you can use the **DependencyService** (reflection) in combination with the interface you created to get and use the class inside your PCL project (see MainPage.xaml.cs):

```
IPlatformInfo info = DependencyService.Get<IPlatformInfo>();
```

⇨ **This is the option we will use throughout this course. It’s the most structured and clear one.**