Zusammenfassung der Arbeit
Abstract of Thesis

Fachbereich: Department:	**Electrical Engineering and Computer Science**
Studiengang: University course:	**Information Technology**
Thema: Subject:	**Investigation and Test of Xamarin.Essentials in Visual Studio on iPhone**

Zusammenfassung:
Abstract:

The purpose of this thesis is to investigate the possibilities and limitations of a cross platform developing tool Xamarin.Essentials. This thesis focuses on the performance on iOS devices. Furthermore, comparisons of the behavior of the APIs on different platforms are also provided. A demonstration program for the APIs is developed. This package reduces the workload of cross platform developers to a large extent. However, it does not provide internal views for some functions. Besides, it contains some redundant functions and lacks some fundamental functions. Despite the drawbacks and limitations of some functions, Xamarin.Essentials proves to be an effective cross platform development tool.

Verfasser: Author:	**Yuxuan Wang**
Betreuender Professor/in: Attending Professor:	**Prof. Dr. Jörg Bayerlein**
WS / SS:	**SS <2019>**

Table of Contents

1 Introduction

1.1 Description

This thesis aims to explore the possibilities and limitation of Xamarin.Essentials. This thesis mainly discusses the lower half APIs on the API list of Xamarin.Essentials (see the full list on the documentation page of Microsoft: [1]). Introduction to the lower half of the APIs and implementation approaches are given. Also, a comparison of the performance on different platforms is made. In addition, some special problems and limitations of the upper half APIs on iOS devices are discussed, as well.

The introduction and implementation approaches of the upper half APIs and limitations of the lower half APIs are included in another thesis written by Qin LU.

1.2 Motivation

There are several reasons that can explain why this thesis chooses to research on Xamarin.Essentials.

Firstly, Xamarin.Essentials provides a list of cross platform APIs that support UWP, Andriod, and iOS devices. The developers only need to program once for all three platforms. This could save time for development on a large scale.

Secondly, the documentation of Xamarin.Essentials given by Microsoft is not desirably detailed. This thesis can proceed further into Xamarin.Essentials and can hopefully serve as better documentation for Xamarin.Essentials.

1.3 Structure of Thesis

This thesis is composed of 6 main sections.

In section 2, the background information for this thesis is provided. Besides, testing devices are listed.

In section 3, implementing approaches for both hardware and software are given.

In section 4, the lower half APIs of Xamarin.Essentials are examined. In each subsection, introductions, implementation approaches, and comparisons of them are provided. At the end of this section, an additional API is implemented.

In section 5, limitations of the upper half APIs on iOS devices are briefly discussed.

In section 6 and section 7, a conclusion is drawn and several pieces of advice for further researches are given.

2 Background Information

Background information needed for this thesis is provided in this section. Meanwhile, the reasons to choose certain tools are also given. Necessary equipment for testing is listed at the end of this section.

2.1 Visual Studio 2017 Professional

Visual Studio is an integrated development environment (IDE) released by Microsoft, which enables its users to write code in various programming languages. [2] Among these programming languages, C# supports Xamarin, a cross platform tool which is the main technology used in this project. Visual Studio includes necessary and convenient features to build a project: automatic code completion, source control, etc. [2] Visual Studio 2017 has 3 editions: Community, Professional, and Enterprise. [3] Enterprise edition has better tools regarding cross platform development. However, the Enterprise edition is far too expensive for an academic research project. Besides, Technische Hochschule Lübeck has a school license for Professional edition. Thus, Visual Studio 2017 Professional is chosen.

2.2 C# and Xamarin.Essentials

C# is an object oriented programming language. C# shares many features with Java and other C family programming languages. [4] Xamarin.Essentials is a package that provides various APIs that give access to certain functions on UWP, Andriod, and iOS devices. [1] With these APIs provided by Xamarin.Essentials and other features of Xamarin, the same application on three platforms can "share up to 90% of their code". [5]

Xamarin.Essentials supports C# and F#. As mentioned above, Java programmers, such as the author, are likely to be familiar with C#. Thus, C# is chosen in this project.

2.3 Xcode

Xcode is an IDE on macOS used to develop applications on Apple devices. [6] The latest version is Xcode 10.2. [7] In this project, Xcode is irrelevant to the programming part. It is only used to build the application on iOS devices with the code passed by a Windows computer. This Windows computer should have network access to the mac on which Xcode is installed. Though it might be possible to program directly on mac with Visual Studio for Mac, in which case the Windows computer is not needed, compatibility of Windows has made programming on Windows a better choice.

2.4 Used Devices for Testing

The appendix program is tested on Android, iOS, UWP platforms. The testing devices involve iPhone 6, Xiaomi 5, Surface Pro 4, iPad 6, Raspberry Pi, and some Windows computers.

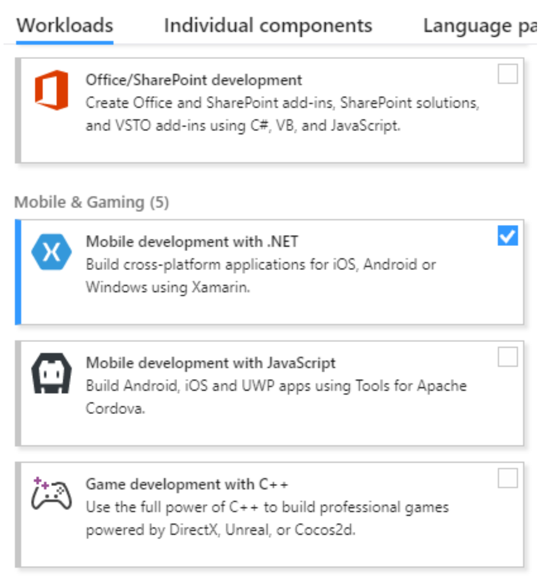
A computer that runs on Windows 10 can also run the appendix program similar to a virtual machine, although lots of sensors are missing on a traditional computer. Surface Pro 4 can be viewed as a mobile device, though some sensors are still missing. Details can be found in section 4. On the Raspberry Pi, Windows IoT, which is designed for embedded systems, is installed. Consequentially, some of the functions of the appendix program should also work on the Raspberry Pi.

3 Installation and Configuration

This section explains the procedure for installing the needed software and for connecting the hardware.

3.1 Visual Studio 2017 Professional and Xamarin.Essentials

The installation pack can be found in the download page of Microsoft Visual Studio: [8]. During the installation, *Mobile Development with .NET* under the tag *Workloads* should be selected as shown in Figure 3.1. Also, Xamarin components should be selected under the tag *Individual components* as shown in Figure 3.2. [9]



Workloads | Individual components | Language pa

☐ Office/SharePoint development
Create Office and SharePoint add-ins, SharePoint solutions, and VSTO add-ins using C#, VB, and JavaScript.

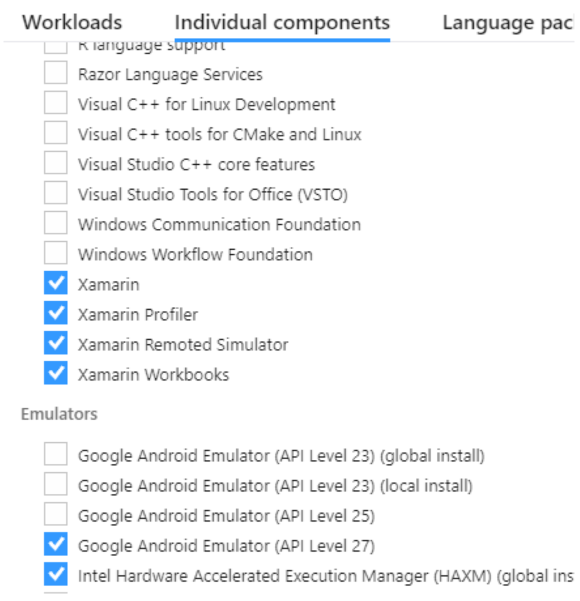
Mobile & Gaming (5)

✓ Mobile development with .NET
Build cross-platform applications for iOS, Android or Windows using Xamarin.

☐ Mobile development with JavaScript
Build Android, iOS and UWP apps using Tools for Apache Cordova.

☐ Game development with C++
Use the full power of C++ to build professional games powered by DirectX, Unreal, or Cocos2d.

Figure 3.1: Selecting Workloads



Workloads | Individual components | Language pac

- ☐ R language support
- ☐ Razor Language Services
- ☐ Visual C++ for Linux Development
- ☐ Visual C++ tools for CMake and Linux
- ☐ Visual Studio C++ core features
- ☐ Visual Studio Tools for Office (VSTO)
- ☐ Windows Communication Foundation
- ☐ Windows Workflow Foundation
- ✓ Xamarin
- ✓ Xamarin Profiler
- ✓ Xamarin Remoted Simulator
- ✓ Xamarin Workbooks

Emulators

- ☐ Google Android Emulator (API Level 23) (global install)
- ☐ Google Android Emulator (API Level 23) (local install)
- ☐ Google Android Emulator (API Level 25)
- ✓ Google Android Emulator (API Level 27)
- ✓ Intel Hardware Accelerated Execution Manager (HAXM) (global ins

Figure 3.2: Selecting Components

To start a project using Xamarin.Essentials, this package needs to be installed into the project. In the *Solution Explorer* panel, right click on the solution and select *Manage NuGet Packages*. Search for Xamarin.Essentials and install the package into all the projects. [10]

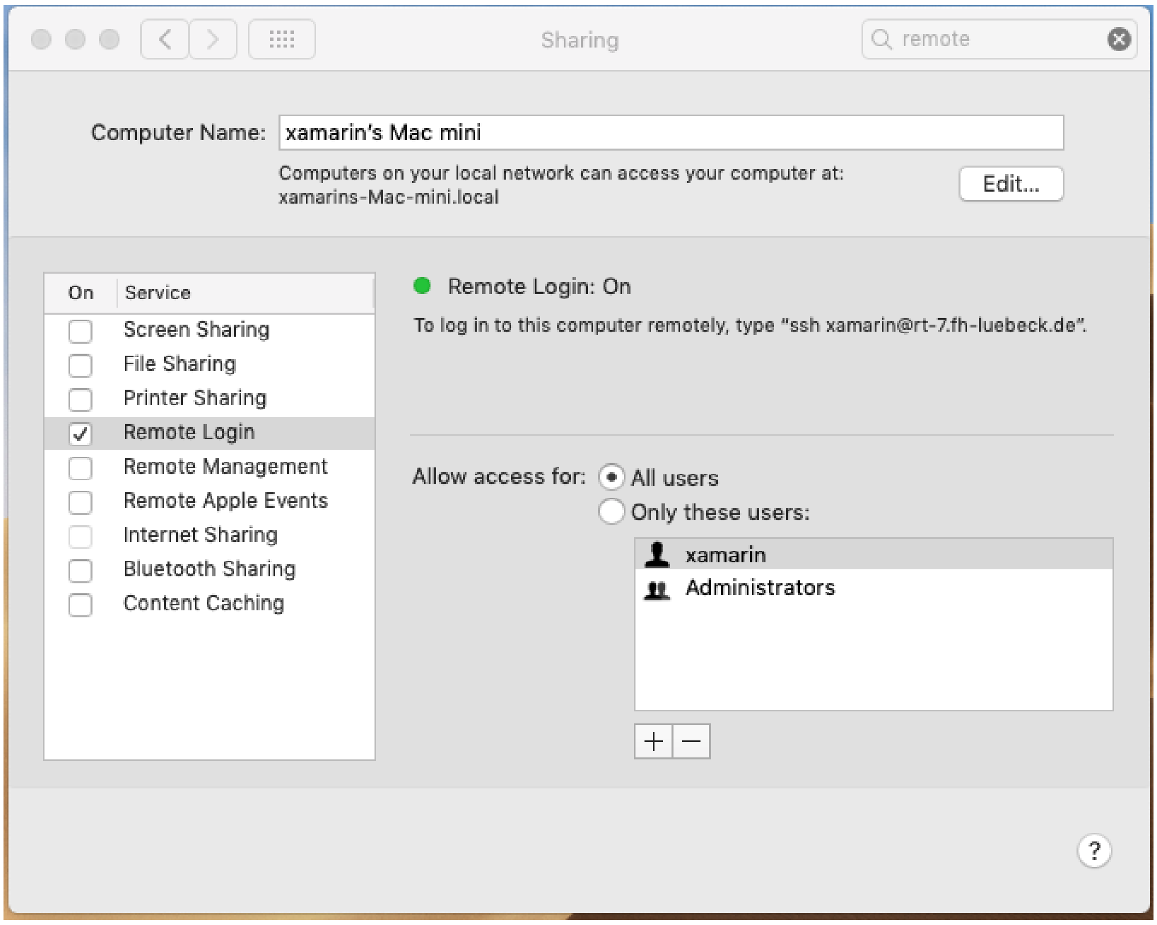
3.2 Xcode and Visual Studio for Mac

Xcode can be downloaded in App Store on Mac. Whereas Visual Studio for Mac is not available on App Store. Visual Studio for Mac should be downloaded in the download page of Microsoft Visual Studio: [8]. A warning might pop up when installing an application from the Internet on Mac. It is recommended to install all the platforms and tools when installing Visual Studio for Mac.

3.3 Connection to Mac and iOS devices

Mac and Windows computer should have network access to each other. [11] An easy way is to connect them in a LAN network by using a router. The iOS device should be directly connected to Mac.

In order to let Windows computer recognize Mac, remote login on Mac should be enabled. [12] Settings for remote login can be found in *Share* pane in *System Preferences* as illustrated in Figure 3.3.


Sharing

Computer Name: xamarin's Mac mini

Computers on your local network can access your computer at: xamarins-Mac-mini.local

On	Service
☐	Screen Sharing
☐	File Sharing
☐	Printer Sharing
✓	Remote Login
☐	Remote Management
☐	Remote Apple Events
☐	Internet Sharing
☐	Bluetooth Sharing
☐	Content Caching

● Remote Login: On

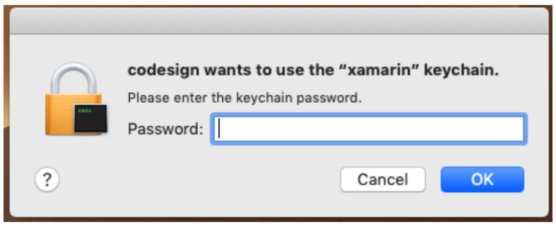
To log in to this computer remotely, type "ssh xamarin@rt-7.fh-luebeck.de".

Allow access for: [✓ All users] Only these users:

xamarin
Administrators

Figure 3.3: Enabling Remote Login on Mac

On Windows computer, after creating a Xamarin project, click *Pair to Mac* button. A window should show up, on which the Mac is shown. Select it and click *Connect*. During the first time of connecting, user name and password of Mac is needed.


codesign wants to use the "xamarin" keychain.

Please enter the keychain password.

Password:

Figure 3.4: Dialog Requiring Password

Sometimes when Visual Studio is trying to load the program onto iOS devices, Visual Studio is stuck for a long time. A possible reason could be that Visual Studio is compiling for the first time after its start. Another possibility is that a dialog which requires the user password of Mac pops up on Mac as shown in Figure 3.4. Only after inputting the password will Visual Studio continue loading the program onto iOS devices.

4 APIs of Xamarin.Essentials

This section provides introductions and implementation approaches of the APIs of Xamarin.Essentials. Meanwhile, comparisons of the APIs on each platform are given. Moreover, a function that is not included in Xamarin.Essentials is also presented at the end of this section.

4.1 Magnetometer

Xamarin.Essentials.Magnetometer gives access to the magnetometer sensor of the device. [13] This class consists of one property *IsMonitoring*, two methods *Start* and *Stop* for controlling the sensor, and one event *ReadingChanged* for handling the reading returned by the sensor.

The property *IsMonitoring* is a boolean value that indicates whether the sensor is monitoring. The method *Start* should be called with a parameter that indicates the sensor speed. There are four categories of sensor speed: *Default*, *UI*, *Game*, and *Fastest*. Generally, these four categories imply a sensor speed from slow to fast. The frequency of getting data of each category varies significantly depending on the platform and performance of the device itself. For example, the frequency of *Fastest* on iPhone 6 merely reaches the frequency of *Game* on Xiaomi 5. The reading returned by the sensor can be accessed in the event *ReadingChanged*.

4.1.1 Coordination System and Reading

The reading represents a three-dimensional vector which consists of three components of magnetic field intensity in three axes. The unit of the reading is microtesla. [13]

The coordination system of the device is defined as follows: [14]

- The positive X axis points to the right of the display in portrait mode.
- The positive Y axis points to the top of the device in portrait mode.
- The positive Z axis points out of the screen.

4.1.2 Implementing Magnetometer

Listing 4.1 shows a way to implement the magnetometer. Line 1 to line 5 shows the content of the *Clicked* event of a button. Line 3 indicates that every time this event is triggered, the method *ToggleMagnetometer* is called. In line 4, the text of the button changes between “Start” and “Stop”. Line 7 to line 18 shows the content of the method *ToggleMagnetometer*. Line 9, 10, 15, 16, 17 is a try-catch block which handles some exceptions. This is not closely related to implementing the magnetometer so it is omitted. Line 11 to line 14 guarantee that the sensor changes its state every time the method is called. This way, the control on the sensor binds to the button.

```
void OnButtonClicked(object sender, EventArgs e)
{
    ToggleMagnetometer();
    button.Text = String.Format("{0}", Magnetometer.IsMonitor-
ing == true ? "Stop" : "Start");
}

public void ToggleMagnetometer()
{
    try
    {
        if (Magnetometer.IsMonitoring)
            Magnetometer.Stop();
        else
            Magnetometer.Start(speed);
    }
    catch ...
    ...
}
```

Listing 4.1: Starting and Stopping Magnetometer

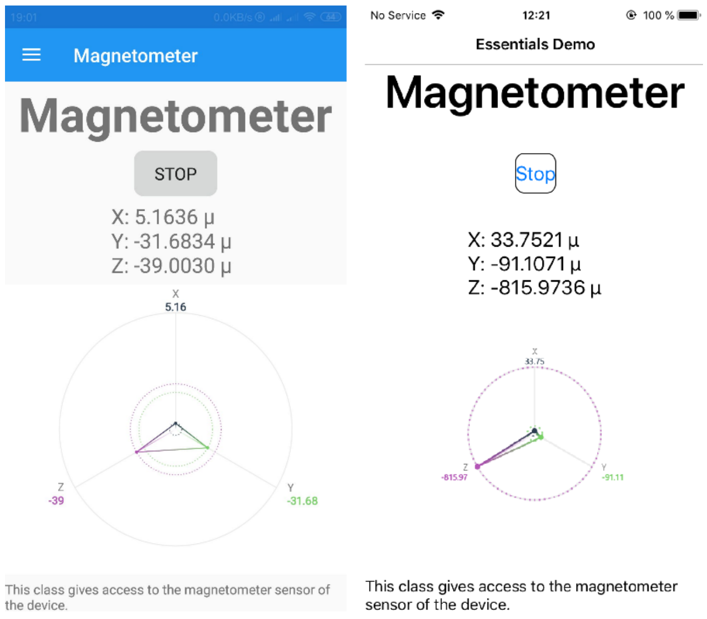
4.1.3 Handling the Reading

```
1.  public MagnetometerDemo()
2.  {
3.      ...
4.      Magnetometer.ReadingChanged += Magnetometer_ReadingChanged;
5.      ...
6.  }
7.  void Magnetometer_ReadingChanged(object sender, MagnetometerChangedEventArgs e)
8.  {
9.      var data = e.Reading;
10.     var data_X = data.MagneticField.X;
11.     var data_Y = data.MagneticField.Y;
12.     var data_Z = data.MagneticField.Z;
13.     ...
14.     var entries = new[]
15.     {
16.         new Entry((float)Math.Abs(Math.Round(data_X, 2)) * 10)
17.         {
18.             Label = "X",
19.             ValueLabel = Math.Round(data_X, 2).ToString(),
20.             Color = SKColor.Parse("#2c3e50")
21.         },
22.         ...
23.     };
24.     chartView.Chart = new RadarChart()
25.     {
26.         Entries = entries,
27.         ...
28.     };
29. }
```

Listing 4.2: Handling Data from Magnetometer

Listing 4.2 shows a way to retrieve and process the reading data from the magnetometer. Line 1 to line 6 indicates the constructor of the page. Unrelated codes are omitted. Line 4 shows that the method shown from line 7 to line 29 is set as the event handler of the magnetometer. Line 9 to line 12 shows how to retrieve data from the parameter. As mentioned in section 4.1.1, the reading consists of three components in three axes. Line 13 is some omitted unrelated code. Line 14 to line 23 inputs the data into a local variable *entries*. Line 22 represents the input processes of *data_Y* and *data_Z*. They are basically the same as the manipulation on *data_X* shown from line 16 to line 21, therefore, they are omitted. Line 24 to line 28 puts the data into a micro chart that can show the reading graphically. In line 27, the omitted code is setting some other properties of the micro chart such as text size.

4.1.4 Comparison on Different Platforms



X: 5.1636 µ
Y: -31.6834 µ
Z: -39.0030 µ

This class gives access to the magnetometer sensor of the device.

X: 33.7521 µ
Y: -91.1071 µ
Z: -815.9736 µ

This class gives access to the magnetometer sensor of the device.

Figure 4.1: Comparison of Magnetometer

Figure 4.1 shows the performance of magnetometer on Android and iOS devices. As can be seen, apart from the variation of the reading, magnetometer works similarly on Android and iOS platforms. The reading on iOS is extremely larger than the magnetic field intensity of the Earth. A possible cause could be the interference of the magnet nearby in the workshop when the screenshot was taken. The author was using the magnet to test the compass on different phones. Unfortunately, Raspberry Pi and Surface Pro 4 do not have magnetometer sensors so this class is not tested on the UWP platform.

4.2 MainThread

Xamarin.Essentials.MainThread enables a block of code to run on the main thread. Codes that need to access elements on user interface must run on the main thread, or sometimes called the UI thread. Meanwhile, there are some methods running on threads other than the main thread, such as methods that starts sensors with Game speed. When these methods are called, they need to get back to the main thread. [15]

Code that needs to get back to the main thread can simply be put inside the block shown in Listing 4.3. Line 1 calls the method *BeginInvokeOnMainThread*, which requires an *Action* as a parameter. [16] Line 1 uses a lambda expression to shorten the code. The code between line 2 and line 4 will run on the main thread.

```
1.  MainThread.BeginInvokeOnMainThread(() =>
2.  {
3.      // code that needs to get back to the main thread
4.  });
```

Listing 4.3: Code Block for Main Thread

When methods that start sensors are called, it is recommended to include this part of code inside the block above regardless of the sensor speed. When slower sensor speed is chosen and this part of code is already running on the main thread, use of MainThread seems redundant. On the contrary, the performance of the program hardly worsens. [15] Moreover, it prevents further bugs when sensor speed needs to be adjusted in the future.

4.3 Map

Xamarin.Essentials.Map gives access to the default map application of the device. [17] This class does not provide an internal map function. Instead, it enables the application to jump to the default map application with a specified placemark.

The method *OpenAsync* can be called with one or two parameters. The first parameter is either a *Location* instance or a *Placemark* instance. The second parameter is a *MapLaunchOptions* instance, which is an optional parameter. [18] Also, it is possible to replace the first parameter with two *Double* instances as the geographic coordinate. The compiler is still able to read it. A *Location* instance is constructed by geographic coordinates. A *Placemark* instance consists of much more detailed information than a *Location* instance. For example, the country name can be specified in a *PlaceMark*. The *MapLaunchOptions* instance is used to describe the placemark shown on the map and to choose the navigation mode, such as by car or by bicycle. [19]

4.3.1 Implementing Map

This class does not ask for location information and is only responsible for gathering destination information and jumping to the map application. The navigation proceeds in the map application. Therefore, no specific implementation regarding privacy or permission is needed.

```
1.  public MapDemo()
2.  {
3.      ...
4.      button.Clicked += OnButtonClickedAsync;
5.      ...
6.  }
7.  async void OnButtonClickedAsync(object sender, EventArgs e)
8.  {
9.      var location = new Location(47.645160, -122.1306032);
10.     var options = new MapLaunchOptions
11.     {
12.         NavigationMode = NavigationMode.Driving
13.     };
14.     await Map.OpenAsync(location, options);
15. }
```

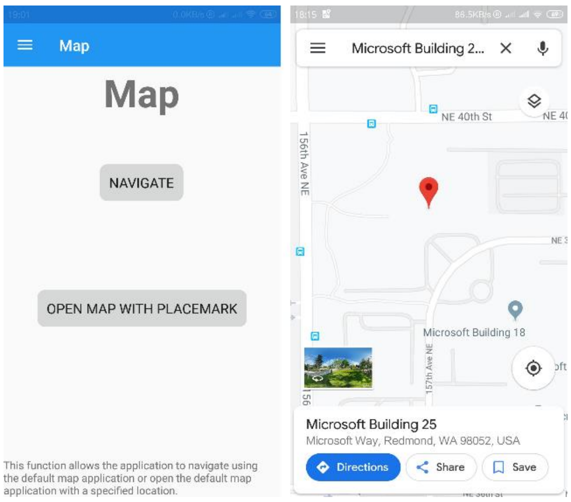
Listing 4.4: Implementing Map

Listing 4.4 shows a way to implement the *Map* class. Line 1 to line 6 is the constructor of the demonstration page. Line 4 sets the method *OnButtonClickedAsync* as the *Clicked* event of a button. Line 7 to line 15 is the content of the method. Line 9 is constructing a *Location* instance.

Line 10 to line 13 is constructing a *MapLaunchOptions* instance that indicates a navigation mode of driving. Line 14 calls the method *OpenAsync* with the two instances mentioned above. This method is an asynchronous method so that the “await” keyword is used here. As a result, after this block of code is executed, the application should jump to the default map application and navigates to a place which has a geographic coordinate of (47.64, -122.13) by car.

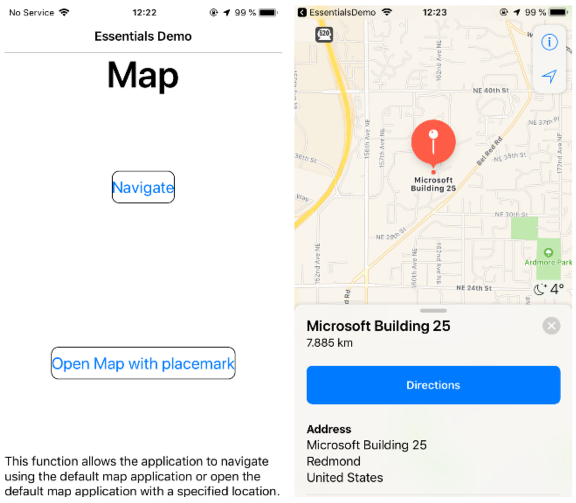
4.3.2 Comparison on Different Platforms

As can be shown in Figure 4.2, Figure 4.3, and Figure 4.4, Map works similarly on different platforms. They open the map with the same specified placemark in spite of the different display. A map application is not installed on the Raspberry Pi. Consequently, this class does not work on a Raspberry Pi.



This function allows the application to navigate using the default map application or open the default map application with a specified location.

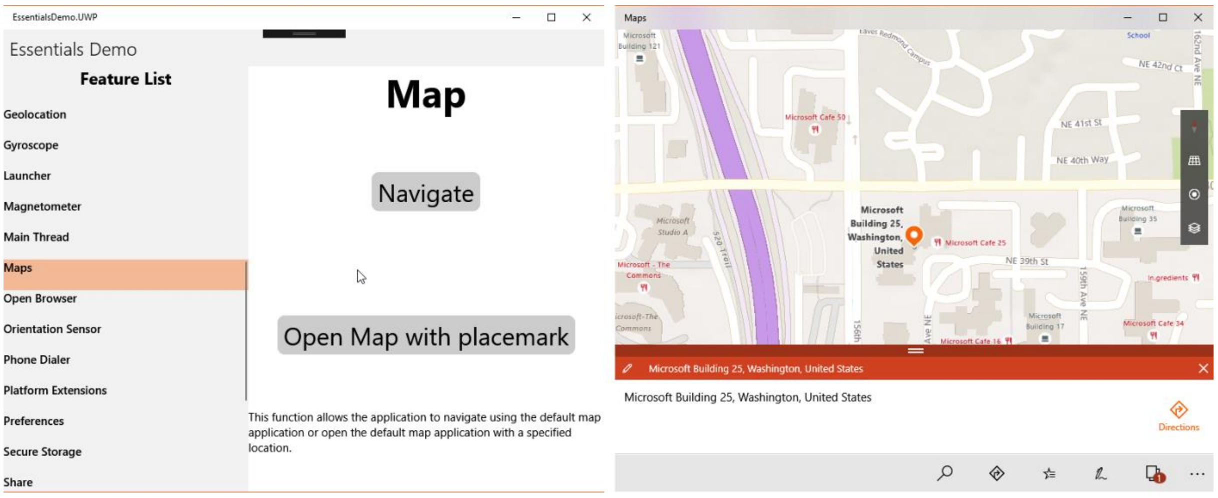
Figure 4.2: Map on Android



This function allows the application to navigate using the default map application or open the default map application with a specified location.

Address
Microsoft Building 25
Redmond
United States

Figure 4.3: Map on iOS



This function allows the application to navigate using the default map application or open the default map application with a specified location.

Microsoft Building 25, Washington, United States

Figure 4.4: Map on UWP

4.4 Browser

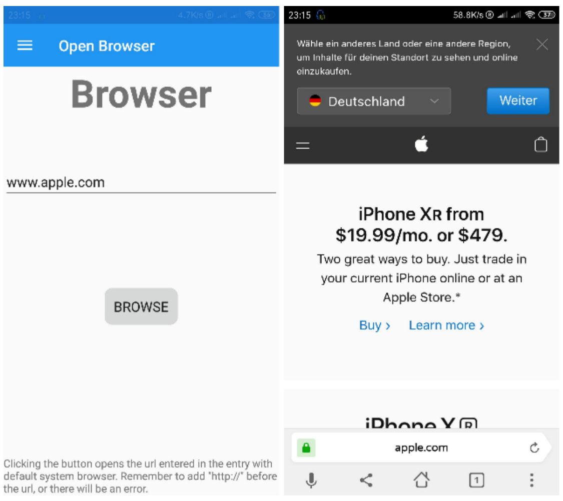
Xamarin.Essentials.Browser gives access to the default browser of the device. [20] This class does not provide an internal browser function. Instead, it jumps to the default browser application or to a new page inside the application, depending on the parameter.

This class contains a method *OpenAsync*. The first parameter of this method can either be a *Uri* instance or a string. A string is much easier and more direct, therefore, a string is recommended here. The first parameter should be a complete URI when calling the method *OpenAsync*. For example, "https://www.google.com/" should be used instead of "www.google.com". A formatting method that helps complete URI using regular expression is included in the appendix program. Further help methods that help writing legal URI can be written by using regular expressions.

The second parameter is an optional parameter. It can either be a *BrowserLaunchMode* instance of a *BrowserLaunchOptions* instance. Launch mode has two categories: *SystemPreferred* and *External*. *SystemPreferred* indicates that the webpage will be opened on the other page inside the application. *External* literally means the webpage will be opened externally. In *BrowserLaunchOptions*, more customization options can be specified, such as the toolbar color. [21]

4.4.1 Comparison on Different Platforms

Figure 4.4 shows that the application opens Apple's official website with its default browser inside the application. Figure 4.6 shows that the application opens google with Safari inside the application. This results from the launch mode of *SystemPreferred* set in the program. However, as Figure 4.7 shows, the application on UWP opens the webpage with Google Chrome, which is an external browser, regardless of the launch mode set in the program. This means that it is of no use to set *BrowserLaunchMode* for the UWP platform. In spite of this flaw, this class is still able to achieve its main purpose on the UWP platform.



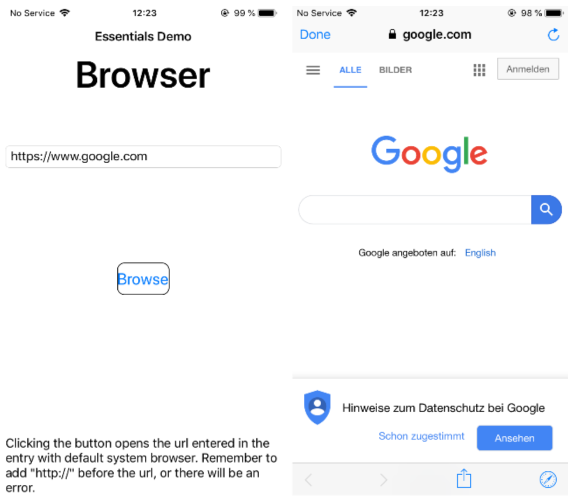
Clicking the button opens the url entered in the entry with default system browser. Remember to add "http://" before the url, or there will be an error.

Wähle ein anderes Land oder eine andere Region, um Inhalte für deinen Standort zu sehen und online einzukaufen.

iPhone XR from
$19.99/mo. or $479.
Two great ways to buy. Just trade in your current iPhone online or at an Apple Store.*
Buy › Learn more ›

iPhone XR

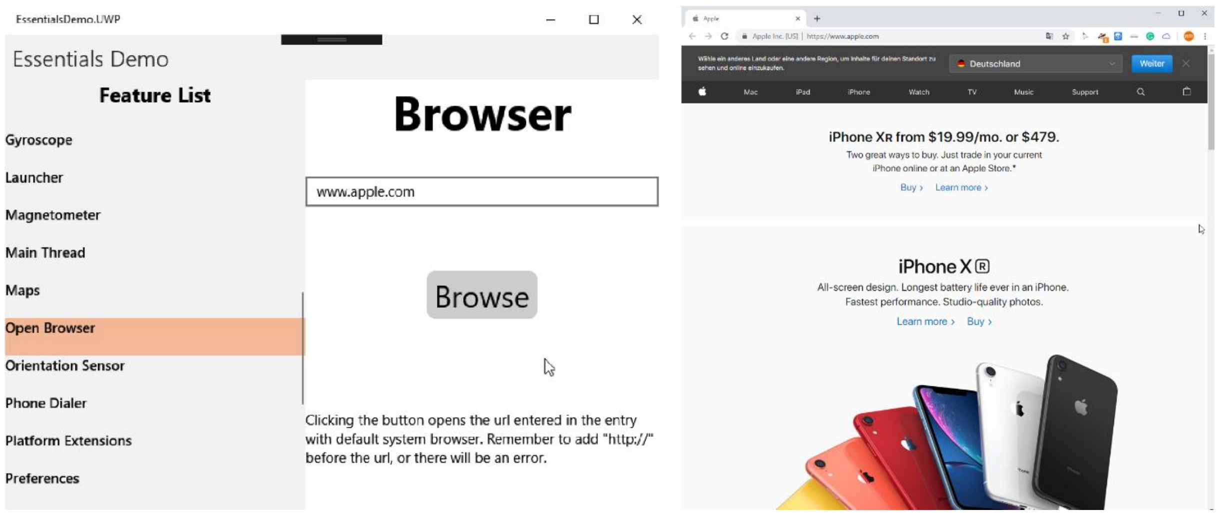
Figure 4.5: Browser on Android



Clicking the button opens the url entered in the entry with default system browser. Remember to add "http://" before the url, or there will be an error.

Hinweise zum Datenschutz bei Google

Figure 4.6: Browser on iOS



Clicking the button opens the url entered in the entry with default system browser. Remember to add "http://" before the url, or there will be an error.

iPhone XR from $19.99/mo. or $479.
Two great ways to buy. Just trade in your current iPhone online or at an Apple Store.*

iPhone XR
All-screen design. Longest battery life ever in an iPhone. Fastest performance. Studio-quality photos.

Figure 4.7: Browser on UWP

4.5 Orientation Sensor

Xamarin.Essentials.OrientationSensor enables the application to monitor the orientation of the device. [14]

4.5.1 Implementation of Orientation Sensor

Control of orientation sensor is basically the same as that of magnetometer which is described in section 4.1.2. As for demonstrating the reading, a picture is introduced in the demonstration page. The demonstration page aims to manipulate the attributes of the image so that the arrow in the picture acts as if it were pointing to a fixed direction when the device rotates. The attributes include *RotationX*, *RotationY*, and *Rotation*.

Listing 4.5 shows a way to handle the data and present it through manipulating the image. Line 1 indicates that this is the *ReadingChanged* event of the orientation sensor. Line 3, 4, 21 is a block in order to get back to the main thread. Details about the main thread can be found in section 4.2. Line 5, 7, 8, 9, and 10 is retrieving the data. Line 12, 13, and 14 is processing the data. Line 19, 20, 21 is setting the attribute the processed data. Details about how to process this data can be found in section 4.5.3. The variables *zero* and *delta* appeared in line 17 and line 19 are explained in section 4.5.5.

```
void OrientationSensor_ReadingChanged(object sender, OrientationSensorChangedEventArgs e)
{
    MainThread.BeginInvokeOnMainThread(() =>
    {
        var data = e.Reading;

        var x = data.Orientation.X;
        var y = data.Orientation.Y;
        var z = data.Orientation.Z;
        var w = data.Orientation.W;

        var α = Math.Atan2(2 * (x * y + z * w), 1 - 2 * (y * y + z * z)) * 180 / Math.PI;
        var β = Math.Asin(2 * (x * z - y * w)) * 180 / Math.PI;
        var γ = Math.Atan2(2 * (x * w + y * z), 1 - 2 * (z * z + w * w)) * 180 / Math.PI;

        zero = α;

        image.Rotation = α - delta;
        image.RotationY = β;
        image.RotationX = -γ - 180;
    });
}
```

Listing 4.5: Implementing Orientation Sensor

4.5.2 Coordination System and Reading

The reading has four values. Altogether the four values represent the orientation relative to the zero position of the device by comparing the coordination system of both. The coordination system of the device is defined as follows: [14]

- The positive X axis points to the right of the display in portrait mode.
- The positive Y axis points to the top of the device in portrait mode.
- The positive Z axis points out of the screen.

The coordination system of zero position is set every time the sensor is started and is defined as follows:

- The positive X axis is tangent to the surface of the Earth and points to the right of the device in portrait mode.
- The positive Y axis is tangent to the surface of the Earth and points to the top of the device in portrait mode.
- The positive Z axis is perpendicular to the surface of the Earth and points up.

On the Android and UWP platform, the coordination system with which the device is comparing its own one is different. This will be discussed in section 4.5.5.

Assume that the rotation axis is described by a normalized vector (a_x, a_y, a_z) and the rotation angle is θ , the four values of the reading have the following meaning: $\left(a_x \sin\left(\frac{\theta}{2}\right), a_y \sin\left(\frac{\theta}{2}\right), a_z \sin\left(\frac{\theta}{2}\right), \cos\left(\frac{\theta}{2}\right)\right)$. All angles here are in radian.

4.5.3 Rotation of Image

The reading has the same form as quaternions. The three attributes of the image have the same form as Euler angles. Mathematically, rotating a coordinate system can be described equally by quaternions or by Euler angles. Therefore, it is possible to convert quaternion into Euler angles.

Assume Euler angle is written in (α, β, γ), the quaternion is written in (x, y, z, w), formulas for conversion between quaternions and Euler angles are shown in Listing 4.6. [22]

$$\begin{pmatrix} \alpha \\ \beta \\ \gamma \end{pmatrix} = \begin{pmatrix} \arctan\left(\frac{2(xy + zw)}{1 - 2(y^2 + z^2)}\right) \\ \arcsin\left(2(xz - yw)\right) \\ \arctan\left(\frac{2(xw + yz)}{1 - 2(z^2 + w^2)}\right) \end{pmatrix}$$

Listing 4.6: Formulas for Conversion between Quaternions and Euler Angles

In C#, a method *Atan2* is introduced to handle zero denominators in inverse tangent functions. Generally speaking, *Atan2* method is more stable than *Atan* method. Take into consideration the variation between the coordinate system of images in C# and the coordinate system of the device defined in section 4.5.2, the conversion from Euler angles to the attributes of the image can be described in the equation shown in Listing 4.7.

$$\begin{pmatrix} Rotation \\ RotationY \\ RotationX \end{pmatrix} = \begin{pmatrix} \alpha \\ \beta \\ -\gamma - \pi \end{pmatrix}$$

Listing 4.7: Relation between Euler Angles and the attributes

Note that all angles here are in radian and should be converted into degrees when coding.

4.5.4 Filter

No filter is needed here. Unlike *Xamarin.Essentials.Accelerometer*, which is discussed in the thesis written by Qin Lu, the data returned by this class is extremely stable. A possible reason is that these data are calculated by readings from accelerometer and gyroscope and an effective filter to deal with the jitter is already used. Another hypothesis is that these data are from another sensor with excellent performance and hardly undergo any jitter.

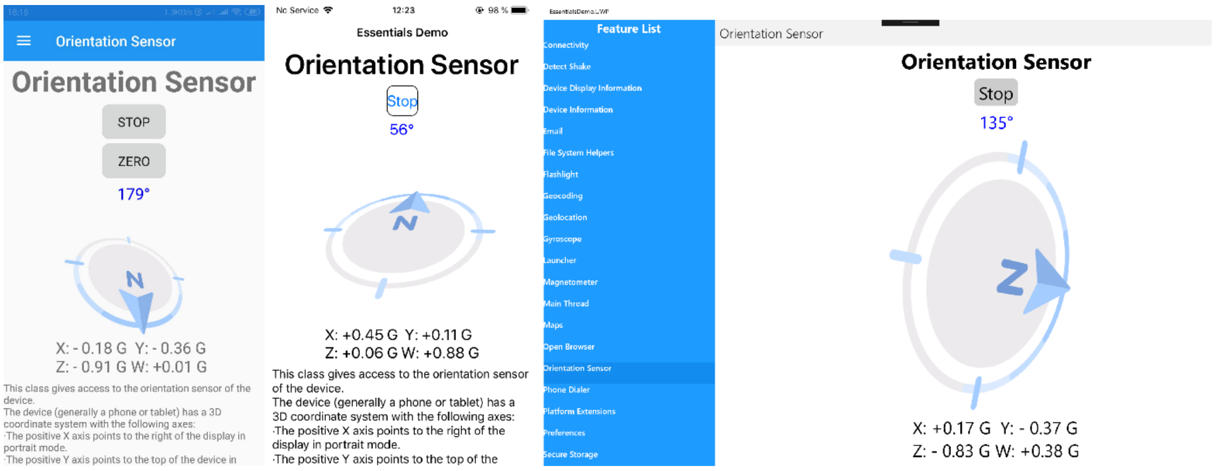
4.5.5 Comparison on Different Platforms

As mentioned in section 4.5.2, the coordination system with which the Android or UWP device is comparing its own one is different. For an Android or UWP device, the reading represents the orientation relative to the Earth. The coordination of the Earth is defined as follows: [14]

- The positive X axis is tangent to the surface of the Earth and points east.
- The positive Y axis is tangent to the surface of the Earth and points north.
- The positive Z axis is perpendicular to the surface of the Earth and points up.

In order to make rotation of the image in the demonstration page look the same on different platforms, certain changes are implemented. A zero button is provided in the demonstration page on these two platforms. After clicking the zero button, the application records the current position and set it to zero position. As a result, the application on Android or UWP works the same as on iOS after clicking the zero button.

This is achieved by introducing two variables: *zero* and *delta*, which appeared in Listing 4.5. As shown in line 16 of Listing 4.5, the variable *zero* is always recording the current position of the device. The variable *delta* serves as an intermediate variable and is originally set to zero. The moment the zero button is clicked is set as the manual zero position. After the zero button is clicked, the value of *zero* is passed to *delta*. Consequentially, the difference between the coordination system of the Earth and that of the manual zero position is represented by *delta*. By subtracting one of the Euler angles by *delta*, the difference between the two coordination system is eliminated.



X: - 0.18 G Y: - 0.36 G
Z: - 0.91 G W: +0.01 G

This class gives access to the orientation sensor of the device.
The device (generally a phone or tablet) has a 3D coordinate system with the following axes:
·The positive X axis points to the right of the display in portrait mode.
·The positive Y axis points to the top of the device in

X: +0.45 G Y: +0.11 G
Z: +0.06 G W: +0.88 G

This class gives access to the orientation sensor of the device.
The device (generally a phone or tablet) has a 3D coordinate system with the following axes:
·The positive X axis points to the right of the display in portrait mode.
·The positive Y axis points to the top of the

X: +0.17 G Y: - 0.37 G
Z: - 0.83 G W: +0.38 G

Figure 4.8: Comparison of Orientation Sensor

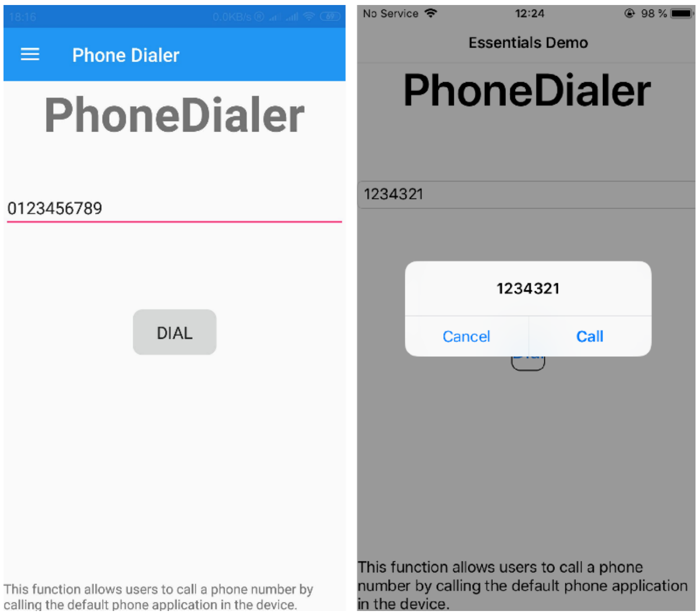
Figure 4.8 shows the performance of this class on each platform. Unfortunately, it is no possible to display movement through screenshots. It performs much better on a moving portable device. There does not exist needed sensors on Raspberry Pi so it is not tested on the Raspberry Pi.

4.6 Phone Dialer

Xamarin.Essentials.PhoneDialer gives access to the phone application of the device. [23]

After calling the method *Open*, the application opens the dialer with the number specified in the parameter. The method *Open* requires a string as its parameter. Therefore, the number should be written in strings instead of integers.

4.6.1 Comparison on Different Platforms



This function allows users to call a phone number by calling the default phone application in the device.

This function allows users to call a phone number by calling the default phone application in the device.

Figure 4.9: Comparison of Phone Dialer

As shown in Figure 4.9, this class works similarly on iOS and on Android. After clicking the dial button, the device calls the specified number. Unfortunately, Surface Pro 4, Windows computer, Raspberry Pi do not have a dialer. So this class is not tested on the UWP platform.

4.7 Platform Extensions

Xamarin.Essentials provides some help methods that can convert some types from system version to platform version. Note that these help methods can only be called in iOS, Android, or UWP project. [24] *Xamarin.Essentials* currently provide help methods for four types: color, point, rectangle, and size. [25]

4.7.1 Implementation of Platform Extensions

As mentioned above, these help methods can only be called in platform projects. As a result, an interface in the main project is needed. The interface should be implemented using platform extensions in each platform project.

Listing 4.8 is the whole content of the interface. Line 1 introduces the package *System.Drawing* to ensure the *Point*, *Size*, and *Rectangle* class below come from a system package. Line 3 indicates the namespace. Line 5 indicates the name of the interface. Line 7 to line 9 specifies the method names, return types, parameters without the bodies of the methods.

```
using System.Drawing;

namespace EssentialsDemo
{
    public interface IPlatformExtensions
    {
        string GetPlatformPoint(Point point);
        string GetPlatformSize(Size size);
        string GetPlatformRect(Rectangle rect);
    }
}
```

Listing 4.8: IPlatformaExtensions.cs

```
using EssentialsDemo.iOS;
using Xamarin.Essentials;
using Xamarin.Forms;

[assembly: Dependency(typeof(PlatformExtensions))]

namespace EssentialsDemo.iOS
{
    public class PlatformExtensions : IPlatformExtensions
    {
        public string GetPlatformPoint(System.Drawing.Point point)
        {
            var platform = point.ToPlatformPoint();
            return platform.ToString();
        }

        public string GetPlatformSize(System.Drawing.Size size)
        {
            var platform = size.ToPlatformSize();
            return platform.ToString();
        }

        public string GetPlatformRect(System.Drawing.Rectangle rect)
        {
            var platform = rect.ToPlatformRectangle();
            return platform.ToString();
        }
    }
}
```

Listing 4.9: PlatformExtensions.cs in iOS Project

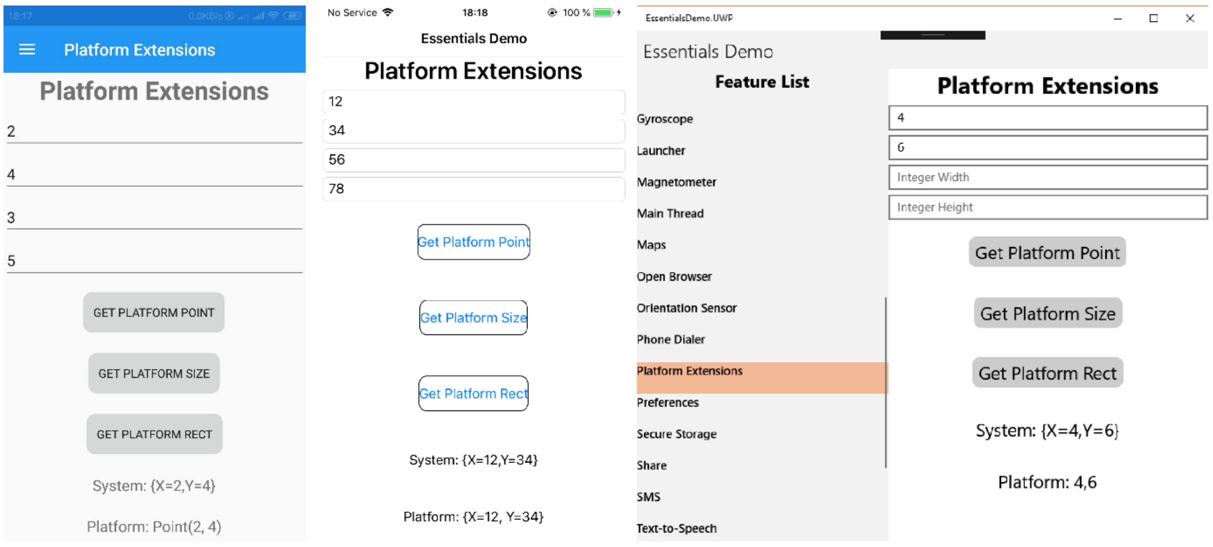
Listing 4.9 shows the whole content of a class that implements the interface in Listing 4.8. Only the class in the iOS project is shown here. However, the class in each platform project is almost the same except the suffix of the namespace. Line 1 to line 3 is introducing packages. Note that *Xamarin.Essentials* is introduced here so that the help functions can be accessed. Line 5 indicates that the class *PlatformExtensions* here provides a concrete implementation of some interface. Line 7 indicates the namespace. Line 9 indicates that this class *PlatformExtensions* is implementing the interface *IPlatformExtensions*. Line 11 to line 15 implements the method shown in line 7 of Listing 4.8. Due to the introduction of *Xamarin.Forms*, the class *Point* needs to be specified that it comes from the *System.Drawing* package instead of *Xamarin.Forms*. Line 13 calls the platform extension method *ToPlatformPoint*. As a result, the parameter *point* is now in the form of *CoreGraphics.CGPoint*, which is a class that can be recognized by iOS devices. Line 14 returns the parameter *point* in its platform form. Line 17 to line 27 is basically doing the same thing.

Listing 4.10 is a *Clicked* event of a button. Line 3, 4, and line 8 to line 12 shows a try-catch block. Line 11 indicates that when an exception is caught, an alert is displayed. Line 5 gathers the text in the entry into a local *Point* variable. Line 6 passes the system form of the point to the first label. Line 7 passes the platform form of the point to the second label. Line 7 uses the class *DependencyService* to retrieve the platform-specific implementation of *IPlatformExtensions*. The implementation approaches of the other classes supported by platform extensions are basically the same.

```
1.  void OnButtonClicked(object sender, EventArgs e)
2.  {
3.      try
4.      {
5.          var system = new System.Drawing.Point(Int32.Parse(entry.Text), Int32.Parse(entry2.Text));
6.          label.Text = "System: " + system.ToString();
7.          label2.Text = "Platform: " + DependencyService.Get<IPlatformExtensions>().GetPlatformPoint(system);
8.      }
9.      catch (Exception)
10.     {
11.         DisplayAlert("Error", "An error has occured.", "OK");
12.     }
13. }
```

Listing 4.10: Referring to the Implemented Method

4.7.2 Comparison on Different Platforms



System: {X=2,Y=4}

Platform: Point(2, 4)

System: {X=12,Y=34}

Platform: {X=12, Y=34}

System: {X=4,Y=6}

Platform: 4,6

Figure 4.10: Comparison of Platform Extensions

As Figure 4.10 shows, platform extensions works differently on each platform. A first sight of this difference can be gained from the different forms of platform display of *Point*. Internally, the *Point* classes on these platforms are completely different. This means the three platforms are interpreting the same code differently.

4.8 Preferences

Xamarin.Essentials.Preferences enables the application to store preferences in key-value pairs. [26] In other words, users' choices can be reproduced by the application when the application is opened the next time.

Most common types, such as string, int, *DateTime* can be stored with the method *Set*. A unique key is required for each value. The method *Get* requires a string as the key and a default value if the key does not exist. A third parameter indicating *sharedName* is optional. Keys and values can be cleared using the method *Clear*. When calling the method *Clear*, *sharedName* can be specified. All key-value pairs with the same *sharedName* can be cleared at once. [27]

4.8.1 Implementing Preferences

In the appendix program, content input by users will be stored and when this application is re-opened, this page will be the same as the application closed.

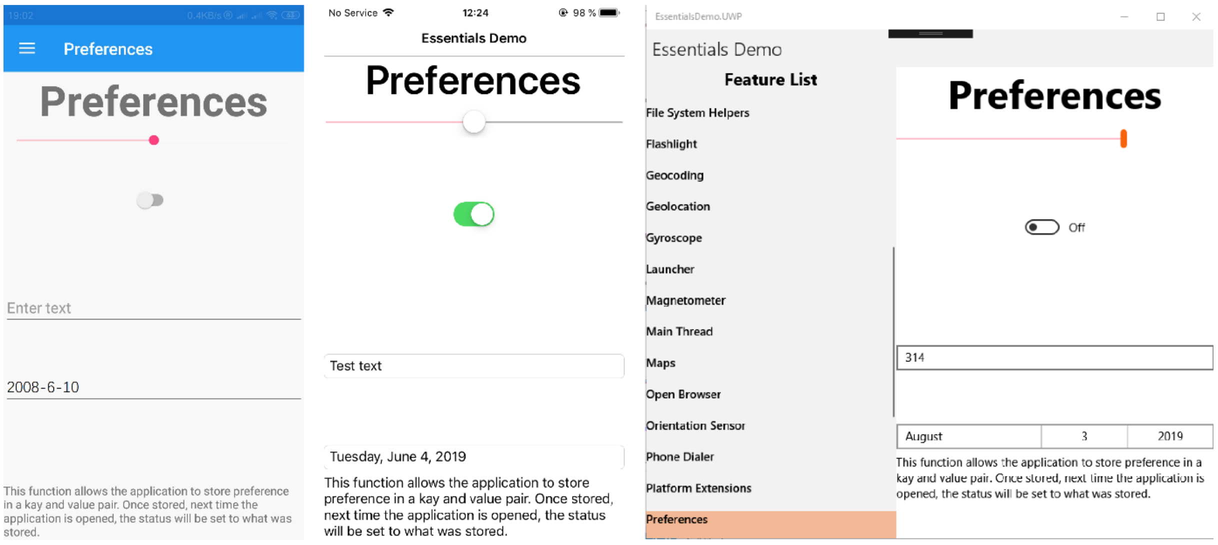
In order to achieve this, the method *Get* is used when initializing the page. A default value can be specified in the second parameter. *Set* methods are called in the input elements' content changed events. Listing 4.11 shows a way to realize this.

Line 1 to line 10 shows that the initialization of the switcher is written in the constructor of the page. Line 6 sets the *IsToggled* property with the return value of the method *Get*. Initially, the value for the key "IsToggled" is not set. Consequentially, this method returns false as specified in the parameter. Line 8 sets the method *OnSwitcherToggled* as the *Toggled* event of the switcher. Line 11 to line 14 describes the method *OnSwitcherToggled*. Generally speaking, whenever the switcher is toggled, its current state is saved with the key “IsToggled”. Consequentially, when the next time the application is opened and the switcher is initialized, line 6 will get the last saved state.

```
1.  public PreferencesDemo()
2.  {
3.      ...
4.      switcher = new Switch
5.      {
6.          IsToggled = Preferences.Get("IsToggled", false)
7.      };
8.      switcher.Toggled += OnSwitcherToggled;
9.      ...
10. }
11. void OnSwitcherToggled(object sender, ToggledEventArgs e)
12. {
13.     Preferences.Set("IsToggled", e.Value);
14. }
```

Listing 4.11: Implementing Preferences

4.8.2 Comparison on Different Platforms



This function allows the application to store preference in a kay and value pair. Once stored, next time the application is opened, the status will be set to what was stored.

This function allows the application to store preference in a kay and value pair. Once stored, next time the application is opened, the status will be set to what was stored.

This function allows the application to store preference in a kay and value pair. Once stored, next time the application is opened, the status will be set to what was stored.

Figure 4.11: Comparison of Preferences

As Figure 4.11 shows, despite that the display on each platform has small variations, this class works equivalently on each platform.

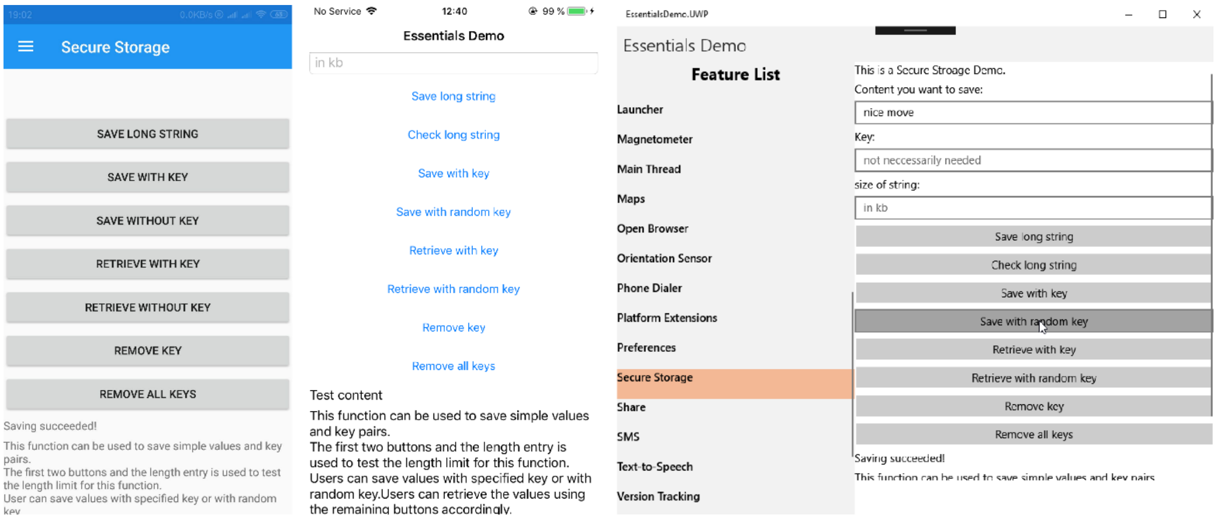
4.9 Secure Storage

Xamarin.Essentials.SecureStorage enables the application to store strings in key-value pairs. [28] From a perspective of coding, secure storage works similarly with *Preferences*.

Unlike the class *Preferences,* the method *GetAsync* does not have a second parameter for a default value and will return a null value when the key does not exist. [28] The method *SetAsync* works similarly to that of *Preference*, except that this method is an asynchronous method and *sharedName* is not available here.

4.9.1 Comparison on Different Platforms

On the secure storage demonstration page, the maximum length of storage can be tested. The maximum length varies from different devices. The maximum length on different devices rages from 10 kilobytes to a few megabytes. Note that the performance is not desirable when working with large strings so that this class is not suitable for storing large strings.



Saving succeeded!

This function can be used to save simple values and key pairs.
The first two buttons and the length entry is used to test the length limit for this function.
User can save values with specified key or with random key

Test content

This function can be used to save simple values and key pairs.
The first two buttons and the length entry is used to test the length limit for this function.
Users can save values with specified key or with random key.Users can retrieve the values using the remaining buttons accordingly.

This is a Secure Stroage Demo.

Saving succeeded!

This function can be used to save simple values and key pairs

Figure 4.12: Comparison of Secure Storage

As Figure 4.12 shows, despite the maximum length of the content, this class works similarly on each platform. This class is also available on Raspberry Pi.

4.10 Share

Xamarin.Essentials.Share enables the application to share data to other applications on the device. [29]

With the method *RequestAsync*, a text or a URI can be shared. The content is specified in the parameter by creating a new instance of *ShareTextRequest* and setting its properties. A *ShareTextRequest* has four properties: *Subject*, *Text*, *Title*, *Uri*. [30]

4.10.1 Comparison on Different Platforms

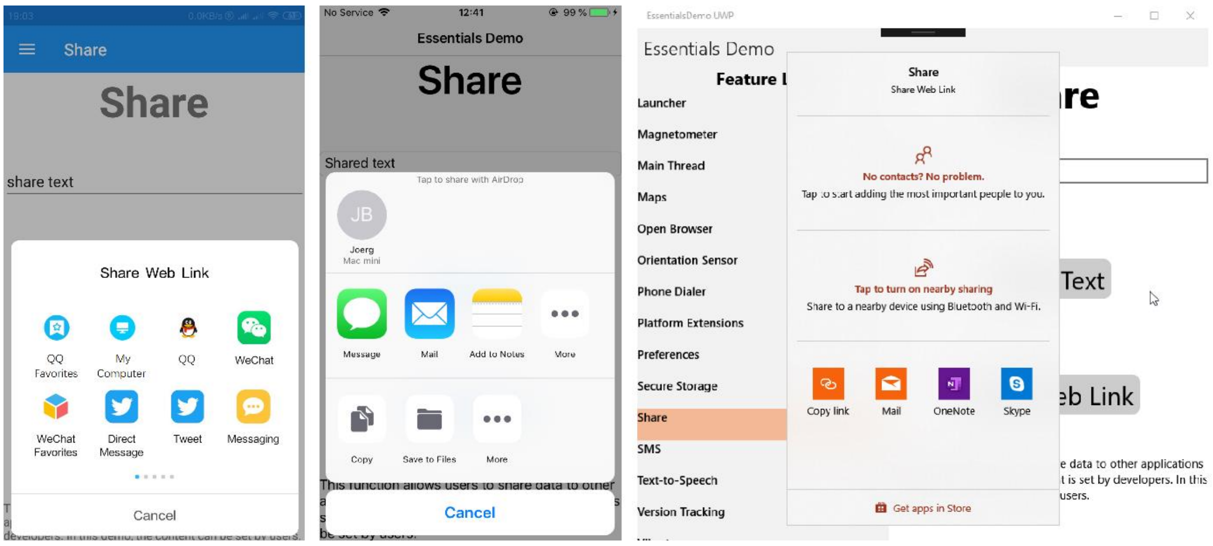


Figure 4.13: Comparison of Share

As shown in Figure 4.13, the string "Share Web Link" appears on Android and UWP, whereas not on iOS. "Share Web Link" is the *Title* property in the demonstration page. This means *Title* property is not used on the iOS platform. Meanwhile, *Subject* property is not used on iOS and on UWP. Unfortunately, this class does not work on Windows IoT.

4.11 SMS

Xamarin.Essentials.SMS gives access to the default SMS application of the device. [31] After calling the method *ComposeAsync* the application jumps to the default SMS application with the specified information in the parameter.

The method *ComposeAsync* needs a *SmsMessage* instance as its parameter. The payload of the message and recipients can be specified when creating the *SmsMessage* instance. Recipients can either be a string or an array of string representing a list of recipients. [32]

4.11.1 Comparison on Different Platforms

Figure 4.14 shows the SMS demonstration page on Android. After clicking the send button, the method *OpenAsync* is called and the application jumps to the SMS application with the message content and recipient number entered in the entries. Figure 4.15 shows the SMS demonstration page on iOS. The process is the same on iOS.

On Surface Pro 4 and Windows Computers, after clicking the send button, the application opens an application called Messaging. However, this application only shows SMS texts from the mobile phone if the mobile phone is bound to the computer. The application can't send SMS texts.

On the other hand, the Raspberry Pi does not respond to the method. Therefore, this class is not tested on the UWP platform. Given the simplicity of the function of this class, it is highly possible that this class would work on UWP platform, as well.

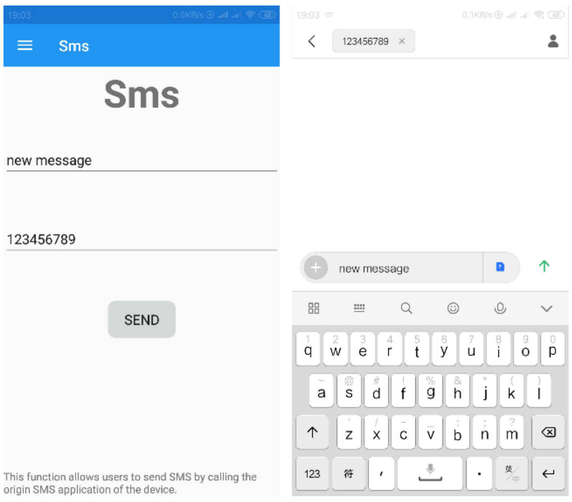

Sms
Sms
new message
123456789
SEND
This function allows users to send SMS by calling the origin SMS application of the device.
123456789
new message

Figure 4.14: SMS on Android

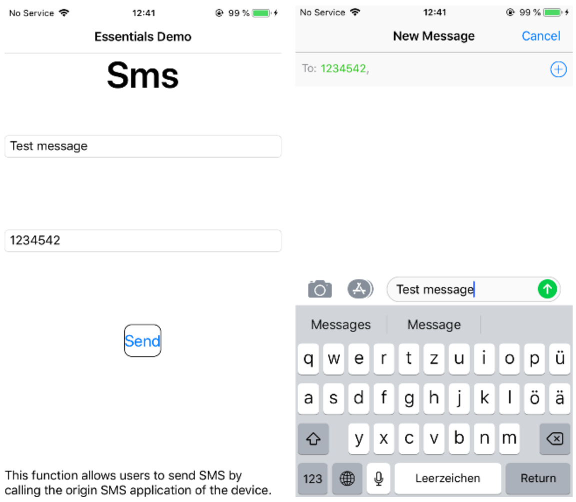


Figure 4.15: SMS on iOS

4.12 Text-to-Speech

Xamarin.Essentials.TextToSpeech enables the application to read text. [33]

The method *SpeakAsync* can be called only with a string, or with a string and a *SpeechOptions* instance, or with a string and a *CancellationToken* instance, or with all of them. Considering multithreading is not involved here, a cancellation token is not needed. The properties *Volume*, *Pitch*, and *Locale* can be set in *SpeechOptions*. *Volume* and *Pitch* are both float values. *Volume* ranges from 0 to 1 and *Pitch* ranges from 0 to 2. With adjustment of *Pitch*, the pitch of the voice can be lower or higher, though the difference is not obvious.

4.12.1 Implementing Text to Speech

```
1.  slider_volume = new Slider { Maximum = 1.0 };
2.  slider_pitch = new Slider { Maximum = 2.0 };
3.  slider_pitch.Value = 1.0;
4.  slider_volume.Value = 0.75;
```

Listing 4.12: Sliders for Volume and Pitch

As Listing 4.12 shows, the adjustment of *Volume* and *Pitch* is made possible by introducing two sliders. Line 1 and line 2 is constructing the sliders as well as setting the maximum values of them. Line 3 and line 4 are setting the default value of these two properties.

```
1.  public TextToSpeechDemo()
2.  {
3.      private Locale locale;
4.      private IEnumerable<Locale> locales;
5.      ...
6.      GetLocalesAsync();
7.      foreach (Locale localeValue in locales)
8.      {
9.          picker_locale.Items.Add(localeValue.Language);
10.     }
11.     picker_locale.SelectedIndexChanged += Picker_locale_SelectedIn-
    dexChanged;
12.     ...
13. }
14. private async void GetLocalesAsync()
15. {
16.     locales = await TextToSpeech.GetLocalesAsync();
17. }
18. private void Picker_locale_SelectedIndexChanged(object sender, Even-
    tArgs e)
19. {
20.     locale = locales.ElementAt(picker_locale.SelectedIndex);
21. }
```

Listing 4.13: Pickers and Methods for Locale

Listing 4.13 shows a way to implement a picker for locales. Line 1 to line 13 is the constructor of the demonstration page. Line 6 gets all the locales the device supports. It could have been more simple if line 16 can be put in line 6. However, *TextToSpeech.GetLocalesAsync* is an asynchronous method and an asynchronous method cannot be called in a constructor.

A solution is to introduce two intermediate variables to mend this contradiction as line 3 and line 4 shows. Two private variables are introduced. Line 6 calls a method shown from line 14 to line 17 and this method calls the asynchronous method *TextToSpeech.GetLocalesAsync*. Ultimately, the return value of *TextToSpeech.GetLocalesAsync* is set to the private variable. Line 7 to line 10 uses *foreach* to put all the locales into the picker.

Line 11 sets the *SelectedIndexChanged* event of the picker, which is shown from line 18 to line 21. Line 20 sets the variable *locale* to the selected locale in the picker.

```
1.  private void Button_speak_Clicked(object sender, EventArgs e)
2.  {
3.      if (!string.IsNullOrWhiteSpace(entry.Text))
4.      {
5.          SpeakNow((float)slider_volume.Value, (float)slider_pitch.Value, locale, entry.Text);
6.      }
7.      else
8.      {
9.          SpeakNow((float)slider_volume.Value, (float)slider_pitch.Value, locale, "Please enter something.");
10.     }
11. }
12.
13. public async Task SpeakNow(float volume, float pitch, Locale locale, string text)
14. {
15.     var settings = new SpeechOptions()
16.     {
17.         Volume = volume,
18.         Pitch = pitch,
19.         Locale = locale
20.     };
21.     await TextToSpeech.SpeakAsync(text, settings);
22. }
```

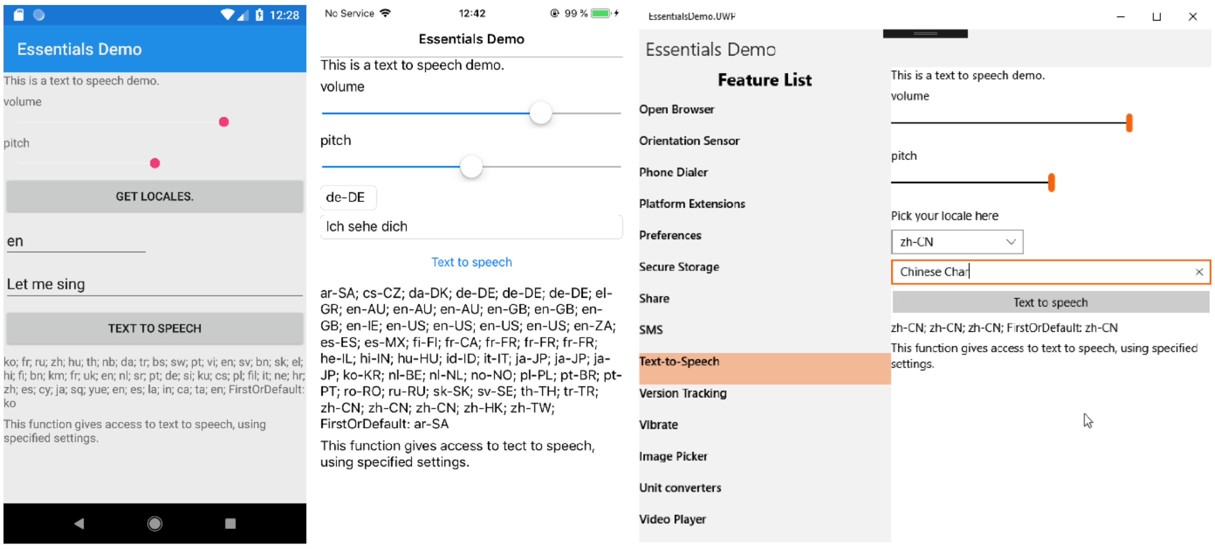
Listing 4.14: Implementing Text to Speech

Listing 4.14 shows the way to finally make the application "speak". Line 1 to line 11 is a *Clicked* event of a button. Line 3 judges whether the entry is empty. If it is not empty, line 5 passes the value of two sliders, the value of the picker, and the content in the entry to the method *SpeakNow*. If it is empty, it instead passes "Please enter something" to the method. Line 13 to line 22 calls the method *SpeakAsync* in a manner that section 4.12 earlier described. Line 17, 18, 19 sets the properties with the parameters. Line 21 calls the method *SpeakAsync*.

4.12.2 Comparison on Different Platforms

On mobile devices, locale supports most common languages, such as English, French. It is not recommended to use the method *FirstOrDefault* to get the default locale. The reason is that this usually does not work and ends up selecting Arabic or French instead of the current language of the system. In the appendix program, a picker is provided to select the language. If there is a need for reading the text in the default language without having to select the language first, it is recommended not to set locale in *SpeechOptions*.

Earlier in section 4.12.1, we had to adopt a strange approach to get all the supported locales. Unfortunately, on the Android platform, the processor is dealing with the asynchronous methods differently from other platforms. The items of the picker are added before the asynchronous method is completed. This means there is nothing in the picker. A solution is to postpone the whole process, which means not putting the initiation code in the constructor. The same code is moved into a *Clicked* event of the get locales button. This way, the locales are first obtained and then the items of the picker are added.



This is a text to speech demo.

en

Let me sing

ko; fr; ru; zh; hu; th; nb; da; tr; bs; sw; pt; vi; en; sv; bn; sk; el; hi; fi; bn; km; fr; uk; en; nl; sr; pt; de; si; ku; cs; pl; fil; it; ne; hr; zh; es; cy; ja; sq; yue; en; es; la; in; ca; ta; en; FirstOrDefault: ko

This function gives access to text to speech, using specified settings.

This is a text to speech demo.

de-DE

Ich sehe dich

ar-SA; cs-CZ; da-DK; de-DE; de-DE; de-DE; el-GR; en-AU; en-AU; en-AU; en-GB; en-GB; en-IE; en-US; en-US; en-US; en-US; en-ZA; es-ES; es-MX; fi-FI; fr-CA; fr-FR; fr-FR; fr-FR; he-IL; hi-IN; hu-HU; id-ID; it-IT; ja-JP; ja-JP; ja-JP; ko-KR; nl-BE; nl-NL; no-NO; pl-PL; pt-BR; pt-PT; ro-RO; ru-RU; sk-SK; sv-SE; th-TH; tr-TR; zh-CN; zh-CN; zh-CN; zh-HK; zh-TW; FirstOrDefault: ar-SA

This function gives access to tect to speech, using specified settings.

This is a text to speech demo.

zh-CN

Chinese Char

zh-CN; zh-CN; zh-CN; FirstOrDefault: zh-CN

This function gives access to text to speech, using specified settings.

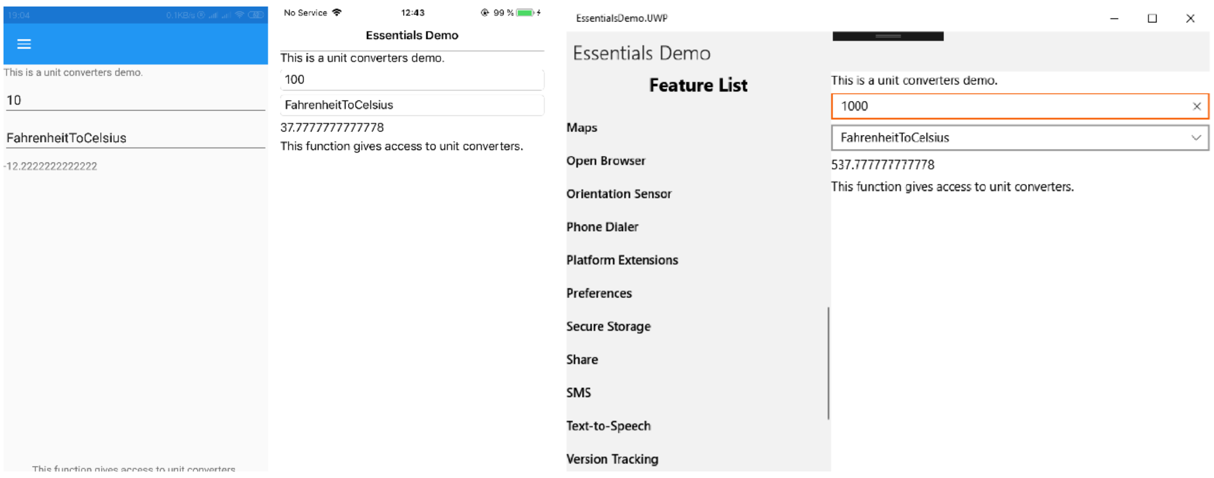
Figure 4.16: Comparison of Text to Speech

As can be shown in Figure 4.16, Android and iOS devices support many languages and the performance of the method *FirstOrDefault* is not desirable. The Raspberry Pi does not have a speaker so this class is not tested on Raspberry Pi.

4.13 Unit Converters

Xamarin.Essentials.UnitConverters provides 23 methods that can convert a unit into another. [34]

For instance, if there is a need for converting degrees to radians, the method *DegreesToRadian* can be chosen. By inputting a degree value to the parameter, the return value should be the corresponding radian value.



This is a unit converters demo.

10

FahrenheitToCelsius

-12.2222222222222

This function gives access to unit converters.

This is a unit converters demo.

100

FahrenheitToCelsius

37.7777777777778

This function gives access to unit converters.

This is a unit converters demo.

1000

FahrenheitToCelsius

537.777777777778

This function gives access to unit converters.

Figure 4.17: Comparison of Unit Converters

As shown in Figure 4.17, this class works nearly the same on each platform. Meanwhile, it also works on the Raspberry Pi.

4.14 Version Tracking

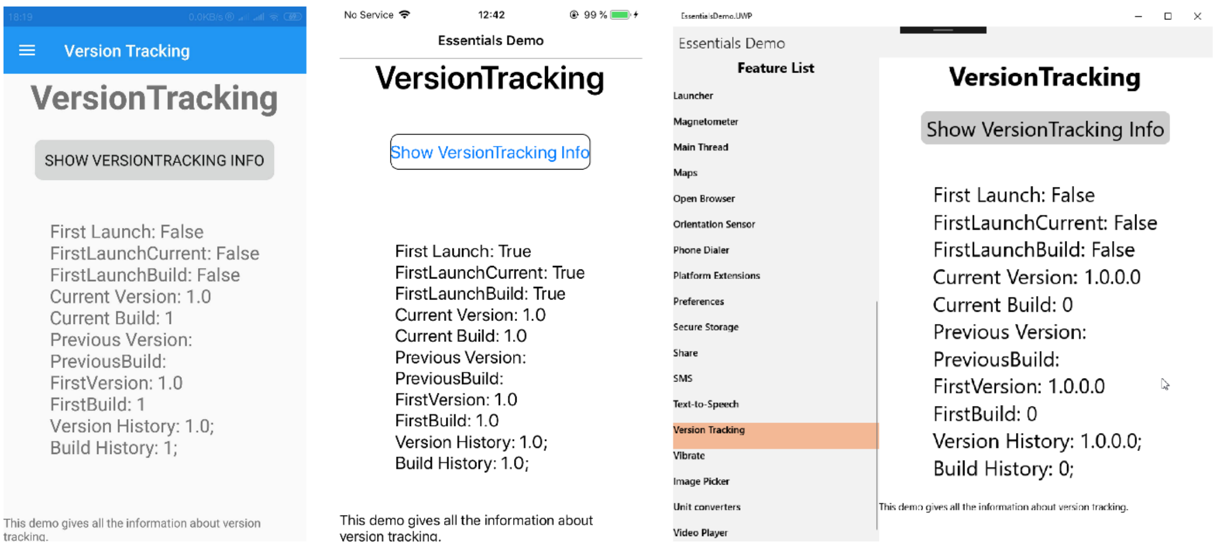
Xamarin.Essentials.VersionTracking gives access to version and some related information about the application. [35]

Most properties of this class are strings except *VersionHistory* and *BuildHistory*. They are lists, or specifically, *IEnumerable<string>*. In order to print the elements, an iterator or the method *foreach* can be used.

The meaning of most properties can be obtained from their name. For example, *VersionTracking.IsFirstLaunchEver* indicates whether this application is launched for the first time ever or not.

Information about version and build can be altered in each platform project in Visual Studio. For Android, information can be manipulated in *AndroidManifest.xml* in the Android project. For iOS, information can be changed in *Info.plist* in the iOS project. For UWP, information can be altered in *Package.appmanifest* in the UWP project.

As Figure 4.18 shows, this class successfully displays the version information on each platform.



```
First Launch: False
FirstLaunchCurrent: False
FirstLaunchBuild: False
Current Version: 1.0
Current Build: 1
Previous Version:
PreviousBuild:
FirstVersion: 1.0
FirstBuild: 1
Version History: 1.0;
Build History: 1;
```

This demo gives all the information about version tracking.

```
First Launch: True
FirstLaunchCurrent: True
FirstLaunchBuild: True
Current Version: 1.0
Current Build: 1.0
Previous Version:
PreviousBuild:
FirstVersion: 1.0
FirstBuild: 1.0
Version History: 1.0;
Build History: 1.0;
```

This demo gives all the information about version tracking.

```
First Launch: False
FirstLaunchCurrent: False
FirstLaunchBuild: False
Current Version: 1.0.0.0
Current Build: 0
Previous Version:
PreviousBuild:
FirstVersion: 1.0.0.0
FirstBuild: 0
Version History: 1.0.0.0;
Build History: 0;
```

This demo gives all the information about version tracking.

Figure 4.18: Comparison of Version Tracking

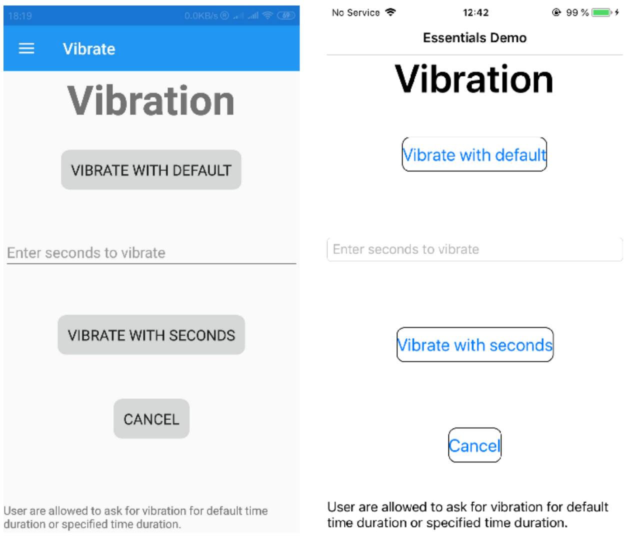
4.15 Vibration

Xamarin.Essentials.Vibration enables the application to vibrate for a certain period of time. [36]

By calling the method Vibrate, the device should vibrate for default vibration duration. Another option is to set the vibration duration in the parameter. The unit of time duration is in millisecond. The time duration can be set to a maximum of 5,000 milliseconds. A cancel method is also provided.

On iOS devices, Vibration in setting should be set to “On” so that the device is allowed to vibrate. Also, any vibration other than 0.5 seconds is not allowed on iOS devices. Therefore, whatever the code is, iOS devices will only vibrate for 0.5 seconds.

Figure 4.19 shows the demonstration page of this class. Unfortunately, Surface Pro 4 does not support vibration. The Raspberry Pi does not have vibrators, either. Therefore, this class is not tested on the UWP platform.



User are allowed to ask for vibration for default time duration or specified time duration.

User are allowed to ask for vibration for default time duration or specified time duration.

Figure 4.19: Comparison of Vibration

4.16 Video Player

This video player uses the default video player elements on the devices to perform an internal video player in an application. Video player is considered a fundamental function on mobile devices. However, an API for the video player is not included in *Xmarin.Essentials*.

4.16.1 Implementing Video Player API

In order to implement a video player, not only classes in the main project are required, but additional classes in the platform projects are also needed.

The classes needed for implementing the video player are all included in folders named *FormsVideoLibrary*. There is a *FormsVideoLibrary* folder in each project. This makes the code in the demo page simpler and easier to read. Furthermore, this makes it easier to copy the code for implementation to other projects. The code for implementation should work in other projects with small changes only in namespaces.

The class *VideoPlayer* in the main project defines the necessary properties needed in *VideoPlayerRenderer*. [37]

There is a *VideoPlayerRenderer* class in each platform project. *VideoPlayerRenderer* deals with the video player view along with the controllers on each platform. [37]

The abstract class *VideoSource* acts like a method library that provides three methods to specify the video source of the video player.

The class *VideroSourceConverter* rewrites the class *TypeConverter*. This class is invoked when the view of the application is written in XMAL and the *Source* property of a video player is set to a string. [38] In Video Player page of the appendix page, the video player is written directly in a cs file so this class is not invoked.

The class *VideoPicker* in each platform project extends the interface *IVideoPicker* in the main project. *VideoPicker* deals with events when picking the source file. [39]

4.16.2 Video Source

The classes *FileVideoSource*, *ResourceVideoSource*, and *UriVideoSource* make it possible to set a file, an application resource, and a URI as the source of the video player respectively.

On iOS devices, only videos taken by the device can be picked as a file source. Videos downloaded by TV application, for example, is not included in the image library. Therefore, they can not be picked. If a video is to be played as a resource, it must be put in the *Resource* folder in the iOS project and its *Build Action* property must be set as *BundleResource*. The resource video should be referred to as its relative path to the *Resource* folder. [40]

On Android devices, if a video is to be played as a resource, it must be put in a folder named "raw" in *Resource* folder in the Android project. Its *Build Action* property should be set as *AndroidResource*. The resource video can simply be referred to as its name as a string when coding. [40]

On UWP devices, only videos in images or videos folders of the device can be picked as file source of the video player. In order to allow the application to select videos from images and videos folder, *Pictures Library* and *Videos Library* should be selected under the tag *Capabilities* in the file *Package.appxmanifest* in the UWP project. In order to play a resource video, the video can simply be put anywhere in the UWP project and its *Build Action* property should be set as *Content*. In the appendix program, a video is put in a folder named *Video*. Accordingly, this video should be referred to as *Video/UWPApiVideo.mp4* when coding. [40]

If a web video is to be played, the URI of the web video should be specified when setting the *Source* property of the video player. This can be achieved by calling the static method *FromUri* from *VideoSource* class and set the URI as a string as the parameter.

4.16.3 Implementing Video Player in Demo Page

```
1.  public VideoPlayerDemo()
2.  {
3.      ...
4.      videoPlayer = new VideoPlayer { VerticalOptions = LayoutOptions.FillAndExpand };
5.      // videoPlayer.Source = VideoSource.FromUri("https://archive.org/download/BigBuckBunny_328/BigBuckBunny_512kb.mp4");
6.      // videoPlayer.Source = VideoSource.FromResource("Video/UWPApiVideo.mp4");
7.      // videoPlayer.Source = VideoSource.FromResource("UWPApiVideo.mp4");
8.      button_selectSource = new Button { Text = "Select Source" };
9.      button_selectSource.Clicked += Button_selectSource_ClickedAsync;
10.     ...
11. }
12. private async void Button_selectSource_ClickedAsync(object sender, EventArgs e)
13. {
14.     string filename = await DependencyService.Get<IVideoPicker>().GetVideoFileAsync();
15.
16.     if (!String.IsNullOrWhiteSpace(filename))
17.     {
18.         videoPlayer.Source = VideoSource.FromFile(filename);
19.     }
20. }
```

Listing 4.15: Implementing Video Player in Demo Page

Listing 4.15 shows a way to implement the video player using the API implemented in section 4.16.1. Line 1 to line 11 is the constructor of the demonstration page. Line 4 is initiating the video player. Line 5 shows a way to set a web video as the source using a URI. Line 6 demonstrates a way to set the resource video in the UWP package as the source. Line 7 sets the resource video in the Android and iOS package as the source. Note that the path and the name should be adjusted accordingly. Line 8 creates a button with a text “Select Source”. Line 9 binds the method from line 12 to line 20 to the *Clicked* event of the button created in line 8. Line 14 uses the class *DependencyService* to retrieve the platform-specific implementation of the interface *IVideoPicker*. As a result, each platform invokes its own video picker view to pick the video source. If the user clicks cancel when picking video source, the variable *filename* could be null. This error is prevented by line 16. Line 18 sets the picked file to be the video source.

4.16.1 Comparison on Different Platforms

Setting the property *VerticalOptions* is extremely important. The other elements mentioned in the other sections can simply be initiated without *VerticalOptions* but video player is an exception. If *VerticalOptions* is not set, the view of video player will not appear in the user interface of UWP application.

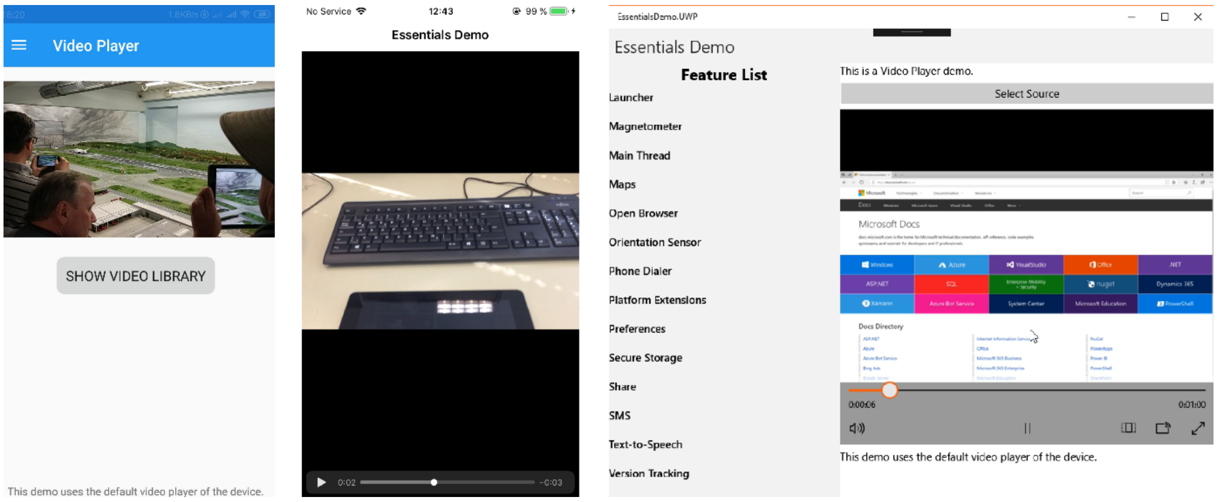


Figure 4.20: Comparison of Video Player

As Figure 4.20 shows, video player uses the default video player of each platform.

5 Limitations on iOS Devices

The upper half APIs in the API list [1] are discussed in this section. APIs with obvious variations on different platforms are focused in this section. Specific limitations on iOS devices are explained. Detailed introductions, implementation approaches, and comparisons on each platform can be found in the thesis written by Qin Lu. The APIs that are not mentioned are considered not having sizeable variations on the iOS platform.

5.1 Battery

According to the API documentation [41], there are altogether five categories of *PowerSource*: *AC*, *Battery*, *Unknown*, *Usb*, and *Wireless*. While on iOS devices, there are only two out of five possible return values: *AC* and *Battery*. Any kind of charging is sorted to *AC*.

5.2 Compass

Xamarin.Essentials.Compass provides its own low pass filter to deal with the jitter. However, this feature is not available on iOS. Even if the low pass filter is implemented, it will be ignored by the compiler of iOS. Fortunately, the reading of the compass does not undergo much jitter. If a filter has to be implemented anyway, a filter is introduced in the thesis written by Qin Lu.

5.3 Flashlight

On iPhone that installed iOS 11 or above, and iPhone that supports 3D-touch and installed iOS 10, the brightness of flashlight can be adjusted. Unfortunately, this feature is not taken into consideration by Xamarin.Essentials. It is not possible to set the brightness with this class.

5.4 Launcher

Any application that the launcher intends to open for the first time requires permission from the user. After first permission, the application is free to open the target application.

When the launcher encounters a URL, for example, “http://www.google.com”, iOS devices throws an exception. Whereas the default browser application is launched on other platforms.

6 Conclusion

In this thesis, half of the APIs of the package Xamarin.Essentials are supported with detailed introductions, as well as implementation approaches and features on different platforms. The performance of most APIs is tested. The untested ones result from the lack of test equipment. In addition, a video player is implemented. Meanwhile, a program for testing, as well as for demonstration is provided.

The main functions of the APIs serve well on Android, iOS, and UWP platforms. Firstly, it gives access to most sensors and various functions of the device. Secondly, cross platform programming, which is one of the most important features of Xamarin, is fully demonstrated through the investigation. Most codes are written in the main project once and they manage to run on each platform. Therefore, It is safe to say that Xamarin.Essentials is a desirable choice for cross platform development and is able to reduce the workload of developers to a large extent.

On the other hand, this package has its drawbacks, as well. Firstly, it does not provide internal solutions for some functions. For instance, the API for maps solely enables the application to jump to the default map application. It could have been more favorable if the map could be displayed inside the application. Secondly, some functions are somehow redundant. For example, *FileSystemHelper* provides extremely similar methods to some methods in the package *System.IO*. Moreover, the former methods are not as powerful as the latter. Thirdly, this package lacks APIs for some fundamental functions. For example, the video player implemented in section 4.16. As a result, a number of implementing classes in each platform project have to be written in order to invoke the default video player.

7 Outlook

For further research on Xamarin.Essentials, solving the following remaining problems could serve as a start.

Firstly, as explained in section 2.4, the author does not possess a Windows phone. As a result, some of the APIs cannot be tested.

Secondly, some of the APIs involve multithreading or asynchronous methods. The program is written on the basis of single threading. In other words, the asynchronous methods are viewed as ordinary methods. The possibility of multithreading can be investigated. Asynchronous methods have caused considerable problems in this project. For example, the complicated logic described in section 4.12.1 and 4.12.2 results from an asynchronous method.

Thirdly, the package Xamarin.Essentials is a fairly new tool and is still updating. During the process of this project, a few new APIs are intergraded into the package, such as unit converters. It is recommended that further researchers refer to the official documentation frequently.

8 Reference

[1] J. Montemagno, C. Dunn, N. Malcolm, C. Petzold and B. Umbaugh, "Xamarin.Essentials - Xamarin | Microsoft Docs," 22 April 2019. [Online]. Available: https://docs.microsoft.com/en-us/xamarin/essentials/. [Accessed 5 May 2019].

[2] G. Warren and T. G. Lee, "Overview of Visual Studio | Microsoft Docs," 19 March 2019. [Online]. Available: https://docs.microsoft.com/en-us/visualstudio/get-started/visual-studio-ide/. [Accessed 5 May 2019].

[3] Microsoft, "Compare Visual Studio Product Offerings | Visual Studio," 2019. [Online]. Available: https://visualstudio.microsoft.com/vs/compare/. [Accessed 05 May 2019].

[4] B. Wagner, N. Schonning, M. Wenzel, L. Latham and P. Onderka, "A Tour of C# - C# Guide | Microsoft Docs," 5 April 2019. [Online]. Available: https://docs.microsoft.com/en-us/dotnet/csharp/tour-of-csharp/. [Accessed 5 May 2019].

[5] C. Dunn, D. Britch, R. Anderson and A. Burns, "Introduction to Mobile Development - Xamarin | Microsoft Docs," 28 March 2017. [Online]. Available: https://docs.microsoft.com/en-us/xamarin/cross-platform/get-started/introduction-to-mobile-development. [Accessed 09 May 2019].

[6] Apple, "Xcode - Apple Developer," [Online]. Available: https://developer.apple.com/xcode/. [Accessed 10 May 2019].

[7] Apple, "Xcode on the Mac App Store," [Online]. Available: https://itunes.apple.com/us/app/xcode/id497799835/. [Accessed 10 May 2019].

[8] Microsoft, "Downloads | IDE, Code, & Team Foundation Server | Visual Studio," [Online]. Available: https://visualstudio.microsoft.com/downloads/. [Accessed 29 May 2019].

[9] C. Dunn, J. Johnson and D. Britch, "Installing Xamarin in Visual Studio 2019 - Xamarin | Microsoft Docs," 28 August 2018. [Online]. Available: https://docs.microsoft.com/en-us/xamarin/get-started/installation/windows. [Accessed 9 May 2019].

[10] C. Dunn, J. Montemagno, D. Britch, C. Petzold, M. Dickhaus and A. Chebukin, "Get Started with Xamarin.Essentials - Xamarin | Microsoft Docs," 4 November 2018. [Online]. Available: https://docs.microsoft.com/en-us/xamarin/essentials/get-started/. [Accessed 9 May 2019].

[11] Microsoft, "Installing Xamarin.iOS on Windows - Xamarin | Microsoft Docs," 16 April 2018. [Online]. Available: https://docs.microsoft.com/en-us/xamarin/ios/get-started/installation/windows/?pivots=win-vs2017. [Accessed 13 May 2019].

[12] L. O'Brien, C. Dunn, D. Britch, B. Umbaugh and T. Opgenorth, "Pair to Mac for Xamarin.iOS Development - Xamarin | Microsoft Docs," 29 May 2018. [Online]. Available: https://docs.microsoft.com/en-us/xamarin/ios/get-started/installation/windows/connecting-to-mac/index. [Accessed 24 May 2019].

[13] J. Montemagno, C. Dunn, C. Petzold, B. Umbaugh and M. Leibowitz, "Xamarin.Essentials: Magnetometer - Xamarin | Microsoft Docs," 04 November 2018. [Online]. Available: https://docs.microsoft.com/en-us/xamarin/essentials/magnetometer/. [Accessed 17 May 2019].

[14] J. Montemagno, C. Dunn and C. Petzold, "Xamarin.Essentials: OrientationSensor - Xamarin | Microsoft Docs," 4 November 2018. [Online]. Available: https://docs.microsoft.com/en-us/xamarin/essentials/orientation-sensor/. [Accessed 18 May 2019].

[15] J. Montemagno, C. Dunn and C. Petzold, "Xamarin.Essentials: MainThread - Xamarin | Microsoft Docs," 4 November 2018. [Online]. Available: https://docs.microsoft.com/en-us/xamarin/essentials/main-thread/. [Accessed 17 May 2019].

[16] Microsoft, "MainThread.BeginInvokeOnMainThread(Action) Method (Xamarin.Essentials) | Microsoft Docs," [Online]. Available: https://docs.microsoft.com/en-us/dotnet/api/xamarin.essentials.mainthread.begininvokeonmainthread?view=xamarin-essentials. [Accessed 26 May 2019].

[17] J. Montemagno and C. Dunn, "Xamarin.Essentials Map - Xamarin | Microsoft Docs," 2 April 2019. [Online]. Available: https://docs.microsoft.com/en-us/xamarin/essentials/maps. [Accessed 18 May 2019].

[18] Microsoft, "Map Class (Xamarin.Essentials) | Microsoft Docs," [Online]. Available: https://docs.microsoft.com/en-us/dotnet/api/xamarin.essentials.map/. [Accessed 25 May 2019].

[19] Microsoft, "NavigationMode Enum (Xamarin.Essentials) | Microsoft Docs," [Online]. Available: https://docs.microsoft.com/en-us/dotnet/api/xamarin.essentials.navigationmode/. [Accessed 18 May 2019].

[20] J. Montemagno, C. Dunn, D. P, B. Umbaugh and M. Leibowitz, "Xamarin.Essentials Open Browser - Xamarin | Microsoft Docs," 2 April 2019. [Online]. Available:

https://docs.microsoft.com/en-us/xamarin/essentials/open-browser/. [Accessed 18 May 2019].

[21] Microsoft, "Browser Class (Xamarin.Essentials) | Microsoft Docs," [Online]. Available: https://docs.microsoft.com/en-us/dotnet/api/xamarin.essentials.browser/. [Accessed 26 May 2019].

[22] J. L. Blanco, "A tutorial on SE(3) transformation parameterizations," 2019.

[23] J. Montemagno, C. Dunn, domagojmedo, B. Umbaugh and M. Leibowitz, "Xamarin.Essentials: Phone Dialer - Xamarin | Microsoft Docs," 4 November 2018. [Online]. Available: https://docs.microsoft.com/en-us/xamarin/essentials/phone-dialer/. [Accessed 19 May 2019].

[24] J. Montemagno and L. O'Brien, "Xamarin.Essentials Platform Extensions - Xamarin | Microsoft Docs," 13 March 2019. [Online]. Available: https://docs.microsoft.com/en-us/xamarin/essentials/platform-extensions/. [Accessed 19 May 2019].

[25] J. Dick and J. Montemagno, "Essentials/Xamarin.Essentials/Types/PlatformExtensions at master · xamarin/Essentials," 15 March 2019. [Online]. Available: https://github.com/xamarin/Essentials/tree/master/Xamarin.Essentials/Types/PlatformExtensions/. [Accessed 19 May 2019].

[26] J. Montemagno, L. McCarthy, C. Dunn, N. Schonning, M. Leibowitz and C. Petzold, "Xamarin.Essentials: Preferences - Xamarin | Microsoft Docs," 15 January 2019. [Online]. Available: https://docs.microsoft.com/en-us/xamarin/essentials/preferences/. [Accessed 19 May 2019].

[27] Microsoft, "Preferences Class (Xamarin.Essentials) | Microsoft Docs," [Online]. Available: https://docs.microsoft.com/en-us/dotnet/api/xamarin.essentials.preferences/. [Accessed 27 May 2019].

[28] J. Montemagno, C. Dunn, T. Brobbel, M. Leibowitz, J. Alt, C. Petzold and artemious7, "Xamarin.Essentials: Secure Storage - Xamarin | Microsoft Docs," 2 April 2019. [Online]. Available: https://docs.microsoft.com/en-us/xamarin/essentials/secure-storage/. [Accessed 19 May 2019].

[29] J. Montemagno and C. Dunn, "Xamarin.Essentials: Share - Xamarin | Microsoft Docs," 2 April 2019. [Online]. Available: https://docs.microsoft.com/en-us/xamarin/essentials/share/. [Accessed 19 May 2019].

[30] Microsoft, "ShareTextRequest Class (Xamarin.Essentials) | Microsoft Docs," [Online]. Available: https://docs.microsoft.com/en-us/dotnet/api/xamarin.essentials.sharetextrequest/. [Accessed 27 May 2019].

[31] J. Montemagno, C. Dunn, B. Umbaugh and M. Leibowitz, "Xamarin.Essentials: SMS - Xamarin | Microsoft Docs," 4 November 2018. [Online]. Available: https://docs.microsoft.com/en-us/xamarin/essentials/sms/. [Accessed 19 May 2019].

[32] Microsoft, "SmsMessage Class (Xamarin.Essentials) | Microsoft Docs," [Online]. Available: https://docs.microsoft.com/en-us/dotnet/api/xamarin.essentials.smsmessage/. [Accessed 27 May 2019].

[33] J. Montemagno, K. C. Dunn, B. Umbaugh and M. Leibowitz, "Xamarin.Essentials: Text-to-Speech - Xamarin | Microsoft Docs," 4 November 2018. [Online]. Available: https://docs.microsoft.com/en-us/xamarin/essentials/text-to-speech/. [Accessed 19 May 2019].

[34] J. Montemagno, "Xamarin.Essentials Unit Converters - Xamarin | Microsoft Docs," 13 March 2019. [Online]. Available: https://docs.microsoft.com/en-us/xamarin/essentials/unit-converters/. [Accessed 20 May 2019].

[35] J. Montemagno, C. Dunn, B. Umbaugh and M. Leibowitz, "Xamarin.Essentials: Version Tracking - Xamarin | Microsoft Docs," 4 November 2018. [Online]. Available: https://docs.microsoft.com/en-us/xamarin/essentials/version-tracking/. [Accessed 20 May 2019].

[36] X. V. -. X. |. M. Docs, "Xamarin.Essentials: Vibration - Xamarin | Microsoft Docs," 4 November 2018. [Online]. Available: https://docs.microsoft.com/en-us/xamarin/essentials/vibrate/. [Accessed 20 May 2019].

[37] D. Britch, C. Dunn and C. Petzold, "Creating the platform video players - Xamarin | Microsoft Docs," 12 February 2018. [Online]. Available: https://docs.microsoft.com/en-us/xamarin/xamarin-forms/app-fundamentals/custom-renderer/video-player/player-creation/. [Accessed 23 May 2019].

[38] D. Britch, C. Dunn, C. Petzold and B. Umbaugh, "Playing a Web video - Xamarin | Microsoft Docs," 12 February 2018. [Online]. Available: https://docs.microsoft.com/en-us/xamarin/xamarin-forms/app-fundamentals/custom-renderer/video-player/web-videos/. [Accessed 23 May 2019].

[39] D. Britch, C. Dunn and C. Petzold, "Accessing the device's video library - Xamarin | Microsoft Docs," 12 February 2018. [Online]. Available: https://docs.microsoft.com/en-us/xamarin/xamarin-forms/app-fundamentals/custom-renderer/video-player/accessing-library/. [Accessed 23 May 2019].

[40] D. Britch, C. Dunn, J. S. Smith, C. Petzold and B. Umbaugh, "Loading application resource videos - Xamarin | Microsoft Docs," 12 February 2018. [Online]. Available:

https://docs.microsoft.com/en-us/xamarin/xamarin-forms/app-fundamentals/custom-renderer/video-player/loading-resources/. [Accessed 23 May 2019].

[41] Microsoft, "BatteryPowerSource Enum (Xamarin.Essentials) | Microsoft Docs," [Online]. Available: https://docs.microsoft.com/en-us/dotnet/api/xamarin.essentials.batterypowersource/. [Accessed 28 May 2019].

9 List of images

10 List of Listings