WIRTSCHAFTSUNIVERSITÄT WIEN
Vienna University of Economics and Business

Bachelor Thesis

English title of the Bachelor Thesis	Developing mobile Windows Applications
Author **last name, first name(s)**	Scheuer Josef
Student ID number	0754162
Degree program	BaWiso06
Examiner **degree, first name(s), last name**	ao. Univ.Prof. Mag. Dr. Rony G. Flatscher

I hereby declare that

1. I have written this Bachelor thesis independently and without the aid of unfair or unauthorized resources. Whenever content was taken directly or indirectly from other sources, this has been indicated and the source referenced.
2. this Bachelor thesis has neither previously been presented for assessment, nor has it been published.
3. this Bachelor thesis is identical with the assessed thesis and the thesis which has been submitted in electronic form.

Date: 24.04.2012

Signature

Abstract

This bachelor thesis is an introduction to the development of Silverlight applications for Windows Phone. It gives an overview of the platform, the development tools and the concepts of developing and publishing an application.

The development environment is described in detail and the basics of the structure of an application is explained. Then the aspects of Silverlight on the basis of programming examples are discussed and finally, the process of deployment is presented.

Keywords: Windows Phone, Microsoft Silverlight, XAML, Visual Studio Express, XAP-archive, App Hub, Metro

Table of Contents

List of Figures

List of Listings

List of Abbreviations

A-GPS	Assisted Global Positioning System
API	Application Programming Interface
CLR	Common Language Runtime
GUID	Global Unique Identity
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
ID	Identification
IDE	Integrated Development Environment
ISETool	Isolated Storage Explorer Tool
LINQ	Language Integrated Query
MPNS	Microsoft Push Notification Service
MSDN	Microsoft Developer Network
OS	Operating System
RIA	Rich Internet Application
SDK	Software Development Kit
SIP	Software Input Panel
SQL	Structured Query Language
UI	User Interface
URI	Uniform Resource Identifier
VB.Net	Visual Basic .NET
WLAN	Wireless Local Area Network
XAML	Extensible Application Markup Language
XAP	XAML Application Package
XML	Extensible Markup Language
XNA	XNA is Not an Acronym[1]
XP	Extreme Programming

[1] *The name "XNA" originated out of the project's development name, Xbox New Architecture. Instead of being released under the Xbox name, the Xbox 360 was released (2005), and XNA came jokingly to stand for "XNA is Not an Acronym"* [Micr01].

1 Introduction

In the second half of 2010 Microsoft released a new operating system (OS) for smartphones, which was named Windows Phone 7 [Micr02]. The next major release for this OS was Windows Phone 7.5 (Codename “Mango”) in September 2011 [Micr03]. The OS is the successor of the Windows Mobile platform, but different in many ways. It is a completely new platform and represents a clean break from the Windows Mobile platform. Applications that were designed for Windows Mobile are incompatible [Zieg10]. The marketplace for Windows Mobile is closing in May 2012 and applications and games are only from developers or third-party marketplaces available [Micr04].

So the Windows Phone platform is a completely fresh start for Microsoft in the smartphone market. The Phone is primary aimed for the consumer market and the user is the focus. Microsoft also integrates popular products from other platforms such as Xbox, Zune, Office and Bing to Windows Phone [Micr02].

The realignment is also a step towards to “closed ecosystems” like the iPhone. So for the users of Windows Phone, the marketplace is the only way to legally download and install applications, and developers can only release their applications via the App Hub in the marketplace. In the submission process, the application goes through a strict screening process of Microsoft.

Furthermore, each application on the phone is running in its own sandbox and can only be written in managed code [cf. Getz11 p.26]. “The sandbox concept is used to provide an environment where applications have limited privileges and don’t have access to the file system, other applications and system resources that could be exploited” [Shin11].

Also the manufacturers of the smartphones have hardware requirements to run the OS of the phone.

1.1 Motivation and Goal

Smartphones are increasingly becoming part of our lives. The technology enables it to be constantly connected and interact with friends and associates.

With the Windows Phone platform Microsoft wants to make new gains in the smartphone market, despite the success of iPhone and Android. The Windows Phone OS also brings changes to the developers. New development tools are available with which the applications can easily and comfortably be created.

In this paper the author wants to give an overview of the platform and an introduction to the development of applications for the Windows Phone. For the development of the applications the Silverlight or the XNA Framework can be used. The XNA Framework is intended for the development of games, while the Silverlight Framework is used for event driven applications. Also a combination of two frameworks is possible [MSDN01]. In this work the Silverlight Framework will be presented. The programming language C# was chosen because of the similarities with other object-orientated languages like Java.

During the creation of this work, the latest version of the operating system is Windows Phone OS 7.1 with the update 7.10.8107.79 [Micr05]. The tools that were used for the development are the Windows Phone Software Development Kit (SDK) 7.1 which was published on the 25th September 2011 [Micr06].

1.2 Version Numbers and Names

With the major update from Windows Phone 7 to Windows Phone 7.5 the latter one has received two names. Windows Phone 7.5 (Codename “Mango”) is the marketing name, while the version for the OS is Windows Phone OS 7.1. Also the names for the development tools have changed. The Windows Phone Developer Tools was renamed to Windows Phone SDK. Also the “7” from the platform name has disappeared. It is simply called Windows Phone [cf. Getz11 p.29].

1.3 Structure of this Paper

In Chapter 2 the Application Platform, the Phone and Silverlight for Windows Phone will be described.

Chapter three outlines the development tools Visual Studio Express and the Emulator. Also, the structure and the files of the project are explained in more detail. Furthermore, the XAP-archive and the application life cycle is presented.

Chapter four is the biggest part and deals with the development of Silverlight applications for Windows Phone. It contains programming examples to illustrate the concepts and techniques of Silverlight technology. At the beginning of the chapter, the orientation and the layout elements are presented. Then the smartphone controls to build an user interface, the concept of navigation through the phone pages and data binding are covered. Also the isolated storage and push notifications are discussed.

After the development of applications the testing of an application is described in chapter five. It contains a short description of the debugging feature of Visual Studio and the use of the Marketplace Test Kit. In addition, a unit testing framework of the open source community CodePlex is presented.

Chapter six deals with the deployment of an application. There is a brief description of the registration process in the App Hub and the submission and certification is discussed.

2 Phone and Application Platform

This chapter deals with the Windows Phone, the hardware requirements, the software architecture, the Metro design and the Silverlight technology for Windows Phone.

2.1 The Windows Phone

One goal for the latest smartphone of Microsoft is to deliver a phone which integrates the things people really want to do and puts those things in front of the user. On the Start screen of the phone, the user will find “live tiles” that show real-time content such as social media updates, weather data or contacts. They will be updated on the fly with real information [Micr02]. Each installed application on the phone can be pinned to the Start screen.

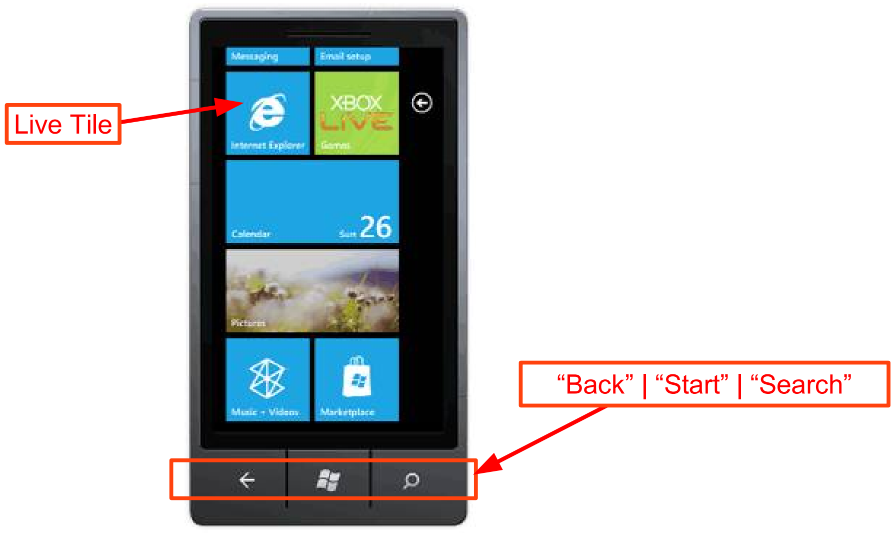


Figure 1: The Windows Phone [Ciap11]

A Windows Phone also must have three hardware buttons, which are named “Back”, “Start” and “Search”. The Back button will take the user one step back in the navigation sequence. It has a similar function as the back button in a browser. The Start screen of the Phone can be viewed with the Start button and with a click on the Search button, the Bing search engine of Microsoft is launched.

2.2 Hardware Requirements

Windows Phone devices have to meet technical requirements to run the OS. Microsoft has made these technical specifications for hardware manufacturers to ensure that there is a consistent user experience across all devices. In the past, Windows Mobile was often installed on devices whose hardware was not designed for the OS, which has impacted a negative user experience. The Phone needs [cf. Getz11, 42-46]:

- a common set of hardware controls and buttons that include the Start, Search, and Back button,
- a capacitive touch screen with at least four contact points,
- a screen with a resolution of 480 x 800 pixels,
- A-GPS, accelerometer, ambient light sensor and proximity sensor,
- a camera with five mega pixels or more,
- a Direct X9 acceleration and the phone is working with the current ARM-v7 processor with 1 GHz or more.

The Windows Phone supports known touch gestures such as [MSDN02]:

- **Tap:** activation and release of the screen with one finger
- **Double-Tap:** tow tabs in succession
- **Hold:** A finger touches the screen constantly and remains a longer time on the same position.
- **Drag:** The screen is activated with a finger, moves on the screen and will then be released.
- **Flick:** A finger drags across the screen and is lifted up without stopping (e.g. scrolling in a list).
- **Pinch:** Two fingers press on the screen and move around (e.g. rotate a picture).
- **Scale:** Two fingers are spread apart or brought together to zoom in or out.

2.3 Software Architecture

With Windows Phone the software architecture has changed fundamentally. Figure 2 shows an overview of the structure and then the various components are described in more detail.

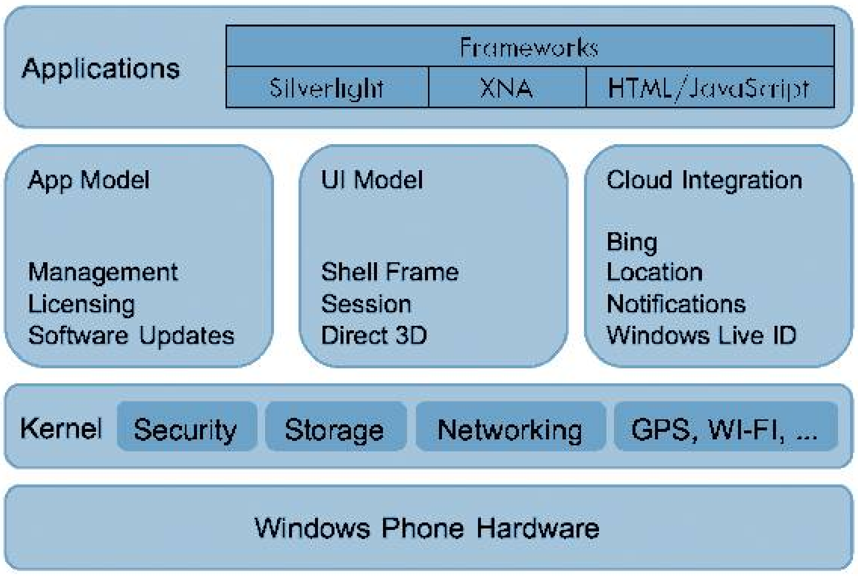


Figure 2: Software Architecture of the Windows Phone [cf. Pren01]

Hardware & Kernel

On the bottom there is the Windows Phone Hardware which was covered in the chapter 2.2. On the top of the Hardware there is the Kernel that provides basic services like Security, Storage, Networking and the device drivers for GPS, WLAN, accelerometer, etc. Details about the structure, the drivers and APIs (native and managed) are not publicly accessible, so a normal application developer will not come into contact with this [cf. Getz11 p.53].

App Model

The App Model handles the management, the licensing, the isolation of the installed applications and the software updates. For the deployment of an application the XAML Application Package (XAP-archive). An application for Windows Phone is not a native application and will be loaded and executed in a host process. It is running in a sandbox with limited privileges [cf. Getz11 p.49].

UI Model

The UI Model contains the Shell Frame Management and the Navigation Model. The Shell Frame Management is responsible for the composing of the user interface (Figure 3).

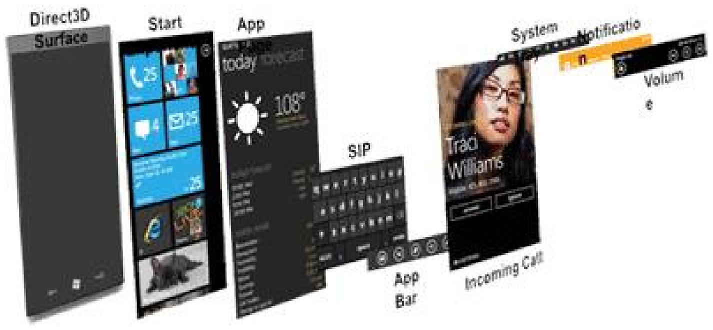


Figure 3: Shell Frame [Pren02]

In the Navigation Model every application is stored in a session. As already mentioned, the navigation of the Phone is similar to the navigation of web pages in a browser. A manipulation of the session information is not possible [cf. Getz11 p.51].

Cloud Integration

Windows Phone is very focused on the cloud. The Cloud Integration provides services like Windows Live ID to synchronize e-mails, the calendar and the contacts if the Live ID is connected with a hotmail account. Also it includes the Bing search, location information and a notification service. Windows Azure is the cloud computing platform of Microsoft that provides various services (e.g. a SQL database) [cf. Getz11 p.52].

Applications

On the very top there is the Application Runtime which is built on the .NET Common Language Runtime (CLR). This is the layer which is designed for the developer. It includes the Silverlight and XNA framework which are used to develop applications for the phone. Also HTML/Javascript is supported, but can only be executed in the browser. In contrast to Windows Mobile it is not possible to install drivers on the phone. The access to the phone is strongly restricted. The functions of the phone can only be used via API calls [cf. Getz11 p.48].

2.4 The Metro Design

Metro is a codename for a typography-based design language created by Microsoft. Is was originally developed and used for the Windows Phone 7 interface. The inspiration for the design comes from the public transport system which has the focus on displaying important information. The Metro design emphasis on simplicity and should enable a unique experience to the user. After the release of Windows Phone Microsoft also included the Metro principle to Xbox 360 and Windows 8 [Trip12].

2.5 Silverlight for Windows Phone

Silverlight is a cross-browser, cross platform technology for writing and running rich internet applications (RIA) for the web, the desktop and the Windows Phone [MSDN03]. For the Windows Phone OS 7.1 Silverlight 4 is used [MSDN04]. Silverlight is a subset of the .NET framework and includes a mini-Common Language Runtime (CLR) [Hube10]. Silverlight applications can only be written in compiled, managed code. Managed code[2] is developed in .NET framework and the code can only be executed under the management of the CLR (Figure 4) [Meha09]. The applications are hosted within a web server, web page or the mobile device. Therefore Silverlight uses the XAP-file that contains the .NET application code [Soda01]. The XAP-file is explained in chapter 3.5 (The XAP-File).

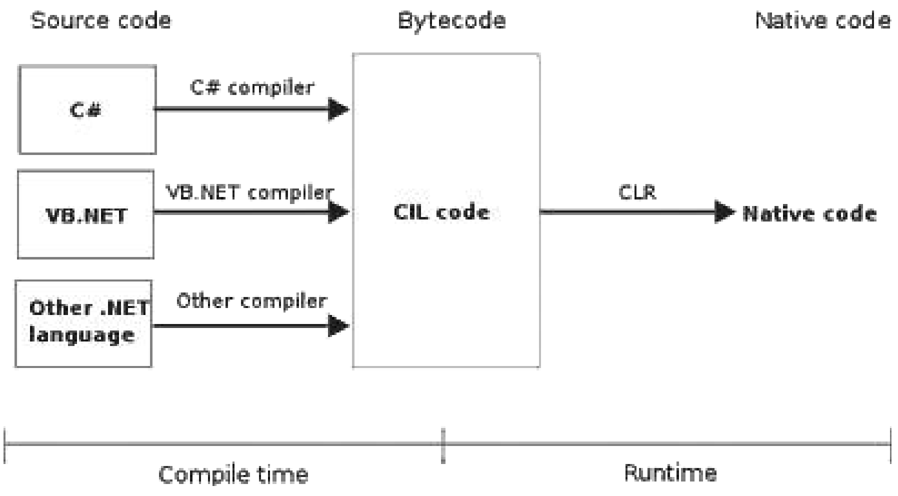


Figure 4: Managed Code [Wiki01]

In Silverlight for Windows Phone the programming language VB.Net and C# (C Sharp) can be used. Furthermore, in the development process of Silverlight there is a strict distinction between the UI and the program logic. It uses the Extensible Application Markup Language (XAML) to define the UI and the code-

[2] Unmanaged Code is developed outside the .NET framework and the applications do not run under the control of CLR (e.g. C++ can be used to write such applications) [Meha09].

behind (C#) defines the application logic. So there are always two files that belong together (a file with a *.xaml* extension for the UI and a *.xaml.cs* for the code-behind). The markup (UI) is joined to the code-behind file through the definition of a partial class. With a partial class you can split the definition of a class over two or more files. The parts of one class are combined when the application is compiled [MSDN05].

2.5.1 XAML and Code-Behind

XAML is a declarative XML-based language and is used to initialize structured values and objects to create visible UI elements such as buttons, text fields, etc. The elements are defined in an object element tag in angle brackets. Listing 1 shows a simple `Button` within a `Grid`. The elements also contain additional attributes for their appearance e.g. the elements have a `Name`, a literal distance (`Margin`) or the `Button` has a `Content`. An attribute is defined with the attribute syntax followed by an operator (=) and the value in a string with quotation marks.

```
        <Grid x:Name="ContentPanel" Grid.Row="1" Margin="12,0,12,0">
            <Button Content="Button" Height="72" Margin="159,149,0,0"
                    Name="button1" Width="160" />
        </Grid>
```

Listing 1: Grid and Button-Definition in XAML

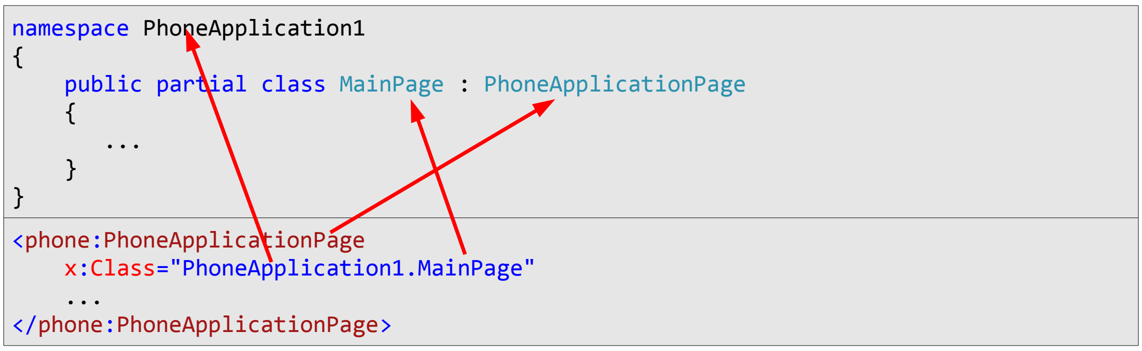
The elements in XAML are mapped to CLR object instances and the attributes become the properties and events of the object [Anat08].

Listing 2 displays an excerpt of an XAML file of a Windows Phone Application that was generated by Visual Studio. The project was named PhoneApplication1. By default, the file named *MainPage.xaml* is the starting page of the application. The opening tag indicates that the `MainPage` will derive from the `PhoneAppliationPage` (which derives from Page Class). The `x:Class` attribute defines the code-behind. The first `xmlns` namespace declares the file as a XAML document. The second namespace (`xmlns:x`) is required for elements with a `x` prefix (e.g. `x:Class` is one of them). The two namespaces are standard for a Silverlight application.

```
<phone:PhoneApplicationPage
    x:Class="PhoneApplication1.MainPage"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    ...
</phone:PhoneApplicationPage>
```

Listing 2: XAML File (Excerpt) - Windows Phone Application Template

In the C#-Code (*MainPage.xaml.cs*) you will see that its class is `MainPage` and it inherits from the `PhoneApplicationPage`. Also the used namespace which is the project name will be found (`PhoneApplication1`). In Listing 3 also the connection of the XAML and C#-Code is visually displayed.



```
namespace PhoneApplication1
{
    public partial class MainPage : PhoneApplicationPage
    {
        ...
    }
}
```

```
<phone:PhoneApplicationPage
    x:Class="PhoneApplication1.MainPage"
    ...
</phone:PhoneApplicationPage>
```

Listing 3: Connection of XAML and Code-Behind

3 Basics

In this chapter the integrated development environment (IDE) will be described and the famous “Hello World” application is built and run. Also the automatically generated files of Visual Studio and the structure of an Windows Phone project will be discussed in a greater detail. And finally, the XAP-file and the application life cycle are presented.

3.1 Windows Phone SDK 7.1

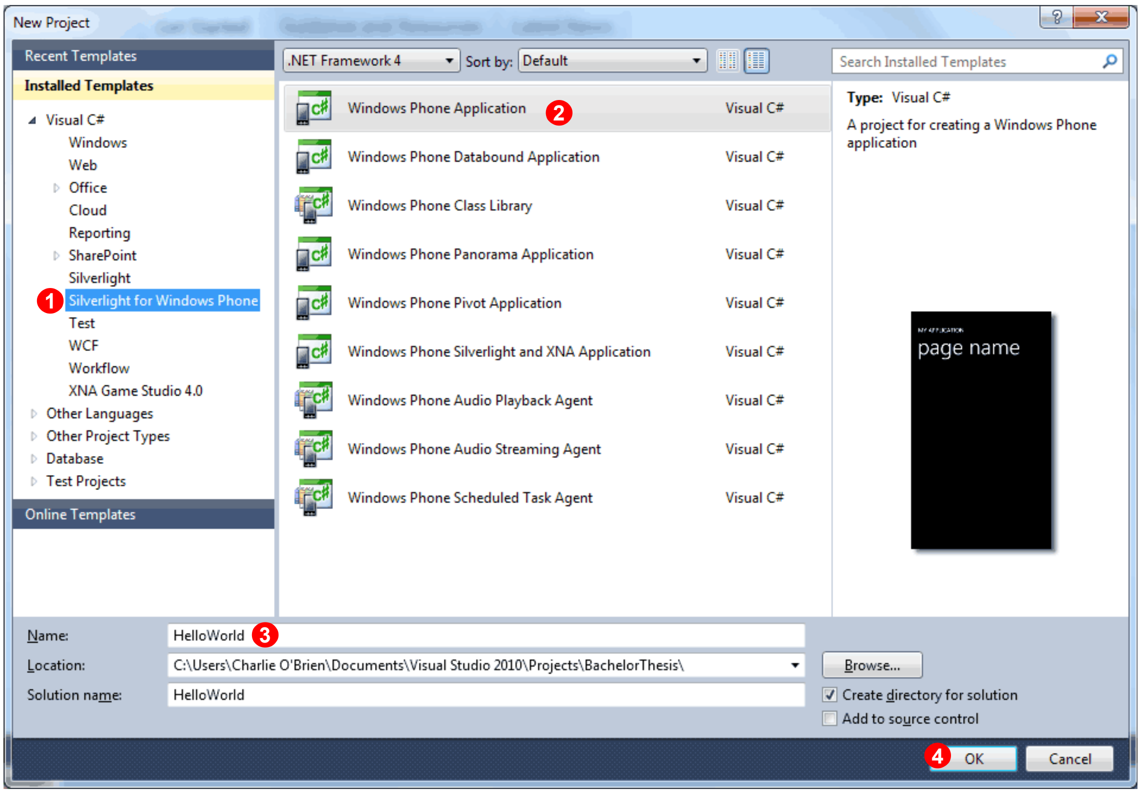
The Windows Phone Software Development Kit (SDK) 7.1 contains all the tools which are needed to develop applications for the Windows Phone. It is for free and can be obtained from: http://create.msdn.com/en-US/.

This work will cover Visual Studio 2010 Express and the Windows Phone Emulator to develop Silverlight applications. Expression Blend is a design tool to create a graphical, XAML-based interface. Each Silverlight project of Visual Studio can be opened in Expression Blend to edit the UI. Expression Blend is not described in detail, because the author focuses on the development of application with code-behind and their aspects.

When the installation process of the SDK has started and if there is already a version of Visual Studio 2010 (e. g. Professional) on the system then the tools of the Windows Phone SDK will be integrated in the existing version, assuming the same language is used. In this paper Visual Studio 2010 Professional with the integrated SDK tools is used. For the further work Visual Studio 2010 Express for Windows Phone is referred as Visual Studio.

3.2 Visual Studio Express

When the installation process of the SDK is successfully completed, Visual Studio can be launched. In the menu item *File → New Project* a new project can be created. Then a New Project dialog box appears (Figure 5). Visual Studio provides for each programming language several project templates.


New Project
Recent Templates
Installed Templates
Visual C#
Windows
Web
Office
Cloud
Reporting
SharePoint
Silverlight
1
Silverlight for Windows Phone
Test
WCF
Workflow
XNA Game Studio 4.0
Other Languages
Other Project Types
Database
Test Projects
Online Templates
.NET Framework 4
Sort by: Default
Search Installed Templates
Windows Phone Application
2
Visual C#
Windows Phone Databound Application
Visual C#
Windows Phone Class Library
Visual C#
Windows Phone Panorama Application
Visual C#
Windows Phone Pivot Application
Visual C#
Windows Phone Silverlight and XNA Application
Visual C#
Windows Phone Audio Playback Agent
Visual C#
Windows Phone Audio Streaming Agent
Visual C#
Windows Phone Scheduled Task Agent
Visual C#
page name
3
Browse...
4
OK
Cancel

Type: Visual C#

A project for creating a Windows Phone application

Name: HelloWorld

Location: C:\Users\Charlie O'Brien\Documents\Visual Studio 2010\Projects\BachelorThesis\

Solution name: HelloWorld

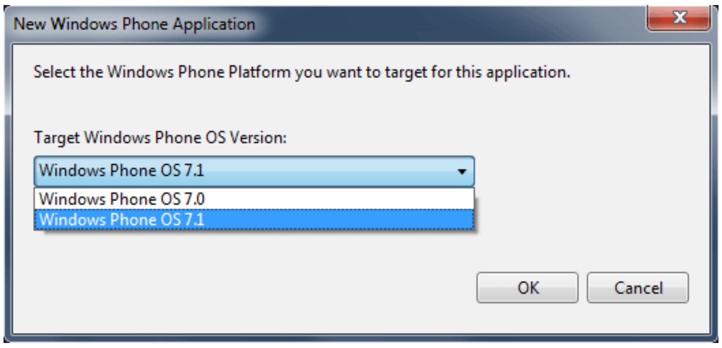
[✓] Create directory for solution

[ ] Add to source control

Figure 5: New Project Dialog - Visual Studio

Under *Visual C# → Silverlight for Windows Phone(#1, Figure 5)*, there are several templates for the Windows Phone. For the "Hello World" example and for all further examples in this work the template *Windows Phone Application (#2, Figure 5)* is used. Also the templates *Windows Phone Databound Application*, *Windows Phone Panorama Application* and *Windows Phone Pivot Application* are discussed during this work. *Windows Phone Class Library* is just a simple C# class.

After renaming the project with "HelloWorld" (#3, Figure 5) and confirming with the *OK* button (#4, Figure 5), a dialog appears where the OS of the Windows Phone for which you want to develop can be selected (Figure 6). In this work always the latest version (Windows Phone OS 7.1) is used.

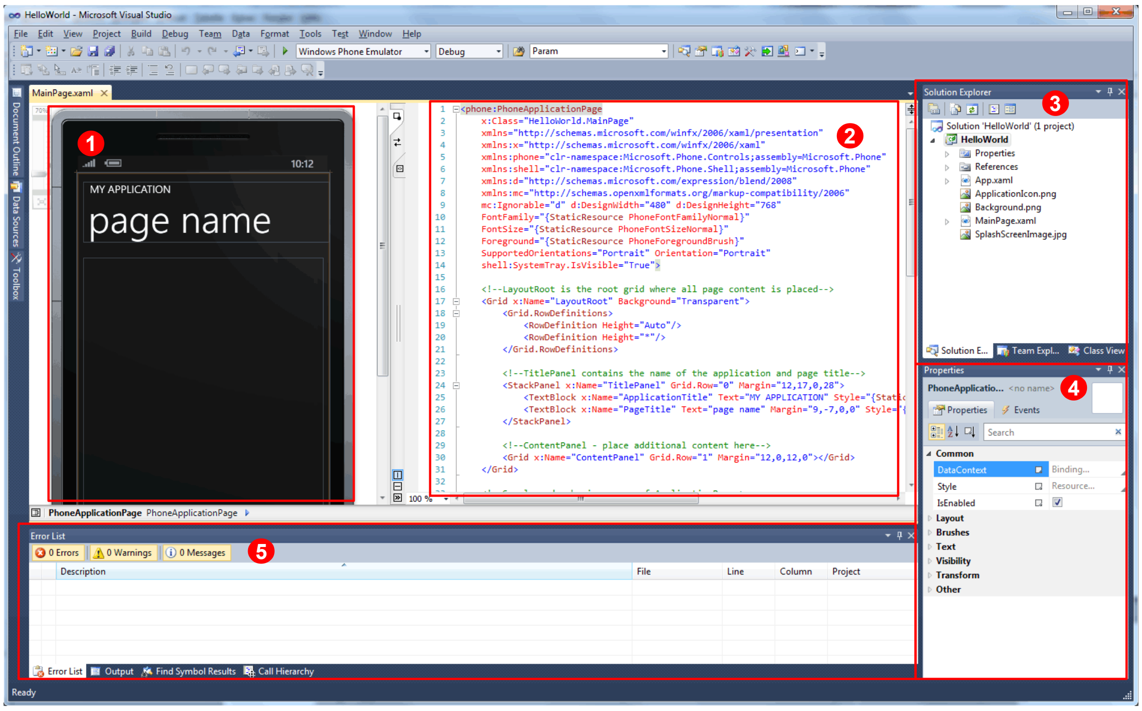


Select the Windows Phone Platform you want to target for this application.

Target Windows Phone OS Version: Windows Phone OS 7.1

Figure 6: Choose Operating System

Then, after pressing the *OK* button, Visual Studio will create a new project. When a new project in Visual Studio is loaded, it creates all necessary files to start with the development of applications. By default, Visual Studio displays a *Design Viewer* (#1, Figure 7), a *XAML-Editor* (#2, Figure 7), a *Solution Explorer* (#3, Figure 7), a *Properties Window* (#4, Figure 7) and an *Error List* where Errors, Warnings and Messages are displayed during the development process (#5, Figure 7).



```
<phone:PhoneApplicationPage
    x:Class="HelloWorld.MainPage"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:phone="clr-namespace:Microsoft.Phone.Controls;assembly=Microsoft.Phone"
    xmlns:shell="clr-namespace:Microsoft.Phone.Shell;assembly=Microsoft.Phone"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    mc:Ignorable="d" d:DesignWidth="480" d:DesignHeight="768"
    FontFamily="{StaticResource PhoneFontFamilyNormal}"
    FontSize="{StaticResource PhoneFontSizeNormal}"
    Foreground="{StaticResource PhoneForegroundBrush}"
    SupportedOrientations="Portrait" Orientation="Portrait"
    shell:SystemTray.IsVisible="True">

    <!--LayoutRoot is the root grid where all page content is placed-->
    <Grid x:Name="LayoutRoot" Background="Transparent">
        <Grid.RowDefinitions>
            <RowDefinition Height="Auto"/>
            <RowDefinition Height="*"/>
        </Grid.RowDefinitions>

        <!--TitlePanel contains the name of the application and page title-->
        <StackPanel x:Name="TitlePanel" Grid.Row="0" Margin="12,17,0,28">
            <TextBlock x:Name="ApplicationTitle" Text="MY APPLICATION" Style="{Stati
            <TextBlock x:Name="PageTitle" Text="page name" Margin="9,-7,0,0" Style="{
        </StackPanel>

        <!--ContentPanel - place additional content here-->
        <Grid x:Name="ContentPanel" Grid.Row="1" Margin="12,0,12,0"></Grid>
    </Grid>
```

Figure 7: Visual Studio Project – “Hello World” Start Screen

On the left side of the main window there is a *Toolbox* tab. With a click on the tab, the *Toolbox* window is displayed. When the pin icon is clicked, the toolbar is shown at all times in the main windows (Figure 8). The *Toolbox* contains the most UI elements. An UI element from the Toolbox can easily be added via drag and drop into the designer.

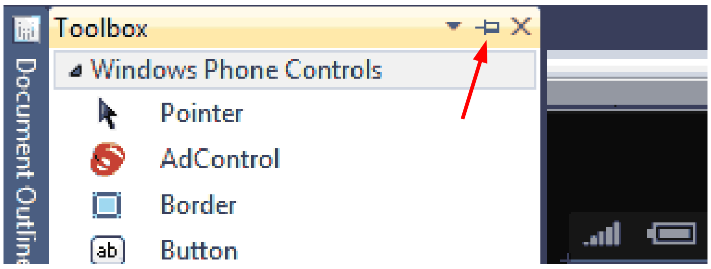


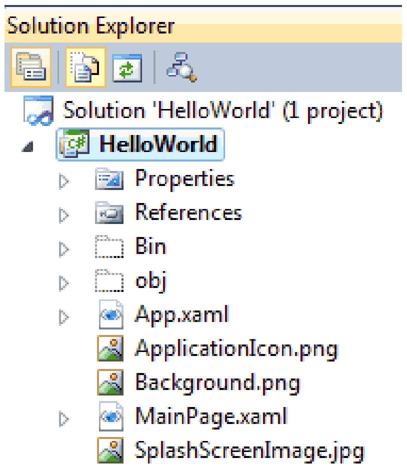
Figure 8: Toolbox

3.3 Structure of the Project

The Solution Explorer of a Windows Phone project shows the contents of a project. In this chapter the structure of the "HelloWorld" example is discussed in a greater detail.

3.3.1 The Solution Explorer

Figure 9 shows the Solution Explorer of the "Hello World" project. In the menu item *Project → Show All Files* all files in the explorer can be shown.

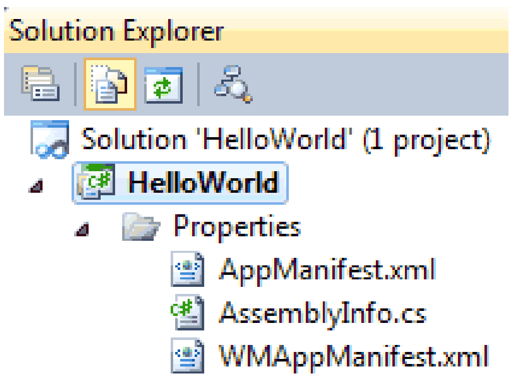


Solution Explorer
Solution 'HelloWorld' (1 project)
HelloWorld
Properties
References
Bin
obj
App.xaml
ApplicationIcon.png
Background.png
MainPage.xaml
SplashScreenImage.jpg

Figure 9: Solution Explorer - "Hello World" Application

3.3.2 Properties Folder

The folder Properties (Figure 10) contains three files: *AppManifest.xml, AssemblyInfo.cs* and *WMAppManifest.xml*.



Solution Explorer
Solution 'HelloWorld' (1 project)
HelloWorld
Properties
AppManifest.xml
AssemblyInfo.cs
WMAppManifest.xml

Figure 10: Properties Folder - Solution Explorer

The *AppManifest.xml* is required to generate the application package (XAP-file). If the file is opened in the project, it is still almost empty. When Visual Studio is building an application, the file will be filled with content. Thus a detailed description of the file can be found in chapter 3.5 (The XAP-File).

The next file is the *AssemblyInfo.cs*. It contains meta data about the application e.g.: title, version or a copyright note (Listing 4).

```
[assembly: AssemblyTitle("HelloWorld")]
...
[assembly: AssemblyCopyright("Copyright ©  2012")]
...
[assembly: AssemblyVersion("1.0.0.0")]
[assembly: AssemblyFileVersion("1.0.0.0")]
[assembly: NeutralResourcesLanguageAttribute("en-US")]
```

Listing 4: Excerpt AssemblyInfo.cs

The last file in the *Properties* folder is the *WMAppManifest.xml*. This file is also included in the XAP-file (chapter 3.5). It is used in the Windows Phone Marketplace submission process and for installation on the device [MSDN06]. Listing 5 shows the *WMAppManifest.xml* for the “Hello World” project. The `Deployment` element describes the used namespace and the platform version. The `App` element contains information such as the `ProductID`, `Version`, type of application (`Genre`), etc. In the `IconPath` element the icon for the application list on the phone is defined. The `Capabilities` element contains information about the used functions of the application (e.g. the microphone, sensors, push notification, etc). The `Task` element describes the first page that will be loaded when the application starts (by default it is the dummy page *MainPage.xaml*). Finally, in the *Tokens* element the image for the life tile of the application is defined.

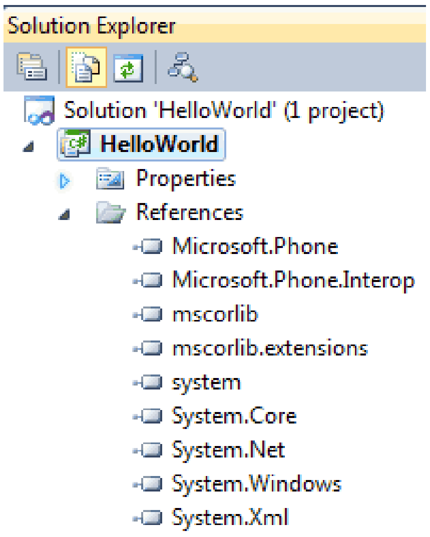
```
<?xml version="1.0" encoding="utf-8"?>

<Deployment xmlns="http://schemas.microsoft.com/windowsphone/2009/deployment"
            AppPlatformVersion="7.1">
  <App xmlns="" ProductID="{4a8de78d-148c-4d00-b265-06f88ff4cbd6}"
     Title="HelloWorld" RuntimeType="Silverlight" Version="1.0.0.0"
     Genre="apps.normal"  Author="HelloWorld author" Description="Sample description"
       Publisher="HelloWorld">
    <IconPath IsRelative="true" IsResource="false">ApplicationIcon.png</IconPath>
    <Capabilities>
      <Capability Name="ID_CAP_GAMERSERVICES"/>
      <Capability Name="ID_CAP_IDENTITY_DEVICE"/>
      ...
      <Capability Name="ID_CAP_APPOINTMENTS"/>
    </Capabilities>
    <Tasks>
      <DefaultTask  Name ="_default" NavigationPage="MainPage.xaml"/>
    </Tasks>
    <Tokens>
```

```
      <PrimaryToken TokenID="HelloWorldToken" TaskName="_default">
        <TemplateType5>
          <BackgroundImageURI IsRelative="true"
                              IsResource="false">Background.png</BackgroundImageURI>
          <Count>0</Count>
          <Title>HelloWorld</Title>
        </TemplateType5>
      </PrimaryToken>
    </Tokens>
  </App>
</Deployment>
```

Listing 5: WMAppManifest.xml - "Hello World" project

3.3.3 References Folder

The *References* folder as shown in Figure 11 contains the references to the used Silverlight Class Libraries.



Solution 'HelloWorld' (1 project)
- HelloWorld
  - Properties
  - References
    - Microsoft.Phone
    - Microsoft.Phone.Interop
    - mscorlib
    - mscorlib.extensions
    - system
    - System.Core
    - System.Net
    - System.Windows
    - System.Xml

Figure 11: References Folder - Solution Explorer

3.3.4 “Bin” and “obj” Folder

When the project is built by Visual Studio the *Bin* folder will contain the files for the deployment of the application and the *obj* folder contains files which are used for compiling the application.

3.3.5 Images

The project also contains three images. The *ApplicationIcon.png* and the *Background.png* have already been described in chapter 3.3.2 in the *WMAppManifest.xml*. The third image named *SplashScreenImage.png* is displayed on the device or emulator, when the program is initializing.

3.3.6 Silverlight files

By default, Visual Studio creates two major Silverlight files (Figure 12). The first file is the App Class which consists of the *App.xaml* and *App.xaml.cs*. The Page Class includes the *MainPage.xaml* and the *MainPage.xaml.cs* and is the class which contains the visuals on the screen. It is also referred to as page.

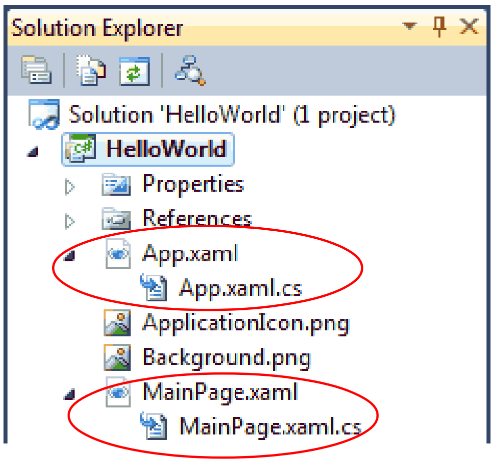
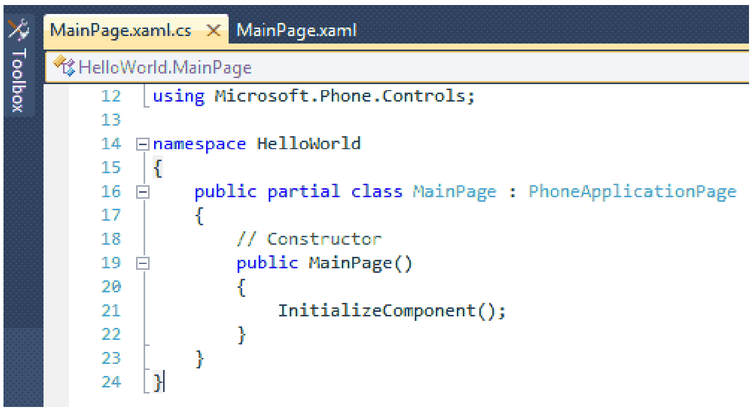


Figure 12: Silverlight Files

With a double click on a *.xaml* file the XAML-Editor (Figure 7) is opened and with a double click on a *.xaml.cs* the C#-Editor is opened.



```
using Microsoft.Phone.Controls;

namespace HelloWorld
{
    public partial class MainPage : PhoneApplicationPage
    {
        // Constructor
        public MainPage()
        {
            InitializeComponent();
        }
    }
}
```

Figure 13: C#-Editor

When the program starts, the App class creates an object of the type `PhoneApplicationFrame` [cf. Getz11 p.104]. This is the top level container for the entire application. Then, in this frame the page objects with the contents are displayed. By default, the application will navigate automatically to the *MainPage.xaml* which is defined in the *WMAppManifest.xml*. The developer can create multiple pages to present the content [MSDN07]. A new page can be created in the menu item *Project → Add New Item...* .

The App Class

The App class is the entry point for the application. Listing 6 shows the XAML-Code for the “Hello World” project. The root element is `Application` where the `App` class will be derived from. The `x:Class` defines the code-behind file. It specifies that a class named `App` in the namespace `HelloWorld` derives from the Silverlight Application class. The following two XML namespaces were already described in chapter 2.5.1 (XAML and Code-Behind). The other two namespaces (`xmlns:phone` and `xmlns:shell`) are unique to the phone. Then, global resources such as colour schemes or brushes for the entire application can be defined. The file also contains events for the application life cycle, which is covered in chapter 3.6 (The Application Life-Cycle).

```
<Application
    x:Class="HelloWorld.App"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:phone="clr-namespace:Microsoft.Phone.Controls;assembly=Microsoft.Phone"
    xmlns:shell="clr-namespace:Microsoft.Phone.Shell;assembly=Microsoft.Phone">

    <!--Application Resources-->
    <Application.Resources>
    </Application.Resources>
    <Application.ApplicationLifetimeObjects>
        <!--Required object that handles lifetime events for the application-->

        <shell:PhoneApplicationService
            Launching="Application_Launching" Closing="Application_Closing"
            Activated="Application_Activated" Deactivated="Application_Deactivated"/>
    </Application.ApplicationLifetimeObjects>

</Application>
```

Listing 6: App.xaml - "Hello World"

Important parts of the code-behind of the App class are explained in the course of this paper and are therefore not listed here (cf chapter 3.4.1 Frame Rate Counters, chapter 3.6 The Application Life-Cycle).

The Page Class

The central element of every program is the Page class. Listing 7 shows the XAML-Code of the `MainPage` class. After the opening tag the `x:Class` element indicates the used namespace (`HelloWorld`) and the class name (`MainPage`) which is also used in the code-behind file (*MainPage.xaml.cs*). Then the following XML namespace declarations are the same as in the *App.xaml.* The `d` and the `mc` namespace are required for XAML design programs Expression Blend and the designer in Visual Studio. Additionally there are some `StaticResource` stettings (FontFamily, FontSize and Foreground) for the page [cf. Petz10 p.14].

```
<phone:PhoneApplicationPage
    x:Class="HelloWorld.MainPage"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:phone="clr-namespace:Microsoft.Phone.Controls;assembly=Microsoft.Phone"
    xmlns:shell="clr-namespace:Microsoft.Phone.Shell;assembly=Microsoft.Phone"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    mc:Ignorable="d" d:DesignWidth="480" d:DesignHeight="768"
    FontFamily="{StaticResource PhoneFontFamilyNormal}"
    FontSize="{StaticResource PhoneFontSizeNormal}"
    Foreground="{StaticResource PhoneForegroundBrush}"
    SupportedOrientations="Portrait" Orientation="Portrait"
    shell:SystemTray.IsVisible="True">
    ...
</phone:PhoneApplicationPage>
```

Listing 7: MainPage.xaml (Excerpt) - "Hello World" Application

The `SupportedOrientation` and `Orientation` attribute will be described in chapter 4.1 (Orientation) and the `shell:SystemTray.IsVisible` attribute is for the visibility of the system tray on a page (Figure 14).

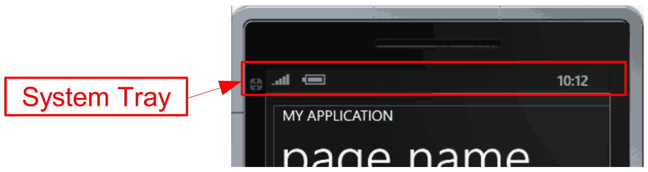


Figure 14: System Tray

Listing 8 shows the *MainPage.xaml.cs* file. By default, the file contains some namespaces and a constructor in the `MainPage` class. In C# namespaces are included with the `using` directive. The namespaces that begin with `System.Windows` are for Silverlight classes and the `Microsoft.Phone.Controls` namespace includes the class `PhoneApplicationPage` [cf. Petz10 p.13].

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Net;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Animation;
using System.Windows.Shapes;
using Microsoft.Phone.Controls;

namespace HelloWorld
{
    public partial class MainPage : PhoneApplicationPage
    {
        // Constructor
        public MainPage()
        {
            InitializeComponent();
        }
    }
}
```

Listing 8: MainPage.xaml.cs - "Hello Word" Application

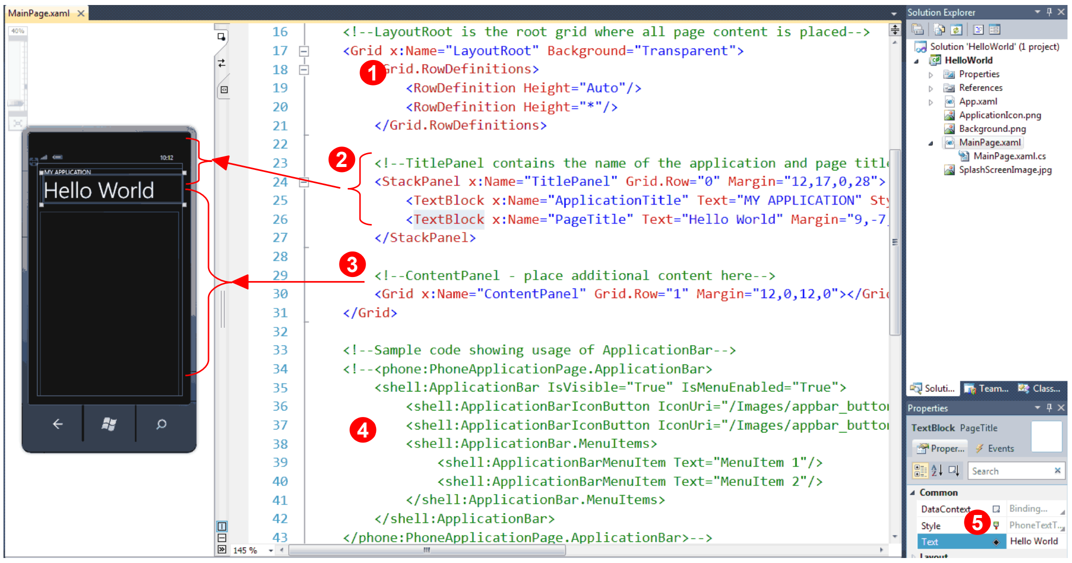
During the build process of the application the program generates a file named *MainPage.g.i.cs*, which is located in the `/obj/Debug` directory. This file is for internal use of the compiler [cf Petz10 p.14]. The constructor of the `MainPage` class in the *MainPage.xaml.cs* (Listing 8) has a method named `InitializeComponent()`. If you place the cursor on the name of the method and press the F12 key a new window with the file *MainPage.g.i.cs* will be displayed on the view and you will see the definition of the method (Listing 9). With the `LoadComponent` method the corresponding XAML file will be read to get information about the application (e.g.: the application or page title).

```
public void InitializeComponent() {
       if (_contentLoaded) {
           return;
       }
       _contentLoaded = true;
       System.Windows.Application.LoadComponent(this, new System.Uri
         ("/HelloWorld;component/MainPage.xaml", System.UriKind.Relative));
       this.LayoutRoot =
         ((System.Windows.Controls.Grid)(this.FindName("LayoutRoot")));
       this.TitlePanel =
         ((System.Windows.Controls.StackPanel)(this.FindName("TitlePanel")));
       this.ApplicationTitle =
         ((System.Windows.Controls.TextBlock)(this.FindName("ApplicationTitle")));
       this.PageTitle =
         ((System.Windows.Controls.TextBlock)(this.FindName("PageTitle")));
       this.ContentPanel =
         ((System.Windows.Controls.Grid)(this.FindName("ContentPanel")));
        }
```

Listing 9: MainPage.g.i.cs - InitializeComponent Method

This file can be ignored by the developer. During the development process it is possible that this file pops up due to an exception. Then the developer should not fix the problem in this file because the real problem is probably in the corresponding XAML file [cf Petz10 p.14].

The body in the *MainPage.xaml* also contains by default some nested elements. Figure 15 shows the “Hello World” project. By default a page contains a `Grid` named `LayoutRoot` as the basic layout container with two rows (#1, Figure 15). Then a `StackPanel` with two `TextBlock` elements are placed in the first row (#2, Figure 15) and another `Grid` which is named `ContentPanel` (#3, Figure 15) in the second row. Also an `ApplicationBar` as a comment is created (#4, Figure 15). For the “Hello World” project the `TextBlock` named `PageTitle` was selected on the Page Designer. Then the element is highlighted in the Page Designer and in the XAML-Editor. After that the value of the `Text` attribute was changed from “page name” to “Hello World” in the Properties Window (#5, Figure 15).



```
<!--LayoutRoot is the root grid where all page content is placed-->
<Grid x:Name="LayoutRoot" Background="Transparent">
    Grid.RowDefinitions>
        <RowDefinition Height="Auto"/>
        <RowDefinition Height="*"/>
    </Grid.RowDefinitions>

    <!--TitlePanel contains the name of the application and page titl
    <StackPanel x:Name="TitlePanel" Grid.Row="0" Margin="12,17,0,28">
        <TextBlock x:Name="ApplicationTitle" Text="MY APPLICATION" Sty
        <TextBlock x:Name="PageTitle" Text="Hello World" Margin="9,-7
    </StackPanel>

    <!--ContentPanel - place additional content here-->
    <Grid x:Name="ContentPanel" Grid.Row="1" Margin="12,0,12,0"></Gri
</Grid>

<!--Sample code showing usage of ApplicationBar-->
<!--<phone:PhoneApplicationPage.ApplicationBar>
    <shell:ApplicationBar IsVisible="True" IsMenuEnabled="True">
        <shell:ApplicationBarIconButton IconUri="/Images/appbar_butto
        <shell:ApplicationBarIconButton IconUri="/Images/appbar_butto
        <shell:ApplicationBar.MenuItems>
            <shell:ApplicationBarMenuItem Text="MenuItem 1"/>
            <shell:ApplicationBarMenuItem Text="MenuItem 2"/>
        </shell:ApplicationBar.MenuItems>
    </shell:ApplicationBar>
</phone:PhoneApplicationPage.ApplicationBar>-->
```

Figure 15: Default XAML Elements - "Hello World" Application

3.4 The Emulator

Now, the „Hello World“ application is complete and ready for debug and run. In the standard toolbar there is a bar for debugging (Figure 16).

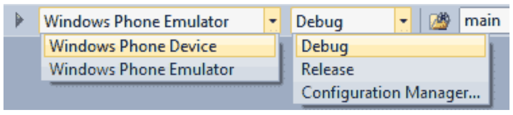


Figure 16: Debugging Bar - Visual Studio

First, the deployment target can be chosen, either a *Windows Phone Device* or the *Windows Phone Emulator*. The deployment on real device is only possible after a registration of a developer phone in the App Hub. Secondly, the *Debug* or *Release* modus can be chosen or the *Configuration Manager* dialog can be opened. For all projects in this work the *Windows Phone Emulator* and the *Debug* modus is selected. With the *Play* button or with the F5 key, Visual Studio builds the project, starts the emulator and launches the application. In Figure 17 the “Hello World” application is deployed in the emulator.

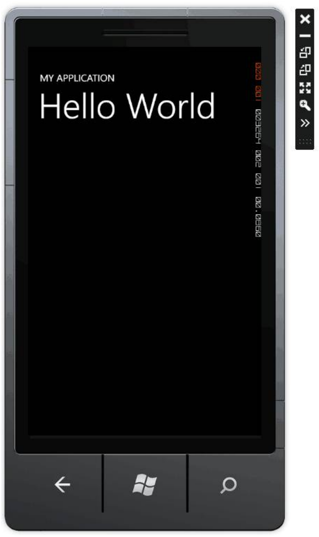


Figure 17: Emulator - "Hello World" Application

The emulator is a virtual machine of the phone in which either the Windows Phone OS 7.0 or the Windows Phone OS 7.1 runs [Wild12].

3.4.1 Frame Rate Counters

When you run an application in the emulator, you can see some numbers at the right top corner on the screen. These numbers are frame rate counters and can be used to monitor the performance of an application [MSDN08].

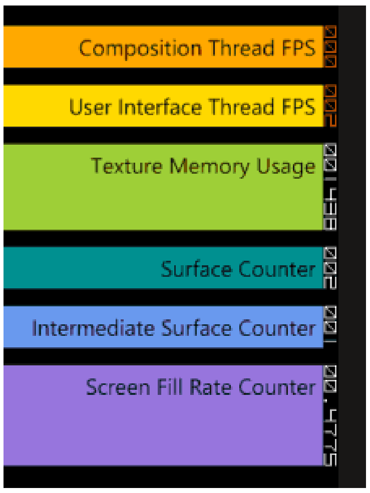


Figure 18: Frame Rate Counters [MSDN08]

The following description is directly taken from the MSDN library [MSDN08]:

Frame rate counter	Description
Composition (Render) Thread	The rate at which the screen is updated.

Frame Rate (FPS)	
User Interface Thread Frame Rate (FPS)	The rate at which the UI thread is running.
Texture Memory Usage	The video memory and system memory copies of textures being used in the application.
Texture Memory Usage	The video memory and system memory copies of textures being used in the application.
Surface Counter	The number of explicit surfaces being passed to the GPU for processing.
Intermediate Surface Counter	The number of implicit surfaces generated as a result of cached surfaces.
Screen Fill Rate Counter	The number of pixels being painted per frame in terms of screens. A value of 1 represents 480 x 800 pixels.

The frame rate counters can also be disabled. The code for the frame rate counters can be found in the *App.xaml.cs* (Listing 10). To disable the frame rate counters the `EnableFrameRateCounter` property must be set to `false` or the code line can simply be commented out.

```
...
public partial class App : Application
    {
        ...
        public App()
        {
            ...
            // Show graphics profiling information while debugging.
            if (System.Diagnostics.Debugger.IsAttached)
            {
                // Display the current frame rate counters.
                Application.Current.Host.Settings.EnableFrameRateCounter = true;
                ...
            }
        ...
    }
```

Listing 10: Frame Rate Counters - App.xaml.cs

3.5 The XAP-File

The XAP-file is the application package. Once Visual Studio has built the project, the XAP-file can be found in the /Bin/Debug directory (Figure 19). It is a compressed zip archive with an .XAP extension and contains all files to run the application [cf. Petz10 p.20]. This file is also called the XAP-archive.

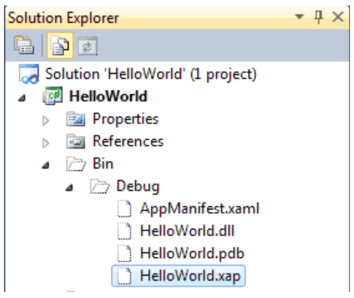
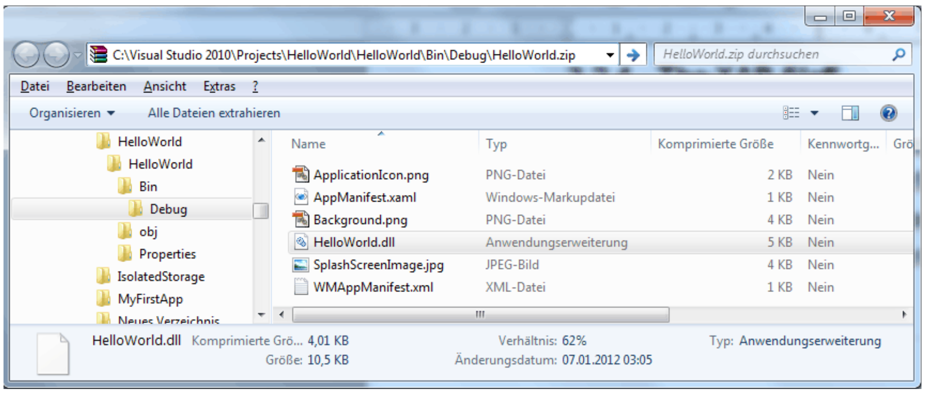


Figure 19: XAP-File - "Hello World" application

The XAP-archive is the file which is hosted on the emulator or on a device. This file must also be submitted to the App Hub to publish the application in the marketplace.

If you navigate with the Windows Explorer to the `/Bin/Debug` directory of the Visual Studio project and rename the *.xap* extension with *.zip* you can see the files inside of the archive (Figure 20). The XAP-archive contains the compiled DLL file of the “Hello World” project (*HelloWorld.dll*), three images which were already described in chapter 3.3.5 and the *WMAppManifest.xml* (cf. Chapter 3.3.2 Properties Folder).


C:\Visual Studio 2010\Projects\HelloWorld\HelloWorld\Bin\Debug\HelloWorld.zip

Name	Typ	Komprimierte Größe	Kennwortg...
ApplicationIcon.png	PNG-Datei	2 KB	Nein
AppManifest.xaml	Windows-Markupdatei	1 KB	Nein
Background.png	PNG-Datei	4 KB	Nein
HelloWorld.dll	Anwendungserweiterung	5 KB	Nein
SplashScreenImage.jpg	JPEG-Bild	4 KB	Nein
WMAppManifest.xml	XML-Datei	1 KB	Nein

HelloWorld.dll Komprimierte Grö... 4,01 KB Größe: 10,5 KB Verhältnis: 62% Änderungsdatum: 07.01.2012 03:05 Typ: Anwendungserweiterung

Figure 20: Explorer - "HelloWorld.zip"

The application package also contains the *AppManifest.xaml*. As in chapter 3.3.2 briefly mentioned, the file is filled with contents in the build process and the code is now shown in Listing 11.

```
<Deployment xmlns="http://schemas.microsoft.com/client/2007/deployment"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" EntryPointAssembly="Hello-
World"
EntryPointType="HelloWorld.App" RuntimeVersion="4.7.50308.0">
  <Deployment.Parts>
    <AssemblyPart x:Name="HelloWorld" Source="HelloWorld.dll" />
  </Deployment.Parts>
</Deployment>
```
Listing 11: AppManifest.xaml - "Hello World" Application

The file contains the deployment details to run the application. The `Deployment` node describes the application and contains a child node (`AssemblyPart`). In the `Deployment` node the `EntryPointAssembly` (which defines the main assembly in the child node), the `EntryPointType` and the `RuntimeVersion` of Silverlight is defined [Piet08]. The entry point `HelloWorld.App` will also be found in the beginning of the *App.xaml* file (Listing 12).

```
<Application
    x:Class="HelloWorld.App"
...
</Application>
```

Listing 12: Entry Point – App.xaml (Excerpt)

3.6 The Application Life-Cycle

The OS of the Windows Phone has been designed in such a way that only one application can be actively running in the foreground at a given time. Other applications will be paused or closed in order to save resources. For example an incoming phone call can interrupt the execution of an application or the user navigates to another application. Therefore, it is important for the developers to consider and understand the application life cycle to ensure a consistent user experience. Often it is necessary that application settings and data are stored, if an application is deactivated or terminated [cf. Getz11 p.377]. A Windows Phone has four states:

- Launched
- Activated
- Deactivated
- Closed

By default, each *App.xaml.cs* file contains methods for the application states as shown in Listing 13.

```
    public partial class App : Application
    {
...
        private void Application_Launching(object sender, LaunchingEventArgs e)
        {
        }

        private void Application_Activated(object sender, ActivatedEventArgs e)
        {
        }
        private void Application_Deactivated(object sender, DeactivatedEventArgs e)
        {
        }

        private void Application_Closing(object sender, ClosingEventArgs e)
        {
        }
...
    }
```
Listing 13: App.xaml.cs - Methods Application Life-Cycle

Launched & Closed

The launched and the closed states are very simple. When the application is started, first the constructor is executed and then the method `Application_Launching` is called. Subsequently, the application runs. If you then use the back button of the Windows Phone to go one step back, the application will be closed (#1, Figure 21).

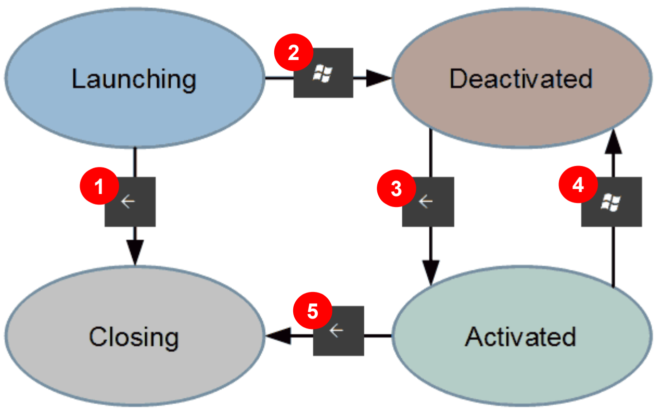


Figure 21: Application Life-Cycle

Deactivated and Activated

When the application is running and the user navigates to the Start screen of the phone (#2, Figure 21), the application will be deactivated. Then, if the back button is pressed to navigate one step back (#3, Figure 21) the application is activated. When the user clicks the start button again to return to the start page, the application is deactivated (#4, Figure 21) and when the back button in the activated state is pressed (#5, Figure 21) the application will be terminated (because the navigation sequence has returned to the beginning). If an application is deactivated, the application is still in the memory of the phone and information about the application is stored (e.g. specific user settings, input data from the user or the last activated page), so when the application is reactivated the record about the application is loaded and the user has the impression that the application never paused. But the OS can terminate a deactivated application, to release resources. Therefore, a deactivated application can be dormant (still in the memory), or tombstoned (the application is terminated). So the tombstoned state is not predictable for the developer. Of course, when the application is tombstoned the state data is also stored, but you do not know exactly what (e.g. the input data of the user is lost).

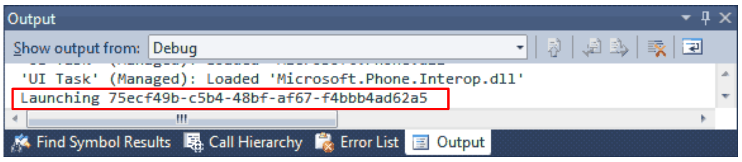
In the following example a new project named *ApplicationLifeCycle* was created in order to demonstrate the application life cycle again for a better understanding. Also the dormant and tombstoned state will be discussed in a greater detail and a method to handle the tombstoned event will be presented. The example is based on a example in the book “Entwickeln für Windows Phone 7.5” of Patrick Getzmann [cf. Getz11 377-384].

On the top of the *App.xaml.cs* file a `System.Diagnostics` namespace is included to get an output in the *Debug Console* of Visual Studio. Then above the constructor a `public static string` named `id` is implemented to store a new Global Unique Identity (GUID). Each method of the application states contains a command that will display the state and die GUID in the *Debug Console (Output Window)* of Visual Studio. Under the menu item *Debug → Windows → Output* the output window can be displayed in Visual Studio.

```
...
using System.Diagnostics;

namespace ApplicationLifeCycle
{
    public partial class App : Application
    {
        public static string id = Guid.NewGuid().ToString();
        ...
        /// Constructor for the Application object.
        public App()
        {
           ...
        }

        private void Application_Launching(object sender, LaunchingEventArgs e)
        {
            Debug.WriteLine("Launching " + App.id);
        }

        private void Application_Activated(object sender, ActivatedEventArgs e)
        {
            Debug.WriteLine("Activated " + App.id);
            if (!e.IsApplicationInstancePreserved)
            {
                MessageBox.Show("The Application was tombstoned!");
            }
        }

        private void Application_Deactivated(object sender, DeactivatedEventArgs e)
        {
            Debug.WriteLine("Deactivated " + App.id);
        }

        private void Application_Closing(object sender, ClosingEventArgs e)
        {
            Debug.WriteLine("Closing " + App.id);
        }
        ...
    }
}
```

Listing 14: Application Life-Cycle Example

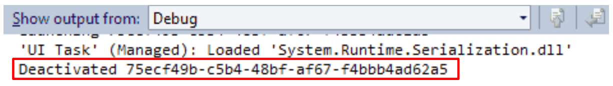
When the application is started the output window shows “`Launching` “ and the `App.id` (Figure 22).



```
'UI Task' (Managed): Loaded 'Microsoft.Phone.Interop.dll'
Launching 75ecf49b-c5b4-48bf-af67-f4bbb4ad62a5
```

Figure 22: Launching

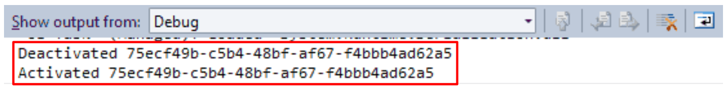
If you press the Start button on the emulator, the application will be deactivated (dormant by default) as shown in Figure 23.



```
'UI Task' (Managed): Loaded 'System.Runtime.Serialization.dll'
Deactivated 75ecf49b-c5b4-48bf-af67-f4bbb4ad62a5
```

Figure 23: Deactivated (Dormant)

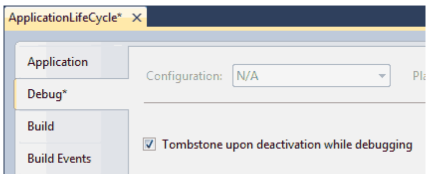
When the back button is pressed, the application will get the state activated and with another click on the back button the application is closed (Figure 24). In each state the GUID in the output windows was the same which means that the instance of the application is still the same.



```
Deactivated 75ecf49b-c5b4-48bf-af67-f4bbb4ad62a5
Activated 75ecf49b-c5b4-48bf-af67-f4bbb4ad62a5
```

Figure 24: Activated and Closing

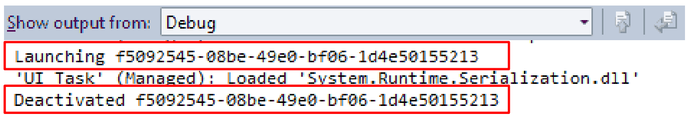
In Visual Studio it is possible to simulate the tombstoned state (the application is terminated by the OS and the state data is saved). Therefore, under the menu item *Project → ApplicationLifeCycle Properties...* the project properties must be opened. In the *Debug* tab *“Tombstone upon deactivation while debugging”* have to be activated (Figure 25).



☑ Tombstone upon deactivation while debugging

Figure 25: ApplicationLifeCycle Properties

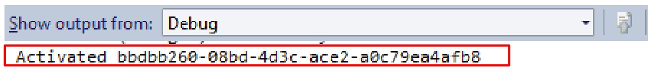
Then the application can be started again and when the initialization is completed and after pressing the Start button the application is set to the deactivated status (Figure 26).



```
Show output from: Debug
Launching f5092545-08be-49e0-bf06-1d4e50155213
'UI Task' (Managed): Loaded 'System.Runtime.Serialization.dll'
Deactivated f5092545-08be-49e0-bf06-1d4e50155213
```

Figure 26: Deactivated (Tombstoned)

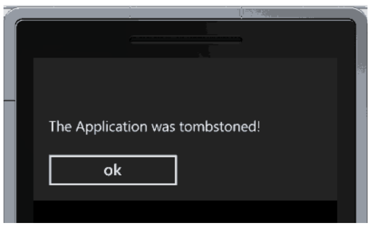
When the application is reactivated via the back button the output window looks like as Figure 27. Note that there is another GUID than for the launching and deactivated output. This means that a new instance of the application has started and the application was tombstoned.



```
Show output from: Debug
Activated bbdbb260-08bd-4d3c-ace2-a0c79ea4afb8
```

Figure 27: Output Window - Activated

In Listing 14 within the `Application_Activated` method the `IsApplicationInstancePreserved` property of events args is implemented. If this property is true, the application was dormant and if the value is false the application was tombstoned. Now, when the program is activated as shown in Figure 27 the emulator should show a message box (Figure 28).



The Application was tombstoned!

Figure 28: MessageBox Activated State

The tombstoned status can be handled on the page basis of an application. Each page in the application will inherit from the `PhoneApplicationPage` class, which will inherit from the `Page` class. This class has a method named `OnNavigatedTo`, which will be called when a page becomes the active page in a frame. Another method is the `OnNavigatedFrom`, which will be called when the page is no longer the active page in a frame [MSDN09]. To use this method in a page of an application, the base method must be overridden.

The `PhoneApplicationPage` class has a property named `State`. This property is a dictionary where objects with a string key (key/value pair) can be saved and loaded.

Listing 15 shows the implementation of the `OnNavigatedTo` and `OnNavigated-From` method and the use of the `State` dictionary. For demonstration purposes the `id` string which was created in the example above (Listing 14) will be saved. In the `OnNavigatedFrom` method the `id` is saved in the `State` dictionary with the key `id`. When the page is loaded, the `OnNavigatedTo` method is invoked. If an `id` is saved in the dictionary and the value is the same as the current `App.id`, the application was in a dormant state, otherwise the application was tomb-stoned. When there is no object in the dictionary, which has the key `id`, the ap-plication was launched.

```
protected override void OnNavigatedFrom(System.Windows.Navigation.NavigationEventArgs e)
{
    base.OnNavigatedFrom(e);
    this.State.Remove("id");
    this.State.Add("id", App.id);
}

protected override void OnNavigatedTo(System.Windows.Navigation.NavigationEventArgs e)
{
    base.OnNavigatedTo(e);

    if (this.State.ContainsKey("id"))
    {
        if (this.State["id"].Equals(App.id))
        {
         MessageBox.Show("The Application was dormant" + this.State["id"].ToString());
        }
        else
       {
         MessageBox.Show("The Application was tombstoned" + this.State["id"].ToString());
       }
    }
    else
    {
        MessageBox.Show("The Application is launched" + App.id.ToString());
   }
}
```
Listing 15: State Dictionary

4 Mobile Applications with Silverlight

This chapter is devoted to the development of Silverlight applications for Windows Phone with Visual Studio. It covers topics such as orientation, building a user interface, navigation through pages, stores data and deals with push notifications.

4.1 Orientation

Windows Phone supports three orientations: namely portrait, landscape left and landscape right. The orientation for a page is defined in the XAML file. The value of the `SupportedOrientations` attribute in a XAML file (Listing 16) contains the supported format of a page. The possible values are: `Portrait`, `Landscape` and `PortraitOrLandscape`. So you can restrict your application to a specific orientation or you can allow it to change the orientation when the phone is turned. In the `Orientation` attribute you can define which orientation the page should have when it is loaded.

```
<phone:PhoneApplicationPage
    ...
    SupportedOrientations="PortraitOrLandscape" Orientation="Portrait"
    OrientationChanged="PhoneApplicationPage_OrientationChanged"
    ...
</phone:PhoneApplicationPage>
```

Listing 16: Page Orientation in XAML

With the implementation of the `OrientationChanged` event in Listing 16, which is fired when the physical orientation of the device has changed, there is the possibility to create an event handler in the code-behind to manipulate elements of the user interface. Listing 17 shows a method which will be called when the `OrientationChanged` event has fired. In this method a `TextBlock` in the XAML file that is named `PageTitle` will display another `Text`, if the orientation of the device has changed.

```
public partial class MainPage : PhoneApplicationPage
    {
        ...
        private void PhoneApplicationPage_OrientationChanged(
            object sender, OrientationChangedEventArgs e)
        {
            if (e.Orientation == PageOrientation.LandscapeRight)
            { this.PageTitle.Text = "Landscape right"; }
            else if (e.Orientation == PageOrientation.LandscapeLeft)
            { this.PageTitle.Text = "Landscape left"; }
            else
            { this.PageTitle.Text = "Portrait"; }
        }
    }
```

Listing 17: Page Orientation in Code-Behind

Figure 29 shows the deployment of the example.

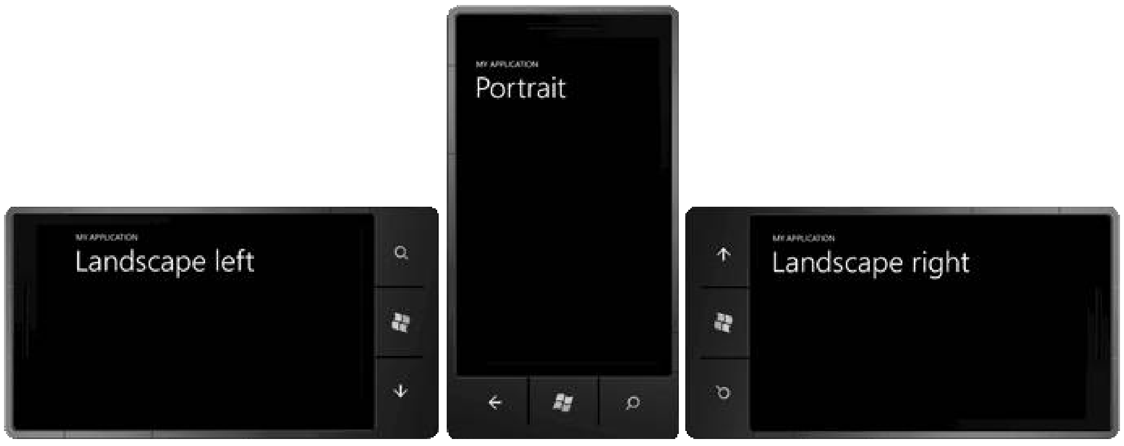


Figure 29: Page Orientation - Emulator

4.2 Layout

As already mentioned in chapter 3.3.6 (The Page Class), when a new Windows Phone Application application is created by Visual Studio, the *MainPage.xaml* contains by default two layout containers, a grid and a stack panel. These are the two basic elements to design a layout of a page. The containers themselves do not have any user interface, but they are determined to arrange elements on the screen. Each container can include one or more child elements (layout containers and visible UI elements).

4.2.1 Grid

The Grid is a good choice for most routine layouts and is very important. It is generally arranged in rows and columns and is very similar to a HTML table. Listing 18 shows a simple grid that was added to the *MainPage.xaml* of a new project. With the attribute `ShowGridLines` the rows and columns are visible in the emulator. With the properties `Grid.RowDefinitions` and `Grid.ColumnDefinitions` new columns and rows can be created. The `Height` attribute sets the row height and the `Width` attribute the column width. The star sizing value (3*) means a weighted proportion of the available space while an absolute value (e.g. 123) is expressing pixels.

```
<Grid x:Name="GridContent" ShowGridLines="True">
    <Grid.RowDefinitions>
        <RowDefinition Height="3*" />
        <RowDefinition Height="3*" />
        <RowDefinition Height="3*"/>
    </Grid.RowDefinitions>

    <Grid.ColumnDefinitions>
        <ColumnDefinition Width="5*" />
        <ColumnDefinition Width="5*" />
    </Grid.ColumnDefinitions>

    <TextBlock Grid.Column="0" Grid.Row="0" Text="Row 0 / Column 0"
               VerticalAlignment="Center" HorizontalAlignment="Center"/>
    <TextBlock Grid.ColumnSpan="2" Grid.Row="1" Text="Row 1 / Column 2"
               VerticalAlignment="Center" HorizontalAlignment="Center"/>
    <TextBlock Grid.Column="1" Grid.Row="2" Text="Row 1 / Column 2"
               VerticalAlignment="Center" HorizontalAlignment="Center"/>
</Grid>
```

Listing 18: Grid-Definition in XAML

Additionally, three TextBlock elements have been created in this XAML-Code and with the Grid.Column and the Grid.Row attribute you can define the location for a control element in the grid. Figure 30 shows this application in the emulator.

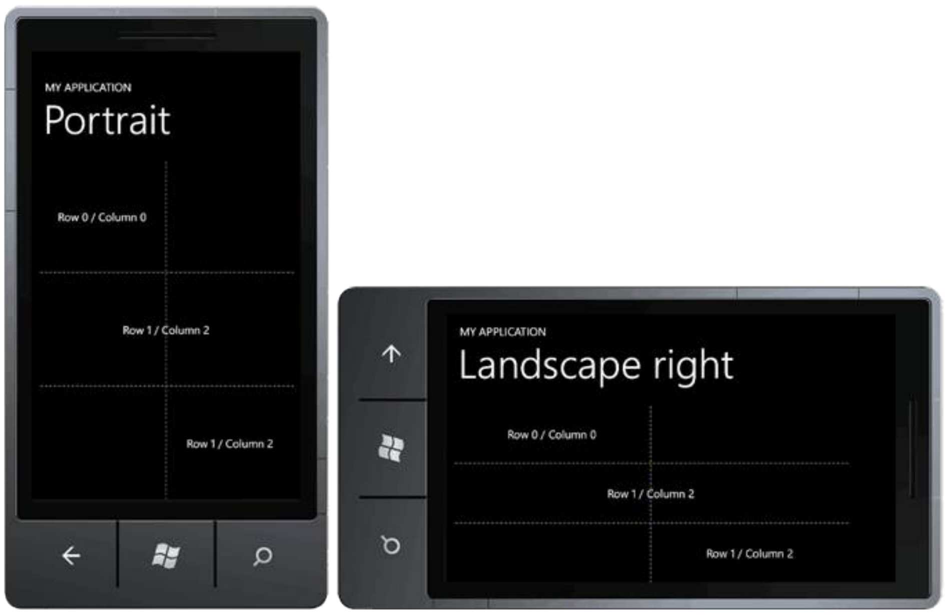


Figure 30: Grid Row and Columns

4.2.2 Stack Panel

The Stack Panel is also a layout container for control elements of the user interface. In contrast to the grid, the children of the `StackPanel` do not overlap. By default, the elements are arranged from top to bottom. With the `Orientation` attribute the vertical arrangement of the elements can be changed (Listing 19).

```
    <Grid x:Name="GridContent" ShowGridLines="True">
        <Grid.RowDefinitions>
            <RowDefinition Height="5*" />
            <RowDefinition Height="5*" />
        </Grid.RowDefinitions>

        <StackPanel >
            <Button Content="Button1" Height="71" Width="160" />
            <Button Content="Button2" Height="71" Width="160" />
            <Button Content="Button3" Height="71" Width="160" />
        </StackPanel>

        <StackPanel Grid.Row="1" Orientation="Horizontal">
            <Button Content="Button" Height="71" Width="160" />
            <Button Content="Button" Height="71" Width="160" />
            <Button Content="Button" Height="71" Width="160" />
        </StackPanel>
    </Grid>
```

Listing 19: StackPanel-Definition in XAML

Figure 31 shows the `StackPanel` in the emulator.

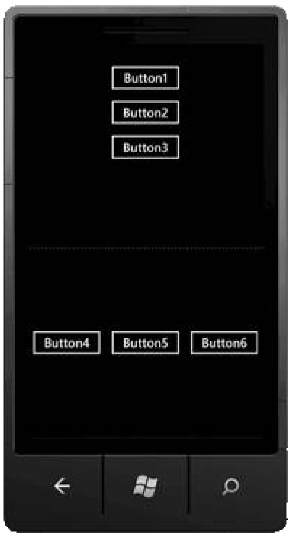


Figure 31: The Stack Panel

4.2.3 Pivot and Panorama

When you create a new project with Visual Studio, you have several templates to choose from. There is also a *Windows Phone Panorama Application* and a *Windows Phone Pivot application* available (cf. chapter 3.2). Both concepts are very similar. The space for the content is wider than the actual width of the device [cf. Petz10 p.712]. Listing 20 shows the XAML-Code for a panorama page. The `Microsoft.Phone.Controls` namespace is required, because this includes the pivot and panorama class. By default, you can find in the `Grid` which is named `LayoutRoot` two panorama items. They include a `ListBox` with a `DataTemplate` which contains a `StackPanel` and two `TextBlock` elements. In the `ItemSource` attribute of the `ListBox` and in the `Text` attribute of the two `TextBlock` elements you will find curly braces with a `Binding` statement with a property. This is the syntax for data binding which will be covered in chapter 4.5 (Data Binding).

```
<phone:PhoneApplicationPage
    ...
    xmlns:controls="clr-namespace:Microsoft.Phone.Controls;assembly=Microsoft.Phone.Controls"
    ...
    <!--LayoutRoot is the root grid where all page content is placed-->
    <Grid x:Name="LayoutRoot" Background="Transparent">

        <!--Panorama control-->
        <controls:Panorama Title="my application">
            <controls:Panorama.Background>
                <ImageBrush ImageSource="PanoramaBackground.png"/>
            </controls:Panorama.Background>

            <!--Panorama item one-->
            <controls:PanoramaItem Header="first item">
                <!--Double line list with text wrapping-->
                <ListBox Margin="0,0,-12,0" ItemsSource="{Binding Items}">
                    <ListBox.ItemTemplate>
                        <DataTemplate>
                            <StackPanel Margin="0,0,0,17" Width="432" Height="78">
                                <TextBlock Text="{Binding LineOne}" TextWrapping="Wrap"
                                           Style="{StaticResource PhoneTextExtraLargeStyle}"/>
                                <TextBlock Text="{Binding LineTwo}" TextWrapping="Wrap"
                                           Margin="12,-6,12,0"
                                           Style="{StaticResource PhoneTextSubtleStyle}"/>
                            </StackPanel>
                        </DataTemplate>
                    </ListBox.ItemTemplate>
                </ListBox>
            </controls:PanoramaItem>

            <!--Panorama item two-->
            <controls:PanoramaItem Header="second item">
            ...
            </controls:PanoramaItem>
        </controls:Panorama>
    </Grid>
</phone:PhoneApplicationPage>
```

Listing 20: Panorama Layout - XAML

A pivot page contains instead of the `PanoramaItem` a `PivotItem` as shown in Listing 21. The code is very similar to the panorama page.

```
<phone:PhoneApplicationPage
   ...
    xmlns:controls="clr-namespace:Microsoft.Phone.Controls;assembly=Microsoft.Phone.Controls"
    ...

    <Grid x:Name="LayoutRoot" Background="Transparent">
        <controls:Pivot Title="MY APPLICATION">

            <!--Pivot item one-->
            <controls:PivotItem Header="first">
               <ListBox x:Name="FirstListBox" Margin="0,0,-12,0"
                        ItemsSource="{Binding Items}">
                    <ListBox.ItemTemplate>
                        <DataTemplate>
                          <StackPanel Margin="0,0,0,17" Width="432" Height="78">
                              <TextBlock Text="{Binding LineOne}" TextWrapping="Wrap"
                                         Style="{StaticResource PhoneTextExtraLargeStyle}"/>
                              <TextBlock Text="{Binding LineTwo}"
                                         TextWrapping="Wrap" Margin="12,-6,12,0"
                                         Style="{StaticResource PhoneTextSubtleStyle}"/>
                          </StackPanel>
                        </DataTemplate>
                    </ListBox.ItemTemplate>
                </ListBox>
            </controls:PivotItem>

            <!--Pivot item two-->
            <controls:PivotItem Header="second">
            ...
        </controls:Pivot>
    </Grid>
    ...
</phone:PhoneApplicationPage>
```

Listing 21: Pivot Page - XAML

On the left side in Figure 32 a default pivot page is shown. On the right side there is a standard panorama page. Both pages are only one page object. Inside the pivot page the user can navigate to various pivot items with a link in the `Header` (Listing 21 `Header="first"` and `Header="second"`). In the panorama page the user can navigate to the next panorama item with the flick touch gesture.

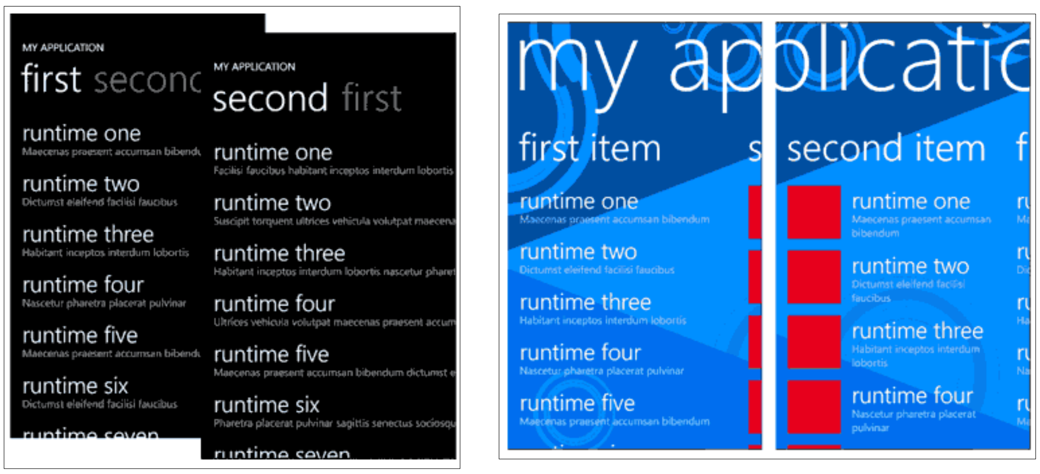


Figure 32: Pivot and Panorama Page

The pivot page can be used to present collections of information sliced to several subsets. The idea is to group similar data [Bosc11]. For example, the search result of Bing is displayed in a pivot page (Figure 33).

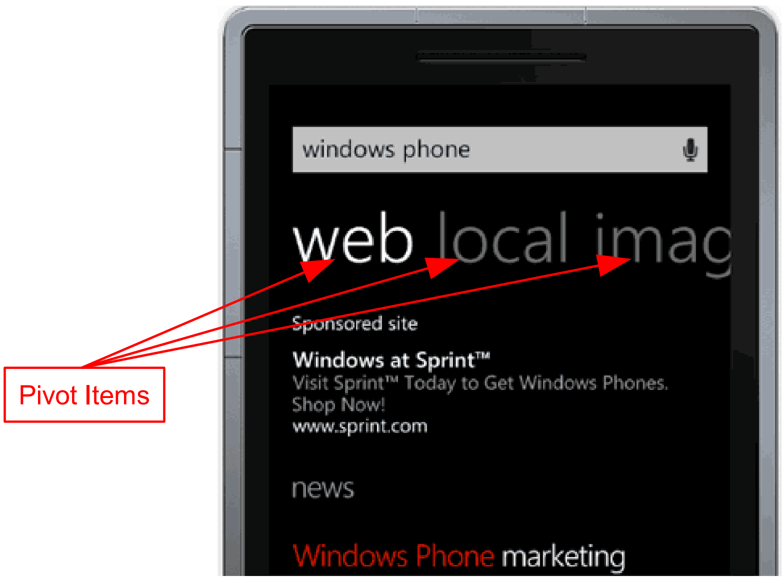


Figure 33: Bing - Search Result

The panorama page is used to slide through a big content. The content is divided to different blocks which are assigned to different horizontal items [Kart12]. Figure 34 show the concept of a panorama page.

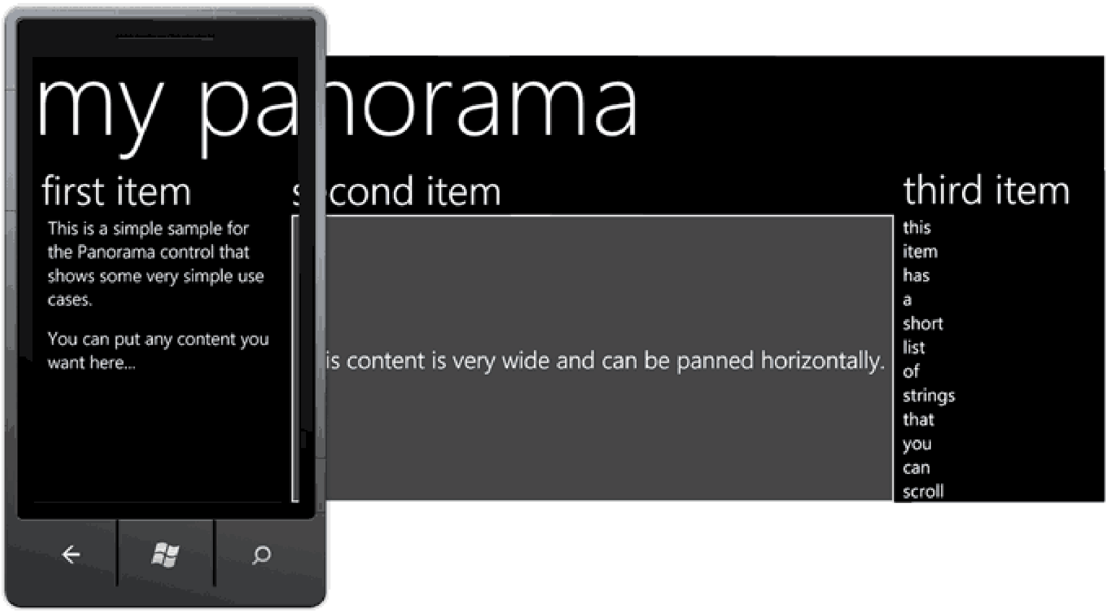

my panorama
first item
This is a simple sample for the Panorama control that shows some very simple use cases.
You can put any content you want here...
second item
is content is very wide and can be panned horizontally.
third item
this item has a short list of strings that you can scroll

Figure 34: Panorama Example [Wilc10]

4.3 Smartphone Controls

The control elements are used for the interaction with the user. On the one hand there are elements that use the touch interface (e.g. `Button`, `CheckBox`, etc.) and on the other hand there are elements (e.g. `TextBox`, `PasswordBox`) that are used for the input of the user with the keyboard. The toolbox in Visual Studio will provide most of the supported user interface elements for a phone application. A comprehensive overview of all supported controls in Silverlight for Windows Phone OS 7.1. will be found in the MSDN library [MSDN10]. In this chapter the basic elements which were also used in the most programming examples in this work will be described.

4.3.1 Button

The `Button` is used to trigger an action after touch. In the following example a message box will be displayed, after the button was pressed.

```
<Button Content="Button" Height="72"  Width="160" Margin="135,249,161,286"
        Name="button1" Click="button1_Click" />
```

Listing 22: Button-Definition in XAML

A click event with an event handler (`button1_Click`) was added to the `Button` tag. For this click event a method in the code-behind file which shows the message box is implemented.

```
private void button1_Click(object sender, RoutedEventArgs e)
        {
            MessageBox.Show("Button was pressed!");
        }
```

Listing 23: Code-Behind - Button Click Event

4.3.2 HyperlinkButton

The `HyperlinkButton` is used to navigate to other pages in an application. The `NavigateUri` property contains the link to the destination page. In the `TargetName` property the target frame can be specified (`_blank` or `_self`).

```
<HyperlinkButton Content="Button" Height="30" Width="200" Name="button"
                 NavigateUri="/Examples/Button.xaml" TargetName="_blank"/>
```

Listing 24: HyperlinkButton-Definition in XAML

4.3.3 TextBlock

The `TextBlock` element is primarily used to display text.

```
<TextBlock Margin="40,375,94,39" Name="textBlock1" TextWrapping="Wrap"
           Text="Lorem ipsum dolor sit amet, consetetur sadipscing elitr"/>
```

Listing 25: TextBlock-Definiton in XAML

The `Text` property contains the text. By default, the content will be shown without a line break. With the `TextWrapping` property an automatic line break can be effected (Figure 35).

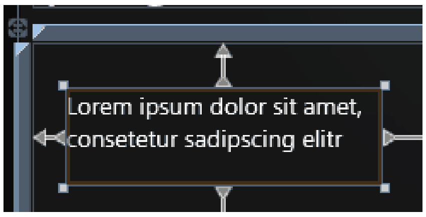


Figure 35: TextBlock - TextWrapping

4.3.4 CheckBox

The `CheckBox` is used to select and clear an option in an application. The controls allow the user to select a combination from a list of options. The important property is `IsChecked`, which can have three states (`true`, `false` or `null`).

```
        <Grid x:Name="ContentPanel" Grid.Row="1" Margin="12,0,12,0">
            <CheckBox Content="CheckBox" Margin="26,18,228,516"
                      Name="checkBox1" IsChecked="{x:Null}"/>
            <Button Content="Button" Margin="6,97,20,425"
                    Name="button1" Click="button1_Click" />
        </Grid>
```

Listing 26: CheckBox-Definition in XAML

At runtime, the state of the `CheckBox` can be queried. The following code-behind shows a method for a click event of the `Button`, which will display the state of the `MessageBox`.

```
private void button1_Click(object sender, RoutedEventArgs e)
        {
            if (this.checkBox1.IsChecked == null)
            {
                MessageBox.Show("CheckBox state is null!");
            }
            else if (this.checkBox1.IsChecked == true)
            {
                MessageBox.Show("CheckBox state is true!");
            }
            else
            {
                MessageBox.Show("CheckBox state is false!");
            }
        }
```

Listing 27: Code-Behind - CheckBox Query

4.3.5 RadioButton

In contrast to `CheckBox`, a `RadioButton` is used when there is a list of options and they are mutually exclusive. In other words, when a non-selected button is selected, a previously selected button will be deactivated. In XAML this is realized by the `GroupName` property that allows you to differentiate multiple groups of mutually exclusive buttons [cf. Petz10 p.273]. The central property is again `IsChecked` with a boolean return type.

```
<RadioButton Content="RadioButton1" Height="72" Margin="50,388,0,147"
             Name="radioButton1" GroupName="Group1" IsChecked="True"/>
<RadioButton Content="RadioButton2" Height="72" Margin="50,359,0,0"
             Name="radioButton2" GroupName="Group1"/>
```

Listing 28: RadioButton-Definition in XAML

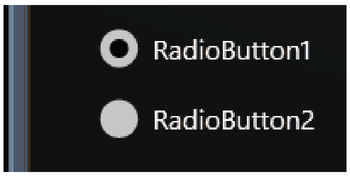


Figure 36: RadioButtons

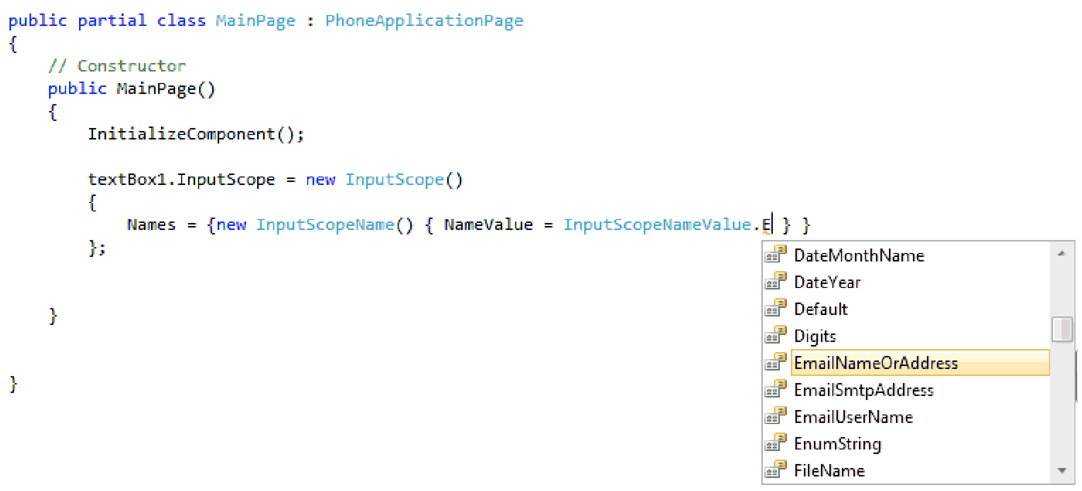
4.3.6 TextBox, PasswordBox and Keyboard Input

The `Textbox` and the `PasswordBox` are the two text entry controls. For the majority of the devices which will not have a physical keyboard, the software input panel (SIP) is used when one of these controls have received focus.

```
            <TextBox Height="72" Margin="47,130,49,405" Name="textBox1"
                     Text="TextBox" Width="361" />
            <PasswordBox Height="72" Margin="48,246,48,289" Name="passwordBox1"
                         Width="361"/>
```

Listing 29: TextBox and PasswordBox-Defintion in XAML

In the `TextBox` it is possible to assign an attribute named `InputScope`. With the `InputScope` you can suggest a specific keyboard for the input field (e.g. for numbers, email address, etc.). This attribute can be defined in the XAML-Code or in the code-behind file. If the definition is made in the code-behind, the developer can make use of the *IntelliSense* of Visual Studio which will provide the possible values for the `InputScope` (Figure 37).



```
public partial class MainPage : PhoneApplicationPage
{
    // Constructor
    public MainPage()
    {
        InitializeComponent();

        textBox1.InputScope = new InputScope()
        {
            Names = {new InputScopeName() { NameValue = InputScopeNameValue.E } }
        };

    }

}
```

Figure 37: InputScope IntelliSense

Figure 38 shows an example where the value “`EmailNameOrAddress`” was chosen for the input scope. The input panel will provide keys (e.g. “@” or “.com”) to enter an email address.

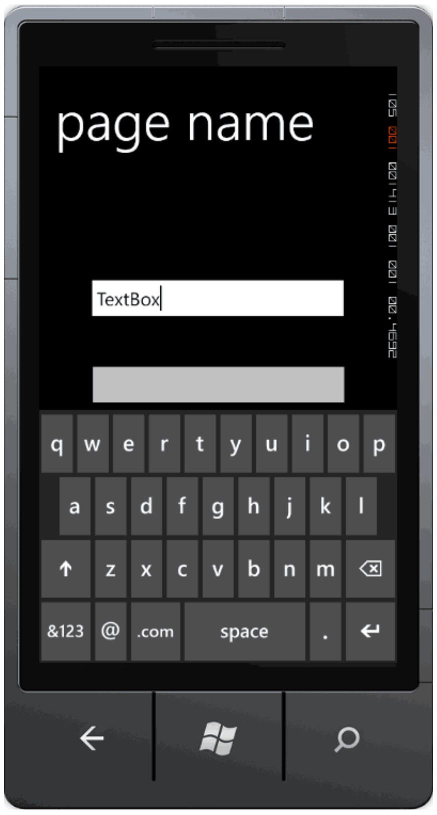


Figure 38: Emulator InputScope

4.3.7 ApplicationBar

The `ApplicationBar` is an alternative to standard controls such as buttons. It can be used instead of creating an own menu. If a new page is added to a project, an `ApplicationBar` in the XAML file is included as a comment. The `ApplicationBar` contains up to four buttons. Additional, menu items can be added. As shown in Figure 39, the `ApplicationBar` is always on the bottom of a page when it is displayed and stays in the same place relative to the phone whenever the phone rotates. When the `SupportedOrientations` property is set to `PortraitOrLandscape`, the images of the `ApplicationBar` turns sideways [cf. Petz10 p.235].

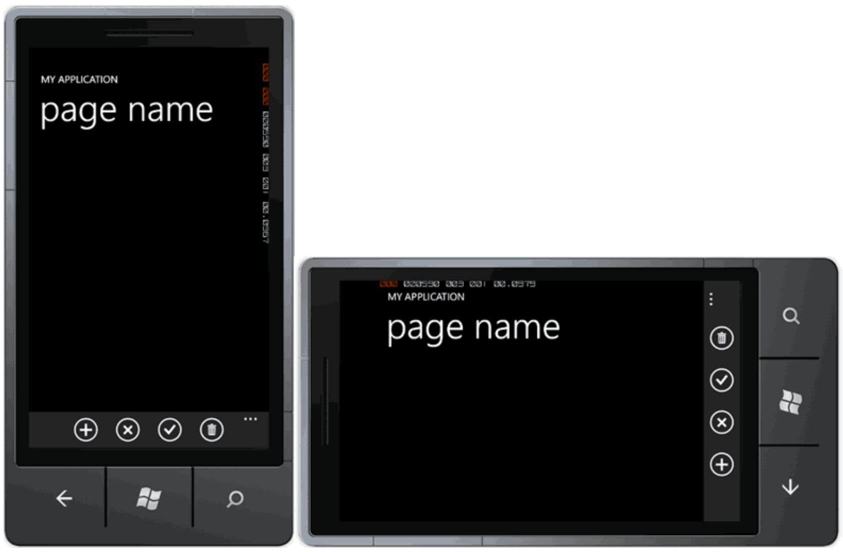
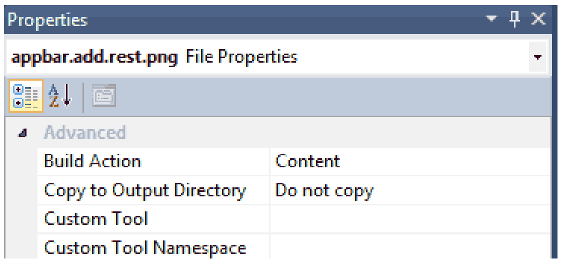


Figure 39: Emulator - ApplicationBar

In the XAML-Code (Listing 30) the `IconUri` and `Text` attribute are required.

```
<phone:PhoneApplicationPage.ApplicationBar>
     <shell:ApplicationBar IsVisible="True" IsMenuEnabled="True">
         <shell:ApplicationBarIconButton IconUri="/Images/appbar.add.rest.png"
                                         Text="Hinzufügen"/>
         <shell:ApplicationBarIconButton IconUri="/Images/appbar.cancel.rest.png"
                                         Text="Abbrechen"/>
         <shell:ApplicationBarIconButton IconUri="/Images/appbar.check.rest.png"
                                         Text="Übernehmen"/>
         <shell:ApplicationBarIconButton IconUri="/Images/appbar.delete.rest.png"
                                         Text="Löschen"/>
         <shell:ApplicationBar.MenuItems>
             <shell:ApplicationBarMenuItem Text="MenuItem 1"/>
         </shell:ApplicationBar.MenuItems>
     </shell:ApplicationBar>
</phone:PhoneApplicationPage.ApplicationBar>
```
Listing 30: ApplicationBar-Definition in XAML

The Windows Phone SDK provides several icons for the application bar. After a standard installation you will find these icons in the following directory: `C:\Program Files\Microsoft SDKs\Windows Phone\v7.1\Icons`. In the folder named `light` you will find black images on a white background and in the `dark` folder white images on a black background. The icons can easily be added via drag and drop from the Windows Explorer into the Solution Explorer of Visual Studio. Make sure that the *Built Action* property of each icon is set to *Content* (Figure 40), because the application bar is not smart enough to find the icons, if the *Built Action* is *Resource* [cf. Petz10 p.233].


Properties

appbar.add.rest.png File Properties

Advanced

Build Action	Content
Copy to Output Directory	Do not copy
Custom Tool	
Custom Tool Namespace	

Figure 40: Properties Window - ApplicationBar

4.4 Navigation

As mentioned in chapter 3.3.6 (Silverlight files) a Windows Phone Application is based on pages similar to a website. If an application consists of more than one page, then it must be possible for the user to navigate between the pages. A simple solution is the `HyplerlinkButton` which was already covered in chapter 4.3.2 (HyperlinkButton). The following solutions are based on a new project which includes three page objects (*MainPage*, *SecondPage* and *ThirdPage*).

4.4.1 Code-Behind Solution

The advantage of navigation in the code-behind is that any XAML element can be used for the navigation. This is made possible by the `NaviagtionService` class. In the following example a Button on *MainPage.xaml* is used to navigate to the *SecondPage.xaml* after the button is pressed. In the `Click` event attribute the name of the event handler `button1_Click` gets defined.

```
        <Grid x:Name="ContentPanel" Grid.Row="1" Margin="12,0,12,0">
            <Button Content="Navigate to Page 2" Height="72" Width="321"
                    Margin="78,78,57,457" Name="button1" Click="button1_Click" />
            ...
        </Grid>
```

Listing 31: XAML Button for Navigation

In the code-behind the `Navigate` method of the `NavigationService` class is called. A new URI object is instantiated and the page name and the URI type (`Relative`, `Absolute` or `RelativeOrAbsolute`) are passed as parameters.

```
        private void button1_Click(object sender, RoutedEventArgs e)
        {
            NavigationService.Navigate(new Uri("/SecondPage.xaml",
                UriKind.Relative));
        }
```

Listing 32: Code-Behind Navigation

4.4.2 Passing Parameters

A simple method to pass data from one page to another page during navigation is to specify a name-value pair in the URI. The following example shows how a string data from a `TextBox` on *MainPage.xaml* is passed to a `TextBlock` on the *SecondPage.xaml*.

```
        <Grid x:Name="ContentPanel" Grid.Row="1" Margin="12,0,12,0">
            ...
            <Button Content="Navigate to Page 2" Height="72" Width="321"
                    Margin="77,262,57,273" Name="button2" Click="button2_Click" />
            <TextBox Height="72" Margin="77,184,54,351" Name="textBox1"
                     Text="TextBox" Width="325" />
         ...
        </Grid>
```

Listing 33: XAML Button and TextBox for Passing Parameter

In the code-behind the `NavigationService` class is used. After the page name a question mark gets added to specify the name-value pair. As name `parameter` is used (which is an arbitrary name) and the string from the `TextBox` is the value.

```
        private void button2_Click(object sender, RoutedEventArgs e)
        {
            NavigationService.Navigate(new Uri("/SecondPage.xaml?parameter="
                + textBox1.Text, UriKind.Relative));
        }
```

Listing 34: Code-Behind for Navigation and Passing Parameter

In the code-behind of the destination page (*SecondPage.xaml.cs*) the `OnNavigatedTo` method is implemented to dispaly the passed parameter in the `TextBlock`.

```
protected override void OnNavigatedTo(System.Windows.Navigation.NavigationEventArgs e)
    {
        base.OnNavigatedTo(e);
        string parameter = String.Empty;
            if (NavigationContext.QueryString.TryGetValue("parameter", out parameter))
            {
                textBlock1.Text = parameter;
            }
    }
```

Listing 35: Code-Behind to Display the Parameter

4.4.3 Sharing Data

Passing data via a name-value pair in the URI works perfectly. A more elegant way to provide data is to store the data in the App class (*App.xaml.cs*), which also corresponds more to the object orientated idea. All pages in a Silverlight application for Windows Phone have access to the App class. This class can be used to store data. For the following example a public property is added in the *App.xaml.cs* above the constructor.

```
public partial class App : Application
    {
        public String Parameter2 { get; set; }
        ...
    }
```

Listing 36: Public String in App.xaml.cs

This property can be used in each page instance of the project. In the *ManPage.xaml.cs* another `TextBox` named `textBox2` and a `Button` named `button3` is added. In the method for the click event of the `Button` the `Application.Current` property returns a reference to the instance of the class that derives from Application and must be casted to an `App`. Then the text, which is contained in `textBox2` will be stored in the property that was defined in the `App` class and the the navigation to the *ThirdPage.xaml* occurs.

```
        private void button3_Click(object sender, RoutedEventArgs e)
        {
            App app = (Application.Current as App);
            app.Parameter2 = this.textBox2.Text;
            NavigationService.Navigate(new Uri("/ThirdPage.xaml",
                UriKind.RelativeOrAbsolute));
        }
```

Listing 37: Code-Behind to Store Data

In the destination page the following code is used within the constructor to read the value of the property.

```
        public ThirdPage()
        {
            InitializeComponent();
            App app = (Application.Current as App);
            textBlock1.Text = app.Parameter2;
        }
```

Listing 38: Code-Behind to Read the Property

4.5 Data Binding

Because of the strict distinction between the UI and the program logic in Silverlight applications, it is often necessary that an UI element reflects the changes on a data object in the code-behind. A connection, or data binding allows a flow between a UI element and a data object or between two UI elements. The examples in this chapter are based on the examples in the book “Entwickeln für Windows Phone 7.5” [cf. Getz11 227-239].

4.5.1 Simple Data Binding

Listing 39 shows a data binding between two UI elements (a `Slider` and a `TextBox`). The `TextBox` shows the value of the `Slider` and if a value (0-10) is typed in the `TextBox`, the `Slider` will be adjusted (if text is entered nothing happens). Therefore, the `Text` property of the `TextBox` contains the binding syntax which means that there is a binding to the value of the `slider1` element. The `Mode` property `TwoWay` determines a data flow in both directions (from the source to the target and vice versa).

```
<Slider Margin="90,222,92,312" Name="slider1" Width="275" />
<TextBox Height="72" Width="275" Margin="90,332,92,203" Name="textBox1"
         Text="{Binding Path=Value, ElementName=slider1, Mode=TwoWay}" />
```

Listing 39: Data Binding of two UI Elements

There are three types of binding:

- **OneTime:** The value from the source to the target is passed only once.
- **OneWay:** This is the default mode. The data can only flow from the source to the target.
- **TwoWay:** The data can flow in both directions, from the source to the target and vice versa.

4.5.2 Change Notification

The `INotifyPropertyChanged` interface is another foundation of data binding. It will be implemented to business objects in the code-behind in order to push changes from a source (e.g. changes to a property of a data object) to a target (e.g. a UI control in XAML). The following example shows the implementation of the interface with a `OneWay` binding.

In the XAML file a `TextBlock`, a `TextBox` and a `Button` were added. The `Text-Block` contains the binding to a data object and shows the property `FirstName` of a `Student` object. In the `TextBox`, the user can enter a new name for the student, which will be confirmed with a button.

```
        <Grid x:Name="ContentPanel" Grid.Row="1" Margin="12,0,12,0">
            <TextBlock Height="39" Margin="52,51,125,546" Name="textBlock2"
                       Text="{Binding FirstName}" />
            <TextBox Height="72" HorizontalAlignment="Left" Margin="52,133,0,431"
                     Name="textBox1" Text="New Firstname" Width="279" />
            <Button Content="Change Firstname" Height="72" Margin="52,211,125,353"
                    Name="button1" Click="button1_Click" />
        </Grid>
```
Listing 40: Change Notification - XAML-Code

The user interface is shown in Figure 41.

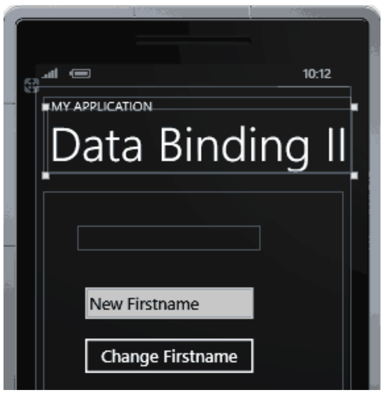


Figure 41: Change Notification UI

In the code-behind file (Listing 41) a new instance of the class `Student` is declared and in the constructor the object is initialized. The `DataContent` property is used to display the properties of the object in the `TextBlock`. The class `Stu-dent` implements the `INotifyPorpertyChanged` interface. For the interface the namespace `System.ComponentModel` is required. Then an event named `Proper-tyChanged` is defined and will be fired if a property is changed. The setter method of the property will call a method named `NotifyPropertyChanged` and the binding element will be updated if the value of the property has changed.

```
...
using System.ComponentModel;

namespace DataBindingII
{
    public partial class MainPage : PhoneApplicationPage
    {
        Student s = new Student();
        // Constructor
        public MainPage()
        {
            InitializeComponent();

            s.FirstName = "Jane";
            s.LastName = "Doe";
            s.StudentNumber = 21;
            ContentPanel.DataContext = s;
        }

        public class Student : INotifyPropertyChanged
        {
            ...
            private string firstName;
            public string FirstName
            {
                get
                {
                    return firstName;
                }
                set
                {
                    firstName = value;
                    NotifyPropertyChanged("FirstName");
                }
            }

            public event PropertyChangedEventHandler PropertyChanged;
            private void NotifyPropertyChanged(string info)
            {
                if (PropertyChanged != null)
                {
                    PropertyChanged(this, new PropertyChangedEventArgs(info));
                }
            }
            ...
        }
        private void button1_Click(object sender, RoutedEventArgs e)
        {
            s.FirstName = textBox1.Text;
        }
    }
}
```

Listing 41: Change Notification - Code-Behind

4.5.3 Data Binding with a Generic List

The following example shows the data binding of a generic list and a `ListBox` element. In the XAML file a `ListBox` with an `DataTemplate` is used. A DataTemplate is used to specify the visualization of your data objects [MSDN11]. For each object in the list, an instance of this template will be created. The `DataTemplate` contains a `StackPanel` (cf. chapter 4.2.2) and three `TextBlock` elements, which contain the binding to the object property.

```
<ListBox Margin="67,100,69,101" Name="lbStudents" >
     <ListBox.ItemTemplate>
        <DataTemplate>
            <StackPanel>
                <TextBlock Text="{Binding FirstName}" />
                <TextBlock Text="{Binding LastName}" />
                <TextBlock Text="{Binding StudentNumber}" Padding="0,0,0,10" />
            </StackPanel>
        </DataTemplate>
    </ListBox.ItemTemplate>
</ListBox>
```

Listing 42: Data Binding - XAML ListBox

The code-behind defines how the generic list will be created and assigned to the `ListBox` element. In contrast to the previous example (Listing 41) the `ItemSource` property must be used to display the list in the `ListBox` because of the `IEnumerable` object type. The `DataContext` property does not generate templates and expects an object type (and not an `IEnumerable` type object) [Soda02].

```
namespace DataBindingIII
{
    public partial class MainPage : PhoneApplicationPage
    {
        List<Student> students = new List<Student>();
        // Constructor
        public MainPage()
        {
            InitializeComponent();

            Student s1 = new Student();
            s1.FirstName = "Jane";
            s1.LastName = "Doe";
            s1.StudentNumber = 1234567;
            students.Add(s1);

            Student s2 = new Student();
            s2.FirstName = "John";
            s2.LastName = "Doe";
            s2.StudentNumber = 7654321;
            students.Add(s2);

            lbStudents.ItemsSource = students;
        }

        public class Student
        {
            public string FirstName { get; set; }
            public string LastName { get; set; }
            public long StudentNumber { get; set; }
        }
    }
}
```

Listing 43: Data Binding - Code-Behind Generic List

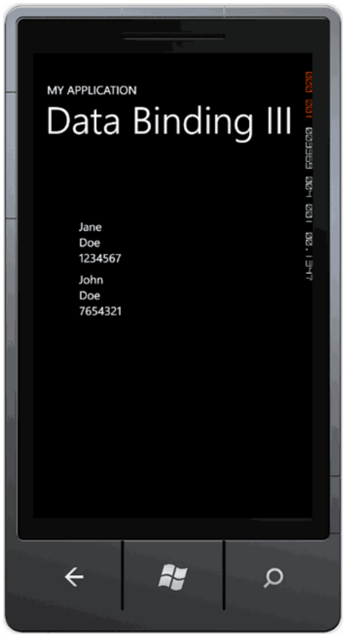


Figure 42: Data Binding with a Generic List

4.6 Data Storage

As a Silverlight application for Windows Phone is running in a sandbox, there is no direct access to the underlying file system of the OS. But the application can use the isolated storage to store the application's data such as settings of the application, files, folders or relational data. Isolated storage is a concept in .NET and is a kind of virtual folder. The physical location of the isolated storage can vary for each OS and the user do not know where the file exactly is stored. An application simply uses the .NET classes to create and access the files [Manj04]. Isolated also means that one application can not see or use the data from another application and vice versa. If two applications need the same data an online storage (cloud service) is required.

The examples in chapter 4.6.1, 4.6.2 and 4.6.4 are based on the isolated storage explanation in the book “Windows Phone 7-Apps” of Christian Bleske [cf. Bles11 215-217].

4.6.1 IsolatedStorageSettings

The following example shows, how the class `IsolatedStorageSettings` can be used. The user can change the background colour of an application and the colour will be saved if the application is deactivated or closed. First, a new public property named `Background` of the type `Brush` is added to the *App.xaml.cs* above the constructor.

```
    public partial class App : Application
    {
        public Brush Background { get; set; }
        ...
    }
```

Listing 44: Background Property - App.xaml.cs

In the class `MainPage` the namespace `System.IO.IsolatedStorage` must be included. The property that was defined in the *App.xaml.cs* will be called after the constructor. If the property contains a value it will be added to the `ContentPanel`. To change and save a new colour value, a button with the event `btnColor_Click` was created. Within the method, a random colour will be created and added to the background and saved in the public property of the `App` class.

```
using System.IO.IsolatedStorage;
using System.IO;

namespace IsolatedStorageI
{
    public partial class IsolatedStorage : PhoneApplicationPage
    {
        public IsolatedStorage()
        {
            InitializeComponent();

            Brush brush = (Application.Current as App).Background;
            if (brush != null)
            {
                ContentPanel.Background = brush;
            }
        }

        private void btnColor_Click(object sender, RoutedEventArgs e)
        {
            Random random = new Random();
         SolidColorBrush solidColorBrush = new SolidColorBrush(Color.FromArgb(255,
         (byte)random.Next(256), (byte)random.Next(256), (byte)random.Next(256)));

            ContentPanel.Background = solidColorBrush;

            (Application.Current as App).Background = solidColorBrush;
        }
    }
}
```

Listing 45: MainPage.xaml.cs - IsolatedStroageSettings

Then for each state of the application (`Launching`, `Activated`, `Deactivated` and `Closing`) the background colour must be saved or loaded. Therefore, two methods are implemented in the *App.xaml.cs*.

```
private void Application_Launching(object sender, LaunchingEventArgs e)
      {
          LoadSettings();
      }

      private void Application_Activated(object sender, ActivatedEventArgs e)
      {
          LoadSettings();
      }

      private void Application_Deactivated(object sender, DeactivatedEventArgs e)
      {
          SaveSettings();
      }

      private void Application_Closing(object sender, ClosingEventArgs e)
      {
          SaveSettings();
      }
```

Listing 46: App.xaml.cs - IsolatedStorageSettings

The `SaveSettings` and `LoadSettings` methods use the `IsolatedStorageSet-tings` object to store a key-value based data in the isolated storage.

```
void SaveSettings()
{
   IsolatedStorageSettings settings = IsolatedStorageSettings.ApplicationSettings;

   if (Background is SolidColorBrush)

   settings["backgroundColor"] = (Background as SolidColorBrush).Color;
   settings.Save();
}

void LoadSettings()
{
   IsolatedStorageSettings settings = IsolatedStorageSettings.ApplicationSettings;
   Color color;

   if (settings.TryGetValue<Color>("backgroundColor", out color))
       Background = new SolidColorBrush(color);
}
```

Listing 47: App.xaml.cs - Save and Load Method

4.6.2 IsolatedStorageFile

To create files and directories in the isolated storage the class `IsolatedStor-ageFile` must be used. The `System.IO` and `System.IO.IsolatedStroage` namespace is required to access this class. In the following example a directory named `Subdirectory1` will be created and the content from a `TextBox` will be saved in a file named *HelloWorld.txt*.

To create a new directory, a new object of the type `IsolatedStorageFile` is required. With the method `GetUserStoreForApplication` you get a new instance of the `IsolatedStorageFile` class. For the folder the method `CreateDirectory` with the name of the directory is invoked.

```
...
using System.IO;
using System.IO.IsolatedStorage;

namespace IsolatedStorageII
{
    public partial class MainPage : PhoneApplicationPage
    {
        // Constructor
        public MainPage()
        {
            ...
            try
            {
                IsolatedStorageFile file =
                    IsolatedStorageFile.GetUserStoreForApplication();

                file.CreateDirectory("Subdirectory1");
            }
            catch (IsolatedStorageException iso)
            {
                //error handling
            }
        }
        ...
    }
}
```

Listing 48: MainPage.xaml.cs - IsolatedStorageFile

In the *MainPage.xaml* a `TextBox` named `textbox1`, a `TextBlock`, a `Load` and `Save` button was added. The two buttons have a click event. Additionally a `Loaded` event (`Loaded="PhoneApplicationPage_Loaded"`) was added (Listing 49). The `Loaded` event is used in the code-behind to disable the `Load` button on screen if no file exist.

```
<phone:PhoneApplicationPage
    x:Class="IsolatedStorageII.MainPage"
    ...
    Loaded="PhoneApplicationPage_Loaded"
    ...
    <Grid x:Name="LayoutRoot" Background="Transparent">
        ...
        <Grid x:Name="ContentPanel" Grid.Row="1" Margin="12,0,12,0">
            <Button Content="Load" Margin="72,284,224,269"
                    Name="loadBtn" Click="loadBtn_Click" />
            <Button Content="Save" Margin="255,284,40,269"
                    Name="saveBtn" Click="saveBtn_Click" />
            <TextBox Margin="70,204,39,360" Name="textBox1" />
            <TextBlock Margin="83,182,40,411" Name="textBlock1"
                       Text="Enter Text:" />
        </Grid>
    </Grid>
    ….
</phone:PhoneApplicationPage>
```

Listing 49: XAML Code - Input IsolatedStorageFile

The code-behind method for the `Loaded` event is shown below. Here the `FileExists` method is used to check whether the file exists or not. If so, the button is enabled, otherwise not.

```
        private void PhoneApplicationPage_Loaded(object sender, RoutedEventArgs e)
        {
            IsolatedStorageFile file =
                IsolatedStorageFile.GetUserStoreForApplication();
            if (!file.FileExists("Subdirectory1/HelloWorld.txt"))
            {
                loadBtn.IsEnabled = false;
            }
        }
```

Listing 50: Loaded Method

Listing 51 shows the method for the save button, which will store the file. In this example the `using` statement wraps the objects. The `using` statement is for classes that support the *IDisposable* interface. If the object is no longer needed, the allocated resources are released (before the garbage collector comes) [Will12].

The method of the click event of the save button contains the code to write the file. First, an object of the type `IsolatedStorageFile` is declared and instantiated. With the `CreateFile` method a new file in the isolated storage is created, opened and the `IsolatedStorageFileStream` is returned from the method. If the file already exists, the original file is deleted and recreated. Subsequently, with a `StreamWriter` the text of the `textBox` is written into the file, the content `textBox` is cleared and the load button enabled.

```
private void saveBtn_Click(object sender, RoutedEventArgs e)
        {
            using (IsolatedStorageFile file =
                IsolatedStorageFile.GetUserStoreForApplication())
            {
                using (IsolatedStorageFileStream stream =
                    file.CreateFile("Subdirectory1/HelloWorld.txt"))
                {
                    using (StreamWriter sw = new StreamWriter(stream))
                    {
                        sw.Write(textBox1.Text);
                        sw.Close();
                        textBox1.Text = "";
                        loadBtn.IsEnabled = true;
                    }
                }
            }
        }
```

Listing 51: Save Button - IsolatedStorageFile

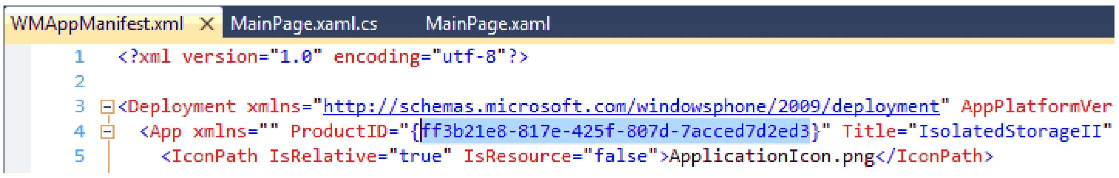
The method for the load event is very similar to the save event which is shown below. Instead of the `CreateFile` Method the `OpenFile` method is used and a `StreamWriter` is required to read the content of the file.

```
        private void loadBtn_Click(object sender, RoutedEventArgs e)
        {

            using (IsolatedStorageFile file =
                IsolatedStorageFile.GetUserStoreForApplication())
            {
                using (IsolatedStorageFileStream stream =
                    file.OpenFile("Subdirectory1/HelloWorld.txt", FileMode.Open))
                {
                    using (StreamReader sr= new StreamReader(stream))
                    {
                        textBox1.Text = sr.ReadToEnd();
                    }
                }
            }
        }
```

Listing 52: Load Button - IsolatedStorageFile

4.6.3 Isolated Storage Explorer Tool

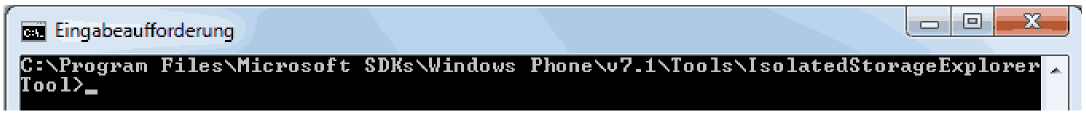
The Windows Phone SDK provides a useful tool for dealing with the isolated storage. It is the Isolated Storage Explorer Tool (ISETool.exe). With this line-command tool you can list, copy and replace files and directories in the isolated storage of the phone. After a standard installation of the SDK the tool will be found in the following folder: `C:\Program Files\Microsoft SDKs\Windows Phone\v7.1\Tools\IsolatedStorageExplorerTool\ISETool.exe`. The tool works with the emulator as well as with a developer device. To use the ISETool, the emulator or device must be running and the `ProductID` of the application is needed. The `ProductID` attribute is contained in the *WPAppManifest.xml* file of the project. Figure 43 shows the ID for the application, which was created in this chapter.



```
<?xml version="1.0" encoding="utf-8"?>

<Deployment xmlns="http://schemas.microsoft.com/windowsphone/2009/deployment" AppPlatformVer
  <App xmlns="" ProductID="{ff3b21e8-817e-425f-807d-7acced7d2ed3}" Title="IsolatedStorageII"
    <IconPath IsRelative="true" IsResource="false">ApplicationIcon.png</IconPath>
```

Figure 43: ProductID - WMAppManifes.xml

In the command-prompt of Windows you can navigate to the ISETool (Figure 44).



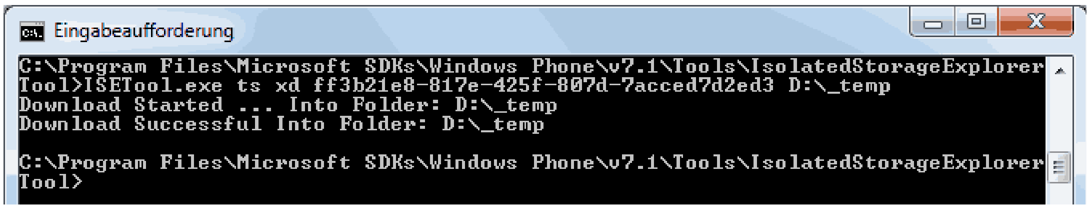
```
C:\Program Files\Microsoft SDKs\Windows Phone\v7.1\Tools\IsolatedStorageExplorer
Tool>_
```

Figure 44: ISETool

Then the following command will copy the isolated storage to a folder named „_temp“ on drive „D:“.

```
ISETool.exe ts xd ff3b21e8-817e-425f-807d-7acced7d2ed3 D:\_temp
```

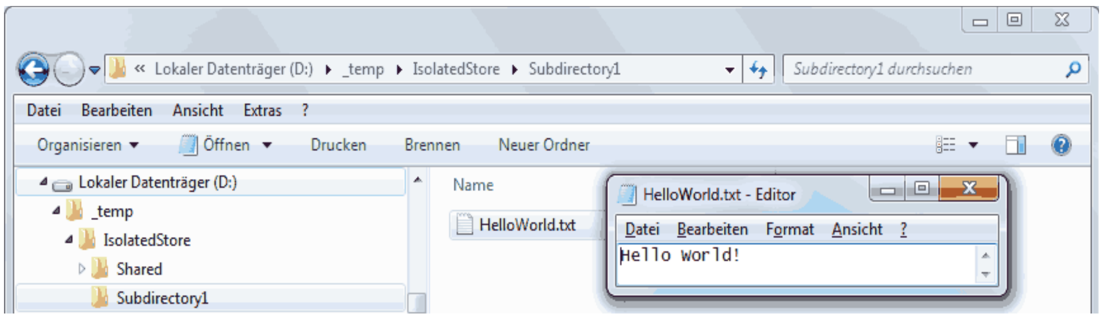
With „ISETool.exe“ the program is run. The command-line option „ts“ copies the isolated storage directory from the emulator to the computer and „xd“ stands for the emulator. After that the ProductID and the target directory follows. When the program is executed a download report is listed (Figure 45).



```
C:\Program Files\Microsoft SDKs\Windows Phone\v7.1\Tools\IsolatedStorageExplorer
Tool>ISETool.exe ts xd ff3b21e8-817e-425f-807d-7acced7d2ed3 D:\_temp
Download Started ... Into Folder: D:\_temp
Download Successful Into Folder: D:\_temp

C:\Program Files\Microsoft SDKs\Windows Phone\v7.1\Tools\IsolatedStorageExplorer
Tool>
```

Figure 45: Download IsolatedStorageFile

When the download is successful the target folder contains the isolated storage from the emulator. Figure 46 shows the file and directory which was created in the chapter 4.6.2.


« Lokaler Datenträger (D:) ▸ _temp ▸ IsolatedStore ▸ Subdirectory1
Subdirectory1 durchsuchen
Datei Bearbeiten Ansicht Extras ?
Organisieren Öffnen Drucken Brennen Neuer Ordner
Lokaler Datenträger (D:)
_temp
IsolatedStore
Shared
Subdirectory1
Name
HelloWorld.txt
HelloWorld.txt - Editor
Datei Bearbeiten Format Ansicht ?

```
Hello world!
```

Figure 46: IsolateStorageFile

4.6.4 XmlSerializer

To write simple text files, the method in chapter 4.6.2 is sufficient. If, however, more complex objects and their data should be stored, you need a different solution. With the `XmlSerializer` objects can be written in an XML file and then be read again. For this, the `System.Xml.Serialization` reference is needed.

The following example shows how a generic list can be stored in an XML file. A list named `student` with two objects is created. In region `serialize` the two objects will be stored in a file named *Students.xml* and in region `deserialize` the file will be read again and displayed in a `listBox` named `lbStudents`. In both regions, a new instance of `IsolatedStorageFile` class and `IsolatedStorage-FileStream` class is created. These have already been presented in Chapter 4.6.2. In `serialize` a new object of `XmlSerializer` is created. A parameter of the type `List<Student>` is passed. With the method `Serialize` the stream and the list are passed. Then the list is cleared. In region `deserialize` the method `Deserialize` is called. As parameter the stream is passed again. The return value is converted back to an object of the type `List`. Then the list is assigned to a XAML UI element (`listBox`) named `lbStudents`.

```
...
using System.IO;
using System.IO.IsolatedStorage;
using System.Xml.Serialization;

namespace XMLSerzializer
{
    public partial class MainPage : PhoneApplicationPage
    {
        private List<Student> student;
        // Constructor
        public MainPage()
        {
            InitializeComponent();

            student = new List<Student>();

            student.Add(new Student() { FirstName = "Jane",
                LastName = "Doe", StudentNumber = 123456 });
            student.Add(new Student() { FirstName = "John",
                LastName = "Doe", StudentNumber = 987654 });

            #region "serialize"
            using (IsolatedStorageFile file =
                IsolatedStorageFile.GetUserStoreForApplication())
            {
                using (IsolatedStorageFileStream stream =
                    file.CreateFile("Students.xml"))
                {
                    XmlSerializer xs = new XmlSerializer(typeof(List<Student>));
                    xs.Serialize(stream, student);
                    student = null;
                }
            }
            #endregion


            # region "deserialize"
            using (IsolatedStorageFile file =
                IsolatedStorageFile.GetUserStoreForApplication())
            {
                using (IsolatedStorageFileStream stream =
                    file.OpenFile("Students.xml", FileMode.Open))
                {
                    XmlSerializer xs = new XmlSerializer(typeof(List<Student>));
                    student = (List<Student>)xs.Deserialize(stream);
                    lbStudents.ItemsSource = student;
                }
            }
            #endregion
        }

        public class Student
        {
            public long StudentNumber {get; set;}
            public string FirstName { get; set; }
            public string LastName { get; set; }
        }
    }
}
```

Listing 53: XML Serializer

With the ISETool, the created file can be viewed (Listing 54).

```
<?xml version="1.0"?>
<ArrayOfStudent xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <Student>
    <StudentNumber>123456</StudentNumber>
    <FirstName>Jane</FirstName>
    <LastName>Doe</LastName>
  </Student>
  <Student>
    <StudentNumber>987654</StudentNumber>
    <FirstName>John</FirstName>
    <LastName>Doe</LastName>
  </Student>
</ArrayOfStudent>
```

Listing 54: XML File - Isolated Storage

4.6.5 LINQ to SQL

With Windows Phone OS 7.1 local databases can be created. In the previous version (OS 7.0) only third-party solutions were available. Microsoft implemented LINQ to SQL to work with databases. LINQ to SQL is an object-orientated approach, which maps an object model to a data model of a relational database. The object model is expressed in the programming language. When an application is running, the LINQ to SQL runtime will translate the Language Integrated Query (LINQ) to the Structured Query Language (SQL) and sends them to the database. The results of the database are then translated back to objects that you can work with in your own programming language.

The following example which is based on the MSDN library [MSDN12], shows a list that uses a local database as a storage where students with their first name, last name and student number can be added and deleted. Figure 47 displays the application.

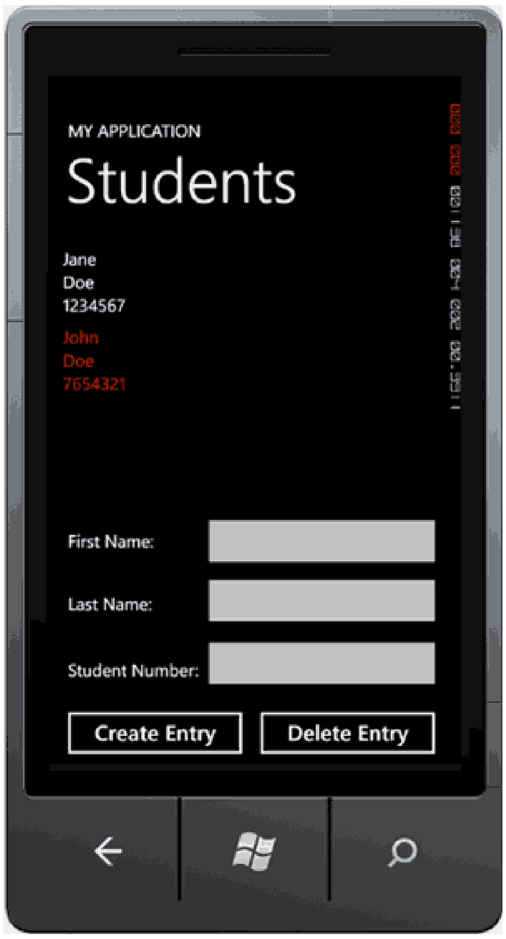


Figure 47: UI - LINQ to SQL

In the `TextBox` elements you can enter the data of a student and with the `Create Entry` button the entry will be added to a `ListBox` and stored in the database. To delete an entry, the desired entry must be selected in the `ListBox` (shown in red in Figure 47) and can then be removed from the list and database with the `Delete Entry` button.

For the user interface the following XAML-Code is implemented instead of the `Grid` named `LayoutRoot` of the default page. A `ListBox` with data binding (cf.

chapter 4.5.3) is added to display the database entries. Two `Button` elements (one for insertion and one for deletion) with a click event, three `TextBox` elements for the input and three `TextBlock` elements for the description of the input fields. Note that the text box for the student number has the attribute `inputscope="number"`, in order to change the keyboard layout for the input.

```
<Grid x:Name="ContentPanel" Margin="12,160,12,0" Grid.RowSpan="2">
            <Grid.RowDefinitions>
                <RowDefinition Height="5*"/>
                <RowDefinition Height="5*"/>
            </Grid.RowDefinitions>
            <ListBox ItemsSource="{Binding lbStudents}" x:Name="lbStudent"
                     Margin="6,6,12,11" Height="287">
                <ListBox.ItemTemplate>
                    <DataTemplate>
                        <StackPanel>
                            <TextBlock Text="{Binding FirstName}" />
                            <TextBlock Text="{Binding LastName}" />
                            <TextBlock Text="{Binding StudentNumber}" Margin="0,0,0,10" />
                        </StackPanel>
                    </DataTemplate>
                </ListBox.ItemTemplate>
            </ListBox>
            <Button Content="Create Entry" Height="72" Width="226"  Margin="0,232,230,0"
              Name="btnCreateEntry" Click="btnCreateEntry_Click" Grid.Row="1"/>
            <Button Content="Delete Entry" Height="72" Width="226" Margin="224,232,6,0"
              Name="btnDeleteEntry" Click="btnDeleteEntry_Click" Grid.Row="1"/>
            <TextBox Height="72" Margin="164,9,6,223" Name="txtBoxFirstName"
                     Text=""  Width="286" Grid.Row="1" />
            <TextBox Height="72" Margin="164,78,6,154" Name="txtBoxLastName"
                     Text="" Width="286" Grid.Row="1" />
            <TextBox Height="72" Margin="164,151,6,81" Name="txtBoxStudentNumber"
                     Text="" Width="286" Grid.Row="1" InputScope="number"/>
            <TextBlock Height="30" Margin="12,30,286,244" Name="textBlock1"
                       Text="First Name:" Grid.Row="1"/>
            <TextBlock Height="30" Margin="12,103,286,171" Name="textBlock2"
                       Text="Last Name:" Grid.Row="1"/>
            <TextBlock Height="30" Margin="12,180,286,94" Name="textBlock3"
                       Text="Student Number:" Grid.Row="1" />
        </Grid>
```

Listing 55: XAML-Code - LINQ to SQL

In the following section, the required classes and methods for the application will be explained step by step. In addition, the entire code-behind file (*Main-Page.xaml.cs*) will be found in the appendix (Appendix I).

First, for each database application a reference to the assembly *System.Data.Linq* is required. The assembly can be added under the menu item *Project -> Add Reference*. Then, the following namespaces are required.

```
using System.Data.Linq;
using System.Data.Linq.Mapping;
using System.ComponentModel;
using System.Collections.ObjectModel;
```

Listing 56: Namespaces for LINQ to SQL

Below the `MainPage` class, the `Student` class (Listing 57) is added which represents the table of the database. With the `Table` attribute as a metainformation on the top of the class, the `Student` class is associated with the `Student` table in the database. The additional parameter `Name` specifies the table name and is optional.

The properties of the class are decorated with the attribute `Column` and describe the table columns. The `Column` attribute has a variety of properties for an exact mapping to the database. The `id` property is declared as the primary key. `IsDbGenerated` means that the database will generate the primary key.

`DbType` describes the data mapping between the object model type (CLR) and the database type (SQL). For the other columns in the example there is no type defined and thus the automatic mapping is used. A detailed “Type Mapping Run-time Behavior Matrix” overview can be found on the Microsoft Developer Network under `http://msdn.microsoft.com/en-gb/library/bb386947.aspx` [MSDN13].

In addition, the `id` column contains a `CanBeNull` property with the value `false` which indicates that the column can not be empty and an `AutoSync` property that specifies the primary key is inserted only at the entry (for performance reasons).

The `Student` class also implements the `INotifyPropertyChanged` and the `INotifyPropertyChanging` interface. The former was already explained in chapter 4.5.2 . The latter one is used for a better memory management of LINQ to SQL. The change tracking of LINQ to SQL works by default with two copies of an object. One object will remain unchanged and the other one can be changed by the application. With the two objects the LINQ to SQL runtime can determine which properties have been updated and submit only the changes to the database. “The `INotifyPropertyChanging` interface allows the application to notify the `DataContext` (Listing 58) when it is modifying a property that will ultimately be submitted as an update to the database. The `DataContext` can use that notification as a trigger to create the copy. This way, only the items that are actually changing need to be duplicated” [MSDN14].

```
    [Table(Name = "Student")]
    public class Student : INotifyPropertyChanged, INotifyPropertyChanging
    {
        private int id;
        private string firstName;
        ...
        [Column(IsPrimaryKey = true, IsDbGenerated = true, DbType = "INT NOT NULL Identity",
                CanBeNull = false, AutoSync = AutoSync.OnInsert)]
        public int ID
        {
            get
            {
                return id;
            }
            set
            {
                if (id != value)
                {
                    NotifyPropertyChanging("ID");
                    id = value;
                    NotifyPropertyChanged("ID");
                }
            }
        }

        [Column]
        public string FirstName
        {
            get
            {
                return firstName;
            }
            set
            {
                if (firstName != value)
                {
                    NotifyPropertyChanging("FirstName");
                    firstName = value;
                    NotifyPropertyChanged("FirstName");
                }
            }
        }
        ...
        public event PropertyChangedEventHandler PropertyChanged;
        private void NotifyPropertyChanged(string propertyName)
        {
            if (PropertyChanged != null)
            {
                PropertyChanged(this, new PropertyChangedEventArgs(propertyName));
            }
        }
        public event PropertyChangingEventHandler PropertyChanging;

        private void NotifyPropertyChanging(string propertyName)
        {
            if (PropertyChanging != null)
            {
                PropertyChanging(this, new PropertyChangingEventArgs(propertyName));
            }
        }
    }
```
Listing 57: Student Table - LINQ to SQL

Below the `MainPage` and `Student` class the `StudentsDataContext` class (Listing 58) is added. The class acts as a link between the database table and the objects and inherits from the `DataContext` class. The class is responsible for reading, writing, updating, deleting table entries and also for tracking changes. The `DBConnectionString` specifies the connection to the database which is named `Students.sdf`. Then the `base(connectionString)` constructor is called and the connection string is passed. Finally a database table named `StudentsTable` is declared.

```
public class StudentsDataContext : DataContext
{
   public static string DBConnectionString = "Data Source=isostore:/Students.sdf";
   public StudentsDataContext(string connectionString)
       : base(connectionString)
   { }
   public Table<Student> StudentsTable;
}
```

Listing 58: Data Context – LINQ to SQL

Listing 59 shows the code to create the database. The code is added to *App.xaml.cs* inside the class constructer for the `App` class that the database is present before the main page runs.

```
using (StudentsDataContext db = new StudentsDataContext
                (StudentsDataContext.DBConnectionString))
            {
                if (db.DatabaseExists() == false)
                {
                    db.CreateDatabase();
                }
            }
```

Listing 59: Create Database – LINQ to SQL

Above the class constructor in the `MainPage` class a global variable of the type `StudentsDataContext studentdDB` is declared, which will be instantiated in the constructor (Listing 60). Thus, the entire class has access to the `StudentsTable`. The `ObservableCollection` is used for binding the data to the UI.

```
        private StudentsDataContext studentsDB;

        private ObservableCollection<Student> students;
        public ObservableCollection<Student> Students
        {
            get
            {
                return students;
            }
            set
            {
                if (students != value)
                {
                    students = value;
                    NotifyPropertyChanged("Students");
                }
            }
        }

        // Constructor
        public MainPage()
        {
            InitializeComponent();

            studentsDB = new StudentsDataContext
```

```
                (StudentsDataContext.DBConnectionString);
            this.DataContext = this;

        }
```

Listing 60: ObservableCollection - LINQ to SQL

Below the class constructor the `OnNaviagtedTo` method (Listing 61) is implemented in which the query is executed. The result of the query is placed into a `Students` collection. Then the collection is assigned to a UI element (`listBox`) named `lbStudents`.

```
        protected override void OnNavigatedTo
            (System.Windows.Navigation.NavigationEventArgs e)
        {
            var studentsInDB = from Student student in studentsDB.StudentsTable
                               select student;

            Students = new ObservableCollection<Student>(studentsInDB);
            lbStudent.ItemsSource = students;
            base.OnNavigatedTo(e);
        }
```

Listing 61: OnNavigatiedTo - LINQ to SQL

Following, the `OnNavigatedTo` method, the `NotifyPropertyChanged` method for data binding and the click events for the UI buttons are defined (Listing 62).

In the `CreateButton` a new `Student` is instantiated and the contents of the input fields are assigned to the properties of the `Student`. Then the object is added to the collection. The `InsertOnSubmit` method belongs to the `StudentsTable` and will register the object for insertion to the database. With the `SubmitChanges` method the object will be added to the database.

In the method for the delete button the selected `ListBox` item will be casted into a new `Student` object. Then the object is removed from the collection, registered for deletion (`DeleteOnSubmit`) and finally removed from the database (`SubmitChanges`).

```
public event PropertyChangedEventHandler PropertyChanged;
        private void NotifyPropertyChanged(string propertyName)
        {
            if (PropertyChanged != null)
            {
                PropertyChanged(this, new PropertyChangedEventArgs(propertyName));
            }
        }

        private void btnCreateEntry_Click(object sender, RoutedEventArgs e)
        {
            Student newStudent = new Student();
            {
                newStudent.FirstName = txtBoxFirstName.Text;
                newStudent.LastName = txtBoxLastName.Text;
                newStudent.StudentNumber = Int64.Parse(txtBoxStudentNumber.Text);
            }

            Students.Add(newStudent);
            studentsDB.StudentsTable.InsertOnSubmit(newStudent);
            studentsDB.SubmitChanges();
            lbStudent.ItemsSource = students;
            txtBoxFirstName.Text = "";
            txtBoxLastName.Text = "";
            txtBoxStudentNumber.Text = "";
        }

        private void btnDeleteEntry_Click(object sender, RoutedEventArgs e)
        {
            Student deleteStudent = (Student)lbStudent.SelectedItem;

            Students.Remove(deleteStudent);
            studentsDB.StudentsTable.DeleteOnSubmit(deleteStudent);
            studentsDB.SubmitChanges();
        }
```

Listing 62: Methods for the Delete and Create Button - LINQ to SQL

4.7 Push Notifications

Push Notifications are messages that are sent from a web service or web application (cloud services) to an application on the Windows Phone. Moreover, Toast and Tile Notifications can be received even though the application is not running. The information will be pushed by the cloud service in order to notify the user that something of interest has happened (e.g. news in a social network or weather alert). There are three types of Push Notifications:

Toast Notifications: A Toast Notification is an overlay message on the top of the user's current screen.

Tile Notifications: A Tile Notification is represented on the Tile on the Start Page.

Raw Notifications: A Raw Notification has no visual representation. They are used to send information to an application and can only be processed when the application is running, otherwise the information will be discarded.

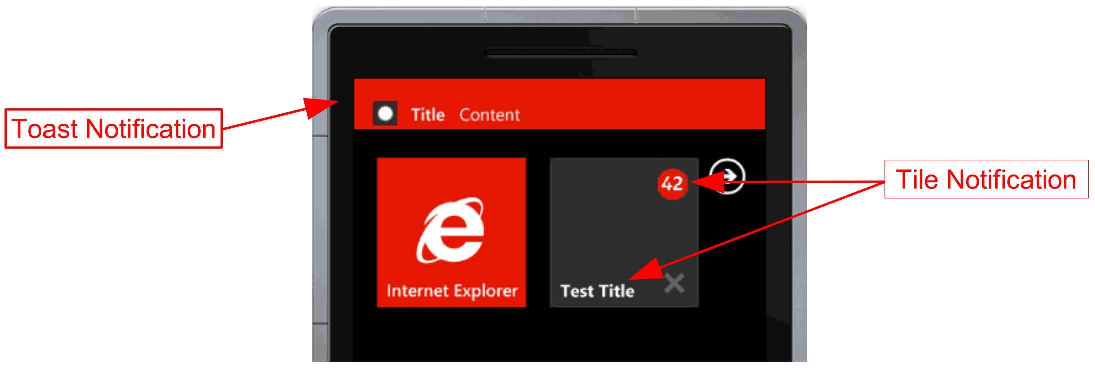


Figure 48: Notifications

Push Notification Architecture

Figure 49 shows the various components that are involved in a Push Notification. As a first step the application on the phone will request a Push Notification URI from the push client service (#1, Figure 49). Then the push client service will negotiate (#2, Figure 49) with the Microsoft Push Notification Service (MPNS) and return an URI to the application (#3, Figure 49) that uniquely identifies the device on the network. In the next step the application can provide the URI to the requested service (#4, Figure 49). Now the cloud service can send notifications to the registered device(s). It will send a HTTP POST message with an XML payload to the MPNS by using the URI which was provided by the phone (#5, Figure 49). Then the MPNS will route the message as a Push Notification to the phone (#6, Figure 49) [MSDN15].

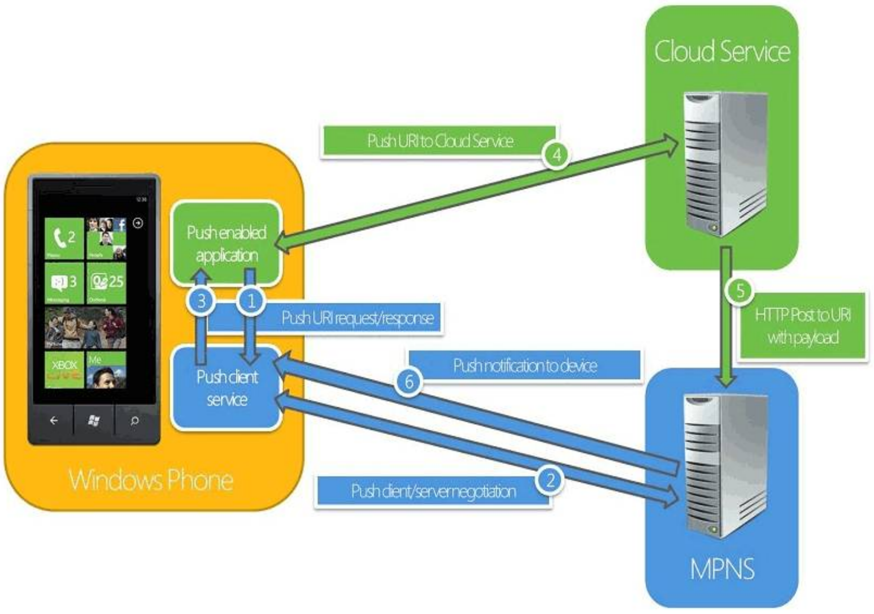

Cloud Service
Push URI to Cloud Service
4
Push enabled application
3
1
Push URI request/response
5
HTTP Post to URI with payload
Push notification to device
6
Push client service
Push client/server negotiation
2
Windows Phone
MPNS

Figure 49: Concept of Push Notifications [MSDN15]

The concepts of the three Push Notifications are very similar. Each notification has an appropriate payload, except Raw Notifications. They do not have a particular payload format. There is no system-wide way to display a raw notification and they can contain whatever data you like.

Because of the similarities of the different types of notification, in the following example only a Toast Notification is described in detail. In the appendix there is a complete example of Tile (Appendix II) and Raw Notification (Appendix III). The examples of the notifications are based on the examples in the book “Entwickeln für Windows Phone 7.5” [cf. Getz11 660-683].

4.7.1 Toast Notification

Toast Notifications have the property that the received message will disappear after a few seconds when the application is not running. The Notification can contain three elements. A Title which is defined as `wp:Text1` in the XML schema, a Content defined as `wp:Text2` and a Parameter as `wp:parm`. Either the Title or Content must be filled and the Parameter element can only be set on Windows Phone OS 7.1 or greater. This example contains two projects, one Windows Phone Application for receiving the Toast Message and a Console Application that acts as web service for demonstration purposes. If only the Windows Phone SDK is used to develop applications (and not a full version of Visual Studio as in this work), Visual C# 2010 Express is needed to build a console application. It can be obtained under: `http://www.microsoft.com/express` [Micr07].

For the Windows Phone Application the following namespaces are required.

```
using System;
using System.Diagnostics;
using System.Windows;
using Microsoft.Phone.Controls;
using Microsoft.Phone.Notification;
```

Listing 63: Toast Notification - Namespaces

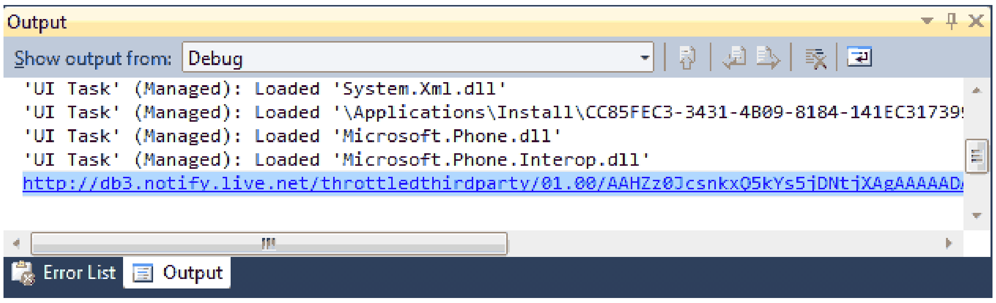
At the beginning of the class (Listing 64), a `channelName` which is an arbitrary name for the channel of the application is created and a `HttpNotificationChannel` object is declared. The `HttpNotificationChannel` is the key class and creates a channel between the MPNS and the Push Client. Then in the constructor the `Find` method will search for an existing channel object. If there is none a new channel with the `channelName` will be instantiated. Then an event handler for the event `ChannelUriUpdated` follows. Also an event handler follows in the case of an error (`ErrorOccured`) and for receiving Toast Notifications while the application is running (`ShellToastNotificationReceived`).

With the call of `channel.Open` the channel will be opened and the `BindToShell-Toast` method will activate the channel for the toast events. The event handler `channel_ShellToastNofificatonsReceived` shows how to react to a toast notification when the application runs. Only toast and raw notifications can be viewed within a running application. With the asynchrone method `BeginInvoke` of the `Dispatcher` class the notification is shown in a `MessageBox`. In the `chan-nel_ChannelUriUpdated` event the URI is shown in the output window of Visual Studio. This is only for testing purposes because normally the URI is passed back to the cloud service.

```
namespace ReceiveToast
{
    public partial class MainPage : PhoneApplicationPage
    {
        string channelName = "ToastNotification";
        HttpNotificationChannel channel;
        // Constructor
        public MainPage()
        {
            InitializeComponent();

            channel = HttpNotificationChannel.Find(channelName);
            if (channel == null)
            {
                channel = new HttpNotificationChannel(channelName);
                channel.ChannelUriUpdated += channel_ChannelUriUpdated;
                channel.ErrorOccurred += channel_ErrorOccured;
                channel.ShellToastNotificationReceived +=
                    channel_ShellToastNotificationReceived;
                channel.Open();

                channel.BindToShellToast();
            }
            else
            {
                channel.ChannelUriUpdated += channel_ChannelUriUpdated;
                channel.ErrorOccurred += channel_ErrorOccured;
                channel.ShellToastNotificationReceived +=
                    channel_ShellToastNotificationReceived;
            }
        }
        void channel_ShellToastNotificationReceived(object sender,
                NotificationEventArgs e)
        {
            Dispatcher.BeginInvoke(() => MessageBox.Show(String.Format
             ("{0} {1}", e.Collection["wp:Text1"], e.Collection["wp:Text2"])));
        }

        void channel_ErrorOccured(object sender,
            NotificationChannelErrorEventArgs e)
        {
            switch (e.ErrorType)
            {
                case ChannelErrorType.ChannelOpenFailed:
                    break;
            }
        }

        void channel_ChannelUriUpdated(object sender,
            NotificationChannelUriEventArgs e)
        {
            Debug.WriteLine(e.ChannelUri.ToString());
        }
    }
}
```

Listing 64: Receive Toast Notification

When the application is started, the channel URI can be found in the output window of Visual Studio.



```
'UI Task' (Managed): Loaded 'System.Xml.dll'
'UI Task' (Managed): Loaded '\Applications\Install\CC85FEC3-3431-4B09-8184-141EC31739
'UI Task' (Managed): Loaded 'Microsoft.Phone.dll'
'UI Task' (Managed): Loaded 'Microsoft.Phone.Interop.dll'
http://db3.notify.live.net/throttledthirdparty/01.00/AAHZz0JcsnkxQ5kYs5jDNtjXAgAAAAD
```

Figure 50: Channel URI

The URI in Figure 50 is used in the console application for sending the Push Notification.

For the console application the following namespaces are required:

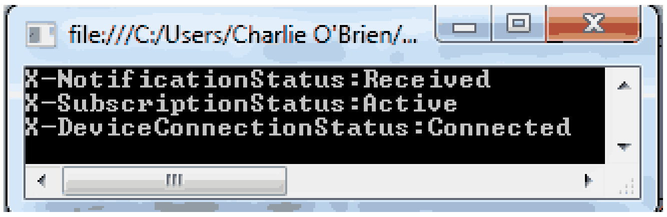
```
using System;
using System.Text;
using System.Net;
using System.IO;
```

Listing 65: Send Toast-Namespaces

In the `Main` class a `url` string is defined and contains the channel URI which was generated in the example above (Figure 50). In the variable `data` (Listing 66) the payload for a toast massage is created. Then a HTTP web request (`WebRequest`) that posts the Toast Notification to the Microsoft Push Notification Service is created. The `ContentType` is `text/xml` and the `Method` is `POST`. `POST` is the only method which will be allowed to sent Push Notifications. Additionally, the request object has two mandatory key/value pairs in the header property. The value `toast` indicates that it is a toast message and the `Notification-Class` determines the priority of the notification message. Then the payload of the notification will be transformed to a byte array for the request stream. In the try block the request stream is posted to the Microsoft Push Notification Service. Each sent notification gets a response. The response object contains information about the transmission which will be displayed in the console.

```
namespace SendToast
{
    class Program
    {
        static void Main(string[] args)
        {

            string url = "http://db3.notify.live.net/throttledthirdparty/01.00/AAH64HM3bkh" +
                         "OQJOI4j32ucHqAgAAAAADAQAAAAQUZm52OjIzOEQ2NDJDRkI5MEVFMEQ";

            string data = "<?xml version=\"1.0\" encoding=\"utf-8\"?>" +
                "<wp:Notification xmlns:wp=\"WPNotification\">" +
                   "<wp:Toast>" +
                        "<wp:Text1>Hello</wp:Text1>" +
                        "<wp:Text2>This is a toast notification!</wp:Text2>" +
                        "<wp:Param>/SecondPage.xaml?parameter=Hello World!</wp:Param>" +
                   "</wp:Toast> " +
                "</wp:Notification>";

            WebRequest request = WebRequest.Create(url);

            request.ContentType = "text/xml";
            request.Method = "POST";
            request.Headers.Add("X-WindowsPhone-Target", "toast");
            request.Headers.Add("X-NotificationClass", "2");

            byte[] bytes = Encoding.UTF8.GetBytes(data);

            using (Stream requestStream = request.GetRequestStream())
            {
                requestStream.Write(bytes, 0, bytes.Length);
                try
                {
                    HttpWebResponse response = (HttpWebResponse)request.GetResponse();
                    string notificationStatus = response.Headers["X-NotificationStatus"];
                    string subscriptionStatus = response.Headers["X-SubscriptionStatus"];
                    string connectionStatus = response.Headers["X-DeviceConnectionStatus"];

                    Console.Write(String.Format("X-NotificationStatus:{0}\r\nX-Subscription" +
                                                "Status:{1}\r\nX-DeviceConnectionStatus:{2}",
                        notificationStatus, subscriptionStatus, connectionStatus));
                    Console.ReadKey();
                }
                catch (WebException ex)
                {
                    Console.Write("WebExeption occured with Statuscode:" +
                        ((HttpWebResponse)ex.Response).StatusCode.ToString());
                    Console.ReadKey();
                }
            }

        }
    }
}
```
Listing 66: Send Toast Notification

Figure 51 shows the status of a successful delivered message.



```
X-NotificationStatus:Received
X-SubscriptionStatus:Active
X-DeviceConnectionStatus:Connected
```

Figure 51: Response Toast Notification

4.7.2 Tile Notification

The process of sending a Tile Notification is very similar to Toast Notification. Only the header elements and the payload of the HTTP request is different. The `X-WindowsPhone-Target` element (Listing 66) in the header contains `token` instead of `toast`. Listing 67 shows the content of the payload. It contains a title string, an URI for a background image and a count element.

```
<?xml version=\"1.0\" encoding=\"utf-8\"?>
<wp:Notification xmlns:wp=\"WPNotification\">
   <wp:Tile>
       <wp:BackgroundImage>URI backgroung image</wp:BackgroundImage>
       <wp:Count>count</wp:Count>
       <wp:Title>title</wp:Title>
   </wp:Tile>
</wp:Notification>
```
Listing 67: Payload Tile Notification

An application tile on the Start screen of the emulator or device can be created when the user taps and holds the application in the application list and then selects “pin to start”.

4.7.3 Raw Notification

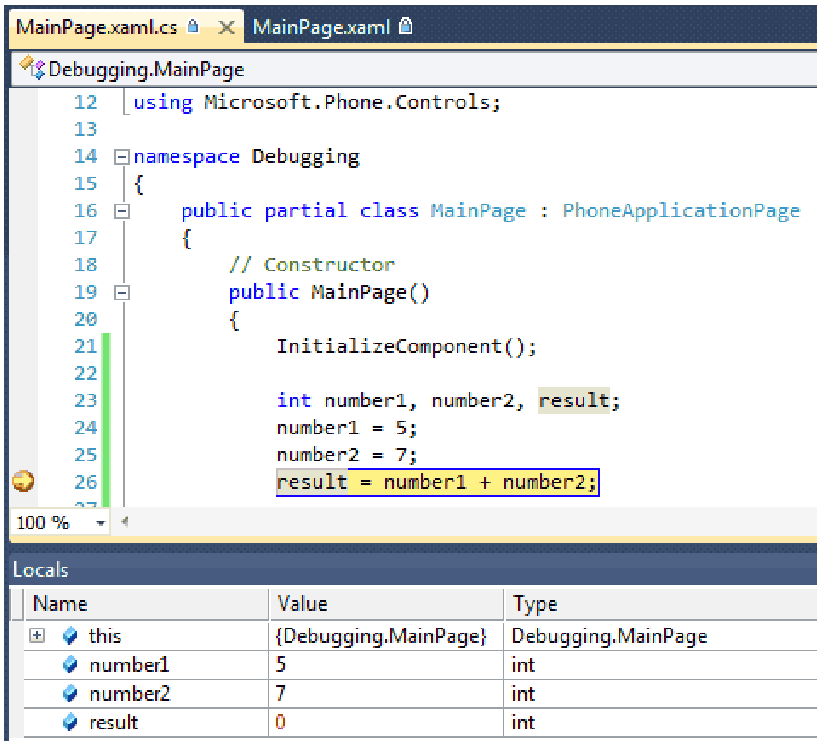
Raw notifications do not have a particular payload format. When the `X-WindowsPhone-Target` is not present in the Header of the HTTP request it is a raw notification.

5 Testing

This chapter deals with the testing of an application. First, the debugging feature of Visual Studio is described, then a unit testing framework is presented and finally the use of the Marketplace Test Kit is covered.

5.1 Debugging

In Visual Studio, it is possible to set so-called breakpoints in the code-behind editor that pauses the execution of an application. The menu *Debug* provides items such as: *Start Debugging (F5)*, *Start without Debugging (Ctrl+F5)*, *Step Into (F11)*, *Step Over (F10)* and *Toggle Breakpoint (F9)*. With *Toggle Breakpoint (F9)* you can set a breakpoint at a certain point in the source code. Also with a mouse click within the C#-Editor at the left side in the gray area a breakpoint can be placed. If the application is executed, Visual Studio stops at the breakpoint. Then, with *Step Into (F11)* the code is executed line by line. If you reach a point where a method is called, Visual Studio branches in this method. With *Step Over (F10)* it is not branched.



```
using Microsoft.Phone.Controls;

namespace Debugging
{
    public partial class MainPage : PhoneApplicationPage
    {
        // Constructor
        public MainPage()
        {
            InitializeComponent();

            int number1, number2, result;
            number1 = 5;
            number2 = 7;
            result = number1 + number2;
```

Name	Value	Type
this	{Debugging.MainPage}	Debugging.MainPage
number1	5	int
number2	7	int
result	0	int

Figure 52: Breakpoint During Execution

5.2 NUnit for Windows Phone

NUnit is a unit testing framework for .NET languages. “Unit tests are one of the corner stones of Extreme Programming (XP)” [Well09]. One programmer writes the source code, while the other one writes test cases. The open source community CodePlex provides a version of NUnit for Windows Phone for an automated testing. With this project a single developer can implement tests for an application. The project can be downloaded at `http://nunitwindowsphone7.codeplex.com/`. After the download, the project must be extracted to a directory. When the project is opened in Visual Studio, the startup project TestSamples has to be selected. The *Properties* (Figure 54) dialog can be opened when the solution (#1, Figure 53) is marked in the Solution Explorer via the item menu *Project* → *Properties* .

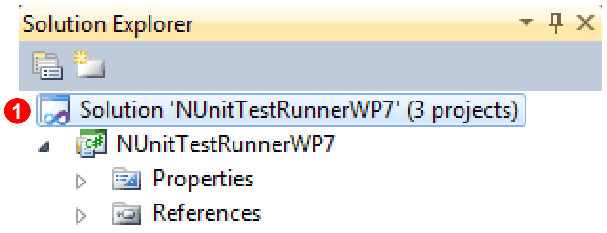
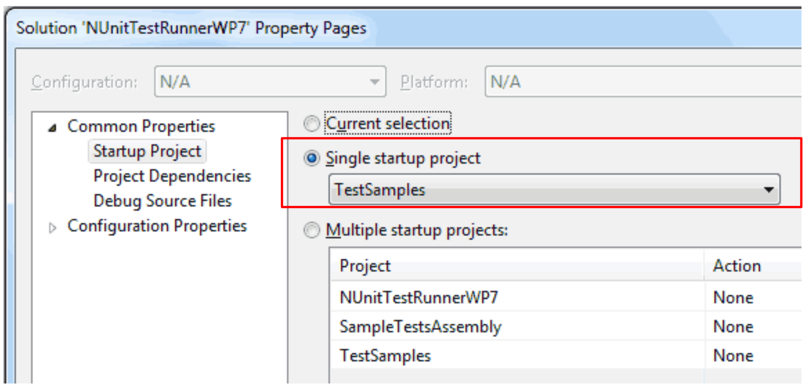


Figure 53: Solution Explorer - NUnit for Windows Phone



Project	Action
NUnitTestRunnerWP7	None
SampleTestsAssembly	None
TestSamples	None

Figure 54: Solution Properties

The following example is based on an example in the book “Windows Phone7-Apps entwickeln” of Christian Bleske [cf. Bles11 316-323]. The two created classes can also be found in the appendix (Appendix IV). First, a class named `Calc` is created in the project “TestSamples”. It contains two methods, `Add`[3] and `Div` as shown in Listing 68.

[3] In this method, an error was made on purpose to demonstrate the functionality.

```
namespace TestSamples
{
    public class Calc
   {

        public Calc()
       {
       }

        public int Add(int parameter1, int parameter2)
       {
            return parameter1 - parameter2;
       }

        public double Div(double parameter1, double parameter2)
       {
            return parameter1 / parameter2;
       }

    }
```

Listing 68: TestSamples - Calc

Then another class named `CalcTest` is created which contains the test cases (Listing 69). For this class the namespace `Nunit.Framework` is required. The attribute `TestFixture` indicates that this class contains test cases and the methods get the attribute `Test`.

```
using NUnit.Framework;

namespace TestSamples
{
    [TestFixture]
    public class CalcTest
   {
        public CalcTest()
       {
       }

        [Test]
        public void TestAdd()
       {
            ...
       }

        [Test]
        public void TestDiv()
       {
            ...
       }
   }
}
```

Listing 69: TestSamples - CalcTest

The developer of NUnit has also implemented Assertions to compare a result of a method with a specified value. The class `Assert` contains Comparison Tests, Condition Tests and Utility Methods.

Comparison Tests

For comparison tests there are two methods available, `Assert.AreEqual` and `Assert.AreSame`. The first one compares the result of a method with a value and the latter one compares references to an objects.

```
[Test]
        public void TestAdd()
        {
            Calc c = new Calc();
            Assert.AreEqual(30, c.Add(10, 20));
        }
        ...

        [Test]
        public void ObjektTest()
        {
            ObjektA a = new ObjektA();
            ObjektA c = new ObjektA();
            a = c;
            Assert.AreSame(c, a);
        }
```

Listing 70: Comparison Tests

Condition Tests

Condition Tests allow testing of true or false statements. For this the methods `Assert.IsTrue`, `Assert.IsFalse`, `Assert.IsNull` and `Assert.IsNotNull` are available.

```
        [Test]
        public void TestResultIsTrue()
       {
            Calc c = new Calc();
            Assert.IsTrue(c.ResultIsTrue());
       }
```

Listing 71: Condition Test

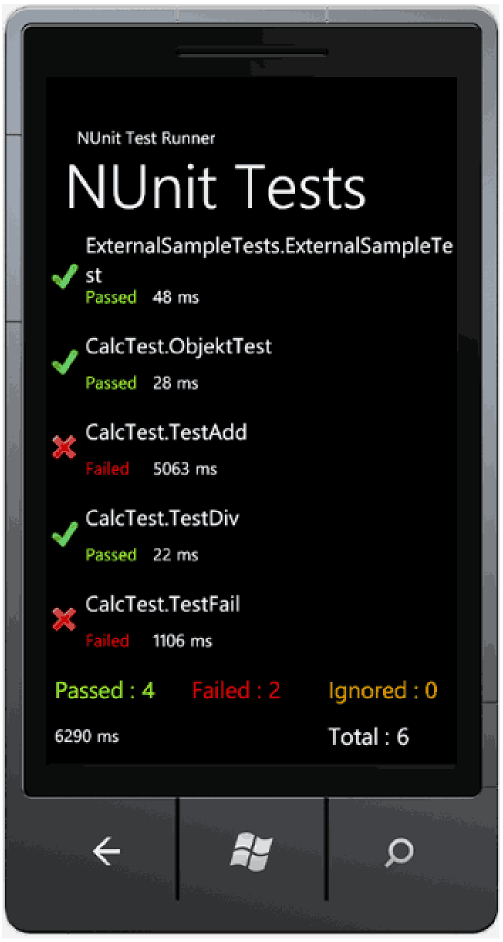
Utility Methods

With Utility Methods the developer can implement own testing methods. There are `Assert.Ignore` and `Assert.Fail`. Listing 72 shows the `Assert.Fail` method. The method `TestFail` creates an object of the class `Calc` and calls the method `ReturnValue`. If the returned value is greater then 2, the parameter of the method `Fail` is shown.

```
        [Test]
        public void TestFail()
        {
            Calc c = new Calc();
            if (c.ReturnValue() > 2)
            {
                Assert.Fail("Fehler!");
            }
        }
```

Listing 72: Utility Method

When the application is started, there is a run button. When the button is pressed, the tests will be executed (Figure 55).



NUnit Test Runner

NUnit Tests

✓ ExternalSampleTests.ExternalSampleTest
Passed 48 ms

✓ CalcTest.ObjektTest
Passed 28 ms

✗ CalcTest.TestAdd
Failed 5063 ms

✓ CalcTest.TestDiv
Passed 22 ms

✗ CalcTest.TestFail
Failed 1106 ms

Passed : 4 Failed : 2 Ignored : 0
6290 ms Total : 6

Figure 55: Emulator - NUnit Tests

5.3 Marketplace Test Kit

The Marketplace Test Kit contains a set of automated and manual tests. The kit helps the developer to prepare an application to be accepted in the Marketplace [MSDN16]. This tool is only available for Windows Phone OS 7.1. For example it contains a validation of the XAP-archive size and the content files.

There is also a capability detection. It shows the used capabilities of the applic-ation that must be specified in the *WMAppManifest.xml* of the XAP-archive.

The Marketplace Test Kit can be opened when the project is marked in the Solution Explorer of Visual Studio under the menu item *Project → Open Mar-ketplace Test Kit.*

6 Deployment

This chapter describes how a completed application will be published. It includes the registration process in the App Hub, the submission flow and the certification requirements.

6.1 The App Hub

A developer account on the App Hub is needed to publish an application in the marketplace. Furthermore, an account enables the registration of a developer device (up to three) [MSDN17]. The registration of a phone is also the only way to deploy and test an application on a real device during development process.

For the membership there is an annual fee (USD 99,00 as of March 2012). Only students have the possibility for a free membership.

For the registration there are some prerequisites. First, a Windows Live ID is needed. Under `https://signup.live.com/` an ID can be created. Then, currently, the membership payment can only be purchased with a credit card (as of March 2012). Students need a valid `DreamSpark` registration (`https://www.dreamspark.com`) which has to be associated with the Windows Live ID. This is required during the registration process as a student in the App Hub. If problems occur during the registration process to verify your student status, you can contact the support of the App Hub.

6.2 Submission and Certification

Figure 56 shows the application submission flow.

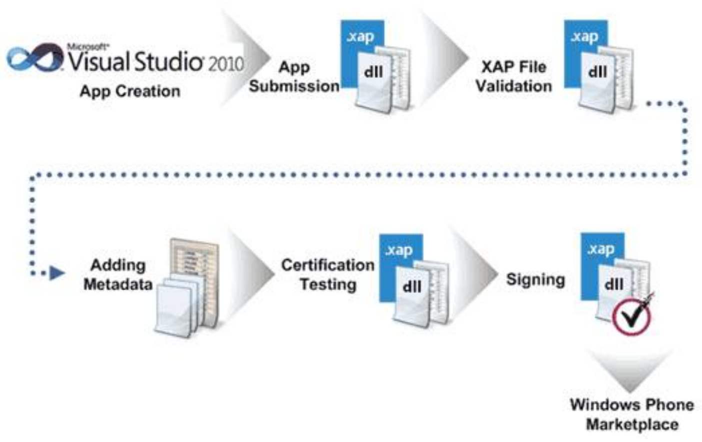
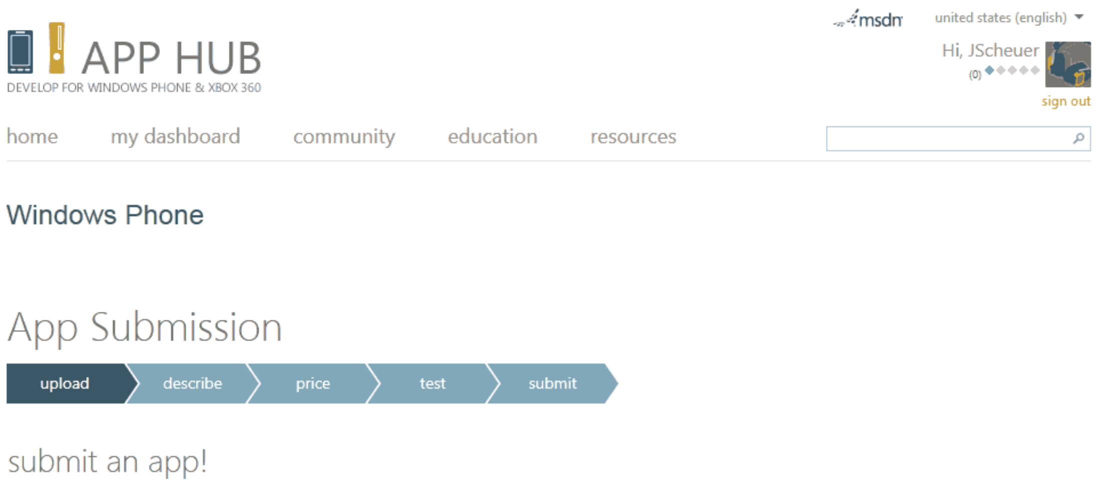


Figure 56: App Submission Flow [Chat11]

In the App Hub you can create a new application submission and in the first step you have to upload the XAP-archive.


APP HUB
DEVELOP FOR WINDOWS PHONE & XBOX 360

msdn united states (english)
Hi, JScheuer
(0)
sign out

home my dashboard community education resources

Windows Phone

App Submission

upload describe price test submit

submit an app!

Figure 57: App Hub - Upload

After the XAP-archive is uploaded, a detailed description of an application has to be made. The application must be assigned to a specific category (e.g. games, music, social). Then you have to provide a description, keywords and images for the artwork (e.g. screenshots and application tiles).

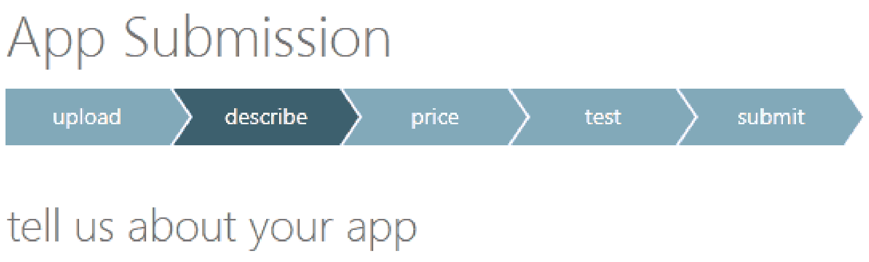


Figure 58: App Hub - Description

In the next step you have to specify the price for the application. To distribute the application for free the price must be set to 0,00. Also the target market must be selected (e.g. worldwide distribution, Europe only, etc.).

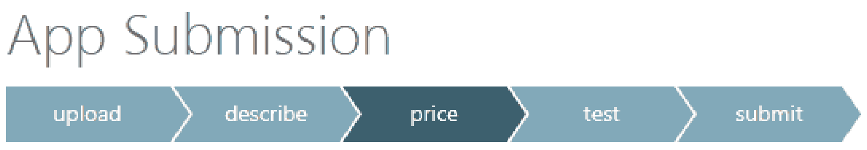


Figure 59: App Hub - Price

In the final step you can enter information for the testers of Microsoft and you have to choose an option when the application will be published (e.g. “As soon as it's certified”).

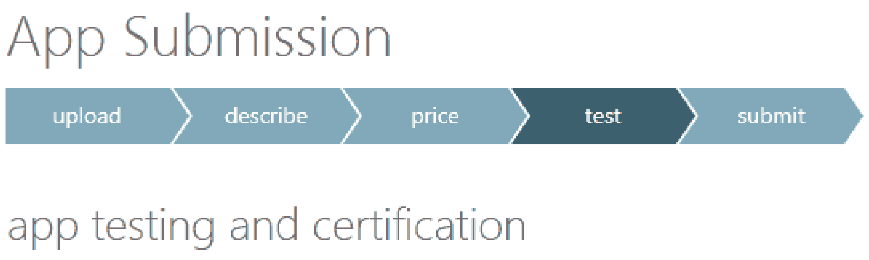


Figure 60: App Hub - Information for Testers

With a click on the “Submit” button, Microsoft will check the application. If all the data for the application is present, the app will be submitted.

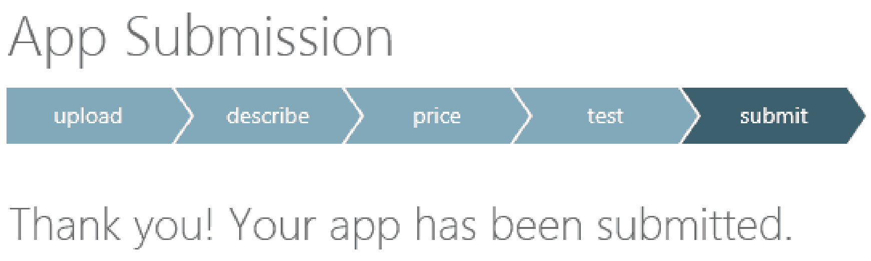


Figure 61: App Hub - Submit

6.3 Certification Requirements

The submitted application have to meet several policies and requirements before it is signed by Microsoft. There are:

- Application Policies
- Content Policies
- Application Submission Requirements
- Technical Certification Requirements
- Additional Requirements for Specific Application Types

Examples for these policies are: the size of the application (max 225 MB), contents regarding violence, alcohol, drugs and so on. The details can be found on the MSDN network under `http://msdn.microsoft.com/en-us/library/hh184843%28v=vs.92%29.aspx` [MSDN18].

7 Conclusion

With the Windows Phone SDK the development of mobile applications for Windows Phone is very comfortable. The MSDN library represents a very good documentation of the programming languages of Microsoft and is developer friendly. Also in the forums and in the blogs of MSDN the developer will find solutions quickly, if there are problems in a project. With regard to Silverlight, the distinction between the UI and the program logic is an interesting concept, because the designers can concentrate on the visual appearance of the application and the programmers to the logic.

Whether Microsoft will win with the Windows Phone in market share of smartphones remains to be seen. In any case, they are moving toward to closed systems and that should be viewed from a critical point of view. Proprietary and closed systems lead to concentration that allows the operators of the platform to benefit from network effects.

From the perspective of users and developers there is also power shift in favour of the vendors of the operating system and the operators of the platform. A reason for concern is the power to restrict content. Microsoft can decide which applications can run on the phone and which not. Now their business includes the approval of content, which opens the door to censorship.

Also, the lock-in effect in economics must be considered. Lock-in effect means to make a customer dependent on a vendor for a product or service due to high switching costs to another vendor. From a economic point of view, these lock-in strategies have a negative effect on the welfare.

References

[Anat08] Nandini S Anatharam: XAML and Silverlight. 2008, http://www.-codeproject.com/Articles/24991/XAML-and-Silverlight, retrieved on 2012-04-13

[Bles11] Bleske, Christian: Windows Phone 7-Apps entwickeln. 2011, Franzis Verlag GmbH

[Bosc11] Andrea Boschin: Windows Phone 7 - Part #5: Panorama and Pivot controls. 2011, http://www.silverlightshow.net/items/Windows-Phone-7-Part-5-Panorama-and-Pivot-controls.aspx, retrieved on 2012-04-23

[Chat11] Chatterjee, Amit: Submitting the Windows Phone application to the Marketplace. 2011, http://blogs.msdn.com/b/amit_chatterjee/archive/2011/07/30/submitting-the-windows-phone-application-to-the-marketplace.aspx, retrieved on 2012-03-07

[Ciap11] Ciappara, Clive: Windows Phone 7 Development. 2011, http://ciappara.com/2011/01/23/windows-phone-7-development/, retrieved on 2012-03-07

[Getz11] Getzmann P., Hackfort S., Nowak P: Entwickeln für Windows Phone 7.5. Arichitektur, Framework, APIs. 2011, O'Reilly Verlag Gmbh & Co. KG

[Hube10] Huber, Thomas Claudius: Silverlight 4. 2010, http://www.galileo-computing.de/download/dateien/2320/galileocomputing_silver-light_4.pdf, retrieved on 2012-02-27

[Kart12] Karthikeyan, Anbarasan: Working With Panorama Control in Windows Phone 7. 2012, http://www.c-sharpcorner.com/UploadFile/ae35ca/working-with-panorama-control-in-windows-phone-72/, retrieved on 2012-04-23

[Manj04] T Manjaly: Isolated Storage in .NET to store application data. 2004, http://www.codeproject.com/Articles/6535/Isolated-Storage-in-NET-to-store-application-data, retrieved on 2012-04-15

[Meha09] Puran Mehra: Managed code and unmanaged code in .NET. 2009, http://www.c-sharpcorner.com/uploadfile/puranindia/managed-code-and-unmanaged-code-in-net/, retrieved on 2012-04-13

[Micr01] Microsoft: Visual Basic XNA. 2012, http://code.msdn.microsoft.-com/windowsdesktop/Visual-Basic-XNA-29cd4963, retrieved on 2012-04-23

[Micr02] Microsoft News Center: Windows Phone 7: A Fresh Start for the Smartphone. 2010, http://www.microsoft.com/Presspass/Features/2010/oct10/10-11WP7main.mspx, retrieved on 2012-03-05

[Micr03] Microsoft News Center: ‘People-Centric’ Windows Phone 7.5 Update Released. 2011, http://www.microsoft.com/Presspass/Features/2011/sep11/09-27WindowsPhone75.mspx, retrieved on 2012-03-16

[Micr04] Microsoft Answers: The Windows Marketplace for Mobile for windows mobile 6.x devices is closing. 2012, http://answers.microsoft.com/en-us/winphone/forum/wp6n-wpmarketplace/the-windows-marketplace-for-mobile-for-windows/ead87a1f-1291-429c-a1ac-2406c684367b?tm=1331232502343, retrieved on 2012-03-16

[Micr05] Microsoft: Windows Phone-Updateverlauf. 2012, http://www.microsoft.com/windowsphone/de-at/howto/wp7/basics/update-history.aspx, retrieved on 2012-03-16

[Micr06] Microsoft: Windows Phone SDK 7.1. 2012, http://www.microsoft.-com/download/en/details.aspx?displaylang=en&id=27570, retrieved on 2012-03-16

[Micr07] Microsoft: Microsoft Visual Studio. 2012, http://www.microsoft.-com/visualstudio/en-us/products/2010-editions/express, retrieved on 2012-04-23

[MSDN01] MSDN: The Silverlight and XNA Frameworks for Windows Phone. 2012, http://msdn.microsoft.com/en-us/library/ff402528%28v=vs.92%29.aspx, retrieved on 2012-03-05

[MSDN02] MSDN: Gesture Support for Windows Phone. 2012, http://msdn.-microsoft.com/en-us/library/ff967546%28v=vs.92%29.aspx, retrieved on 2012-03-14

[MSDN03] MSDN: Silverlight Overview. 2012, http://msdn.microsoft.com/en-us/library/bb404700%28v=vs.95%29.aspx, retrieved on 2012-03-12

[MSDN04] MSDN: What's New in Silverlight for Windows Phone. 2012, http://msdn.microsoft.com/en-us/library/hh237342%28v=vs.95%29.aspx, retrieved on 2012-03-12

[MSDN05] MSDN: x:Class Attribute. 2012, http://msdn.microsoft.com/en-us/library/cc189082%28v=vs.95%29.aspx, retrieved on 2012-03-19

[MSDN06] MSDN: Application Manifest File for Windows Phone. 2012, http://msdn.microsoft.com/en-us/library/ff769509%28v=vs.92%29.aspx, retrieved on 2012-03-01

[MSDN07] MSDN: PhoneApplicationPage Control for Windows Phone. 2012, http://msdn.microsoft.com/en-us/library/ff402539%28v=vs.92%29.aspx, retrieved on 2012-03-02

[MSDN08] MSDN: Frame Rate Counters in Windows Phone Emulator. 2012, http://msdn.microsoft.com/en-us/library/gg588380%28v=vs.92%29.aspx, retrieved on 2012-04-21

[MSDN09] MSDN: Page Class. 2012, http://msdn.microsoft.com/en-us/library/ms611620%28v=vs.92%29.aspx, retrieved on 2012-03-03

[MSDN10] MSDN: Controls in Silverlight for Windows Phone. 2012, http://msdn.microsoft.com/en-us/library/ff426932%28v=vs.95%29.aspx, retrieved on 2012-03-04

[MSDN11] MSDN: DataTemplate Class. 2012, http://msdn.microsoft.com/en-us/library/system.windows.datatemplate.aspx, retrieved on 2012-04-15

[MSDN12] MSDN: How to: Create a Basic Local Database Application for Windows Phone. 2012, http://msdn.microsoft.com/en-us/library/hh202876%28v=vs.92%29.aspx, retrieved on 2012-03-19

[MSDN13] MSDN: SQL-CLR Type Mapping (LINQ to SQL). 2012, http://msdn.microsoft.com/en-gb/library/bb386947.aspx, retrieved on 2012-04-19

[MSDN14] MSDN: Local Database Best Practices for Windows Phone. 2012, http://msdn.microsoft.com/en-us/library/hh286406%28v=VS.92%29.aspx#BKMK_Minimizing-MemoryUsage, retrieved on 2012-04-23

[MSDN15] MSDN: Push Notifications Overview for Windows Phone. 2012, http://msdn.microsoft.com/en-us/library/ff402558%28v=vs.92%29.aspx, retrieved on 2012-03-17

[MSDN16] MSDN: Windows Phone Marketplace Test Kit. 2012, http://msdn.-microsoft.com/en-us/library/hh394032%28v=vs.92%29.aspx, retrieved on 2012-04-23

[MSDN17] MSDN: How to: Register Your Phone for Development. 2012, http://msdn.microsoft.com/en-us/library/ff769508%28v=vs.92%29.aspx, retrieved on 2012-03-07

[MSDN18] MSDN: Application Certification Requirements for Windows Phone. 2012, http://msdn.microsoft.com/en-us/library/hh184843%28v=vs.92%29.aspx, retrieved on 2012-03-07

[Petz10] Petzold, Charles: Programming Windows Phone 7. 2010, http://download.microsoft.com/download/5/0/A/50A39509-D015-410F-A8F2-A5511E5A988D/Microsoft_Press_ebook_Programming_Windows_Phone_7_PDF.pdf, Microsoft Press

[Piet08] Pietschmann, Chris: Silverlight: Anatomy of an .XAP file. 2008, http://pietschsoft.com/post/2008/03/Silverlight-Anatomy-of-an-XAP-file.aspx, retrieved on 2012-03-02

[Pren01]	Prengel, Frank: Die neue Anwendungsplattform im Überblick. 2010, http://www.microsoft.com/germany/msdn/webcasts/librar-y.aspx?id=1032453737, retrieved on 2012-03-19
[Pren02]	Prengel, Frank: Architektur der Anwendungsplattform von Windows Phone 7. 2010, http://www.microsoft.com/germany/msdn/webcasts/library.aspx?id=1032453977, retrieved on 2012-03-12
[Shin11]	Shinder, Debra: Windows Phone 7 Security Implications. 2011, http://www.windowsecurity.com/articles/Windows-Phone-7-Secur-ity-Implications.html, retrieved on 2012-02-26
[Soda01]	Sodani, Dinesh: What is Silverlight. 2010, http://beyondrelational.-com/blogs/dinesh/archive/2010/08/01/what-is-silverlight.aspx, retrieved on 2012-02-07
[Soda02]	Sodani, Dinesh: Different Ways to Bind Data Grid in Silverlight. , http://beyondrelational.com/blogs/dinesh/archive/2010/09/28/dif-ferent-ways-to-bind-data-grid-in-silverlight.aspx, retrieved on 2012-03-04
[Trip12]	Tripathi, Mani: Understanding Metro Style Applications. 2012, http://www.infosysblogs.com/microsoft/2012/01/understanding_metro_style_appl.html, retrieved on 2012-03-05
[Well09]	Wells, Don: Unit Tests. 2009, http://www.extremeprogram-ming.org/rules/unittests.html, retrieved on 2012-03-13
[Wiki01]	Wikipedia: Common Language Runtime. 2012, http://en.wikipedi-a.org/wiki/Common_Language_Runtime, retrieved on 2012-03-12
[Wilc10]	Wilcox, Jeff: Panorama and Pivot controls for Windows Phone developers. 2010, http://www.jeff.wilcox.name/2010/08/look-ing-ahead-at-panorama-and-pivot/, retrieved on 2012-04-23
[Wild12]	Wildermuth, Shawn: Essential Windows Phone 7.5. 2012, Pearson Education, Inc.
[Will12]	Wille, Christoph: Das using Schlüsselwort. , http://www.aspheute.-com/artikel/20020318.htm, retrieved on 2012-03-04

[Zieg10] Ziegler, Chris: Microsoft talks Windows Phone 7 Series development ahead of GDC: Silverlight, XNA, and no backward compatibility. 2010, http://www.engadget.com/2010/03/04/microsoft-talks-windows-phone-7-series-development-ahead-of-gdc/, retrieved on 2012-02-26

Appendix

The appendix contains the complete source code for four examples of the work.

Appendix I

Listing 73 shows the complete code of the file *MainPage.xaml.cs* from the example in chapter 4.6.5 LINQ to SQL.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Net;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Animation;
using System.Windows.Shapes;
using Microsoft.Phone.Controls;
using System.Data.Linq;
using System.Data.Linq.Mapping;
using System.ComponentModel;
using System.Collections.ObjectModel;

namespace StudentsDatabase
{
    public partial class MainPage : PhoneApplicationPage
    {
        private StudentsDataContext studentsDB;

        private ObservableCollection<Student> students;
        public ObservableCollection<Student> Students
        {
            get
            {
                return students;
            }
            set
            {
                if (students != value)
                {
                    students = value;
                    NotifyPropertyChanged("Students");
                }
            }
        }

        // Constructor
        public MainPage()
        {
            InitializeComponent();

            studentsDB = new StudentsDataContext
                (StudentsDataContext.DBConnectionString);
            this.DataContext = this;
        }


        protected override void OnNavigatedTo
            (System.Windows.Navigation.NavigationEventArgs e)
        {
            var studentsInDB = from Student student in studentsDB.StudentsTable
                                   select student;
            Students = new ObservableCollection<Student>(studentsInDB);
            base.OnNavigatedTo(e);
            lbStudent.ItemsSource = students;
        }

        public event PropertyChangedEventHandler PropertyChanged;
        private void NotifyPropertyChanged(string propertyName)
        {
            if (PropertyChanged != null)
            {
                PropertyChanged(this, new PropertyChangedEventArgs(propertyName));
            }
        }

        private void btnCreateEntry_Click(object sender, RoutedEventArgs e)
        {
            Student newStudent = new Student();
            {
                newStudent.FirstName = txtBoxFirstName.Text;
                newStudent.LastName = txtBoxLastName.Text;
                newStudent.StudentNumber = Int64.Parse(txtBoxStudentNumber.Text);
            }

            Students.Add(newStudent);
            studentsDB.StudentsTable.InsertOnSubmit(newStudent);
            studentsDB.SubmitChanges();
            lbStudent.ItemsSource = students;
            txtBoxFirstName.Text = "";
            txtBoxLastName.Text = "";
            txtBoxStudentNumber.Text = "";
        }

        private void btnDeleteEntry_Click(object sender, RoutedEventArgs e)
        {
            Student deleteStudent = (Student)lbStudent.SelectedItem;

            Students.Remove(deleteStudent);
            studentsDB.StudentsTable.DeleteOnSubmit(deleteStudent);
            studentsDB.SubmitChanges();
        }
    }

    //Table
    [Table(Name = "Student")]
    public class Student : INotifyPropertyChanged, INotifyPropertyChanging
    {
        private int id;
        private string firstName;
        private string lastName;
        private long studentNumber;

        [Column(IsPrimaryKey = true, IsDbGenerated = true, DbType = "INT NOT NULL Identity",
                CanBeNull = false, AutoSync = AutoSync.OnInsert)]
        public int ID
        {
            get
            {
                return id;
            }
            set
            {
                if (id != value)
                {
                    NotifyPropertyChanging("ID");
                    id = value;
                    NotifyPropertyChanged("ID");
                }
            }
        }

        [Column]
        public string FirstName
        {
            get
            {
                return firstName;
            }
            set
            {
                if (firstName != value)
                {
                    NotifyPropertyChanging("FirstName");
                    firstName = value;
                    NotifyPropertyChanged("FirstName");
                }
            }
        }

        [Column]
        public string LastName
        {
            get
            {
                return lastName;
            }
            set
            {
                if (lastName != value)
                {
                    NotifyPropertyChanging("LastName");
                    lastName = value;
                    NotifyPropertyChanged("LastName");
                }
            }
        }

        [Column]
        public long StudentNumber
        {
            get
            {
                return studentNumber;
            }
            set
            {
                if (studentNumber != value)
                {
                    NotifyPropertyChanging("StudentNumber");
                    studentNumber = value;
                    NotifyPropertyChanged("StudentNumber");
                }
            }
        }

        public event PropertyChangedEventHandler PropertyChanged;

        private void NotifyPropertyChanged(string propertyName)
        {
            if (PropertyChanged != null)
            {
                PropertyChanged(this, new PropertyChangedEventArgs(propertyName));
            }
        }
        public event PropertyChangingEventHandler PropertyChanging;

        private void NotifyPropertyChanging(string propertyName)
        {
            if (PropertyChanging != null)
            {
                PropertyChanging(this, new PropertyChangingEventArgs(propertyName));
            }
        }
    }

    //Datacontext
    public class StudentsDataContext : DataContext
    {
        public static string DBConnectionString = "Data Source=isostore:/Students.sdf";

        public StudentsDataContext(string connectionString)
            : base(connectionString)
        { }

        public Table<Student> StudentsTable;
    }
}
```

Listing 73: Project "StudentDatabase" - MainPage.xaml.cs

Appendix II

Listing 74 shows the code-behind to receive and Listing 75 to send a Tile Notification (cf. chapter 4.7 Push Notifications).

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Net;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Animation;
using System.Windows.Shapes;
using Microsoft.Phone.Controls;
using System.Diagnostics;
using Microsoft.Phone.Notification;

namespace ReceiveTile
{
    public partial class MainPage : PhoneApplicationPage
    {
        string channelName = "TileNotification";
        HttpNotificationChannel channel;

        // Constructor
        public MainPage()
        {
            InitializeComponent();

            channel = HttpNotificationChannel.Find(channelName);

            if (channel == null)
            {
                channel = new HttpNotificationChannel(channelName);
                channel.ChannelUriUpdated += channel_ChannelUriUpdated;
                channel.ErrorOccurred += channel_ErrorOccured;
                channel.Open();

                channel.BindToShellTile();
            }
            else
            {
                channel.ChannelUriUpdated += channel_ChannelUriUpdated;
                channel.ErrorOccurred += channel_ErrorOccured;
            }
        }

        void channel_ErrorOccured(object sender,
                NotificationChannelErrorEventArgs e)
        {
            switch (e.ErrorType)
            {
                case ChannelErrorType.ChannelOpenFailed:
                    break;
            }
        }

        void channel_ChannelUriUpdated(object sender,
            NotificationChannelUriEventArgs e)
        {
            Debug.WriteLine(e.ChannelUri.ToString());
        }
    }
}
```
Listing 74: Project "ReceiveTile" – MainPage.xaml.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Net;
using System.IO;

namespace SentTile
{
    class Program
    {
        static void Main(string[] args)
        {
            string url = "http://db3.notify.live.net/throttledthirdparty/01.00/AAH64H" +
                         "M3bkhOQJOI4j32ucHqAgAAAAADAgAAAAQUZm52OjIzOEQ2NDJDRkI5MEVFMEQ";

            string data = "<?xml version=\"1.0\" encoding=\"utf-8\"?>" +
                "<wp:Notification xmlns:wp=\"WPNotification\">" +
                   "<wp:Tile>" +
                        "<wp:BackgroundImage></wp:BackgroundImage>" +
                        "<wp:Count>3</wp:Count>" +
                        "<wp:Title>New Messages</wp:Title>" +
                   "</wp:Tile> " +
                "</wp:Notification>";

            WebRequest request = WebRequest.Create(url);

            request.ContentType = "text/xml";
            request.Method = "POST";
            request.Headers.Add("X-WindowsPhone-Target", "token");
            request.Headers.Add("X-NotificationClass", "1");

            byte[] bytes = Encoding.UTF8.GetBytes(data);

            using (Stream requestStream = request.GetRequestStream())
            {
                requestStream.Write(bytes, 0, bytes.Length);
                try
                {
                    HttpWebResponse response = (HttpWebResponse)request.GetResponse();
                    string notificationStatus = response.Headers["X-NotificationStatus"];
                    string subscriptionStatus = response.Headers["X-SubscriptionStatus"];
                    string connectionStatus = response.Headers["X-DeviceConnectionStatus"];

                    Console.Write(String.Format("X-NotificationStatus:{0}\r\nX-" +
                        "SubscriptionStatus:{1}\r\nX-DeviceConnectionStatus:{2}",
                        notificationStatus, subscriptionStatus, connectionStatus));
                    Console.ReadKey();
                }
                catch (WebException ex)
                {
                    Console.Write("WebExeption occured with Statuscode:" +
                        ((HttpWebResponse)ex.Response).StatusCode.ToString());
                    Console.ReadKey();
                }
            }
        }
    }
}
```
Listing 75: Project "SentTile" - Program.cs

Appendix III

Listing 76 shows the XAML-code for the UI. Listing 77 shows the code-behind to receive and Listing 78 to send a Raw Notification (cf. chapter 4.7 Push Notifications).

```xml
<phone:PhoneApplicationPage
    x:Class="ReceiveRaw.MainPage"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:phone="clr-namespace:Microsoft.Phone.Controls;assembly=Microsoft.Phone"
    xmlns:shell="clr-namespace:Microsoft.Phone.Shell;assembly=Microsoft.Phone"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    mc:Ignorable="d" d:DesignWidth="480" d:DesignHeight="768"
    FontFamily="{StaticResource PhoneFontFamilyNormal}"
    FontSize="{StaticResource PhoneFontSizeNormal}"
    Foreground="{StaticResource PhoneForegroundBrush}"
    SupportedOrientations="Portrait" Orientation="Portrait"
    shell:SystemTray.IsVisible="True">

    <!--LayoutRoot is the root grid where all page content is placed-->
    <Grid x:Name="LayoutRoot" Background="Transparent">
        <Grid.RowDefinitions>
            <RowDefinition Height="Auto"/>
            <RowDefinition Height="*"/>
        </Grid.RowDefinitions>

        <!--TitlePanel contains the name of the application and page title-->
        <StackPanel x:Name="TitlePanel" Grid.Row="0" Margin="12,17,0,28">
            <TextBlock x:Name="ApplicationTitle" Text="MY APPLICATION"
                       Style="{StaticResource PhoneTextNormalStyle}"/>
            <TextBlock x:Name="PageTitle" Text="page name" Margin="9,-7,0,0"
                       Style="{StaticResource PhoneTextTitle1Style}"/>
        </StackPanel>

        <!--ContentPanel - place additional content here-->
        <Grid x:Name="ContentPanel" Grid.Row="1" Margin="12,0,12,0">
            <ListBox Margin="26,26,37,222" Name="lbStudents" >
                <ListBox.ItemTemplate>
                    <DataTemplate>
                        <StackPanel>
                            <TextBlock Text="{Binding FirstName}" />
                            <TextBlock Text="{Binding LastName}" />
                            <TextBlock Text="{Binding StudentNumber}" Padding="0,0,0,10" />
                        </StackPanel>
                    </DataTemplate>
                </ListBox.ItemTemplate>
            </ListBox>
        </Grid>
    </Grid>

</phone:PhoneApplicationPage>
```
Listing 76: Project "ReceiveRaw" - MainPage.xaml

```csharp
using System;
using System.Collections.Generic;
using System.Linq;
using System.Net;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Animation;
using System.Windows.Shapes;
using Microsoft.Phone.Controls;
using System.Diagnostics;
using Microsoft.Phone.Notification;
using System.IO;
using System.IO.IsolatedStorage;
using System.Xml.Serialization;

namespace ReceiveRaw
{
    public partial class MainPage : PhoneApplicationPage
    {
        string channelName = "RawNotification";
        HttpNotificationChannel channel;
        private List<Student> student;

        // Constructor
        public MainPage()
        {
            InitializeComponent();

            student = new List<Student>();
            channel = HttpNotificationChannel.Find(channelName);

            if (channel == null)
            {
                channel = new HttpNotificationChannel(channelName);
                channel.ChannelUriUpdated += channel_ChannelUriUpdated;
                channel.ErrorOccurred += channel_ErrorOccured;
                channel.HttpNotificationReceived += channel_HttpNotificationReceived;
                channel.Open();

                channel.BindToShellTile();
            }
            else
            {
                channel.ChannelUriUpdated += channel_ChannelUriUpdated;
                channel.ErrorOccurred += channel_ErrorOccured;
                channel.HttpNotificationReceived += channel_HttpNotificationReceived;
            }
        }

        void channel_ErrorOccured(object sender,
                NotificationChannelErrorEventArgs e)
        {
            switch (e.ErrorType)
            {
                case ChannelErrorType.ChannelOpenFailed:
                    break;
            }
        }

        void channel_ChannelUriUpdated(object sender,
            NotificationChannelUriEventArgs e)
        {
            Debug.WriteLine(e.ChannelUri.ToString());
        }

        void channel_HttpNotificationReceived(object sender, HttpNotificationEventArgs e)
        {
            string message;
            using (StreamReader reader = new StreamReader(e.Notification.Body))
            {
                XmlSerializer xs = new XmlSerializer(typeof(List<Student>));
                student = (List<Student>)xs.Deserialize(reader);
                message = reader.ReadToEnd();
            }
            //Dispatcher.BeginInvoke(() => MessageBox.Show(message));
            Dispatcher.BeginInvoke(() => lbStudents.ItemsSource = student);
        }

        public class Student
        {
            public long StudentNumber { get; set; }
            public string FirstName { get; set; }
            public string LastName { get; set; }
        }
    }
}
```
Listing 77: Project "ReceiveRaw" - MainPage.xaml.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Net;
using System.IO;

namespace SendRaw
{
    class Program
    {
        static void Main(string[] args)
        {
            string url = "http://db3.notify.live.net/throttledthirdparty/01.00/AAEQxeTQY0" +
                "T5R42SYg62O3a8AgAAAAADAQAAAAQUZm52OjIzOEQ2NDJDRkI5MEVFMEQ";

            string data = "<?xml version=\"1.0\" encoding=\"utf-8\"?>" +
                "<ArrayOfStudent>" +
                  "<Student>" +
                    "<StudentNumber>123456</StudentNumber>" +
                    "<FirstName>Jane</FirstName>" +
                    "<LastName>Doe</LastName>" +
                  "</Student>" +
                  "<Student>" +
                    "<StudentNumber>987654</StudentNumber>" +
                    "<FirstName>Jane</FirstName>" +
                    "<LastName>Doe</LastName>" +
                  "</Student>" +
                "</ArrayOfStudent>";

            WebRequest request = WebRequest.Create(url);

            request.ContentType = "text/xml";
            request.Method = "POST";
            request.Headers.Add("X-NotificationClass", "3");

            byte[] bytes = Encoding.UTF8.GetBytes(data);

            using (Stream requestStream = request.GetRequestStream())
            {
                requestStream.Write(bytes, 0, bytes.Length);
                try
                {
                    HttpWebResponse response = (HttpWebResponse)request.GetResponse();
                    string notificationStatus = response.Headers["X-NotificationStatus"];
                    string subscriptionStatus = response.Headers["X-SubscriptionStatus"];
                    string connectionStatus = response.Headers["X-DeviceConnectionStatus"];

                    Console.Write(String.Format("X-NotificationStatus:{0}\r\nX-" +
                        "SubscriptionStatus:{1}\r\nX-DeviceConnectionStatus:{2}",
                        notificationStatus, subscriptionStatus, connectionStatus));
                    Console.ReadKey();
                }
                catch (WebException ex)
                {
                    Console.Write("WebExeption occured with Statuscode:" +
                        ((HttpWebResponse)ex.Response).StatusCode.ToString());
                    Console.ReadKey();
                }
            }
        }
    }
}
```
Listing 78: Project "SendRaw" - Program.cs

Appendix IV

Listing 79 and Listing 80 show the complete code the test cases in chapter 5.2 NUnit for Windows Phone.

```
using System;
using System.Net;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Documents;
using System.Windows.Ink;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Animation;
using System.Windows.Shapes;

namespace TestSamples
{
    public class Calc
    {
        public Calc()
        {
        }

        public int Add(int parameter1, int parameter2)
        {
            return parameter1 - parameter2;
        }

        public double Div(double parameter1, double parameter2)
        {
            return parameter1 / parameter2;
        }

        public bool ResultIsTrue()
        {
            return true;
        }

        public int ReturnValue()
        {
            return 3;
        }

    }

    public class ObjektA
    {
        public ObjektA()
        {
        }
    }
}
```

Listing 79: Project "TestSamples" - Calc.cs

```
using System;
using System.Net;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Documents;
using System.Windows.Ink;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Animation;
using System.Windows.Shapes;
using NUnit.Framework;
```

```
namespace TestSamples
{
    [TestFixture]
    public class CalcTest
    {
        public CalcTest()
        {
        }

        //Comparison Tests
        [Test]
        public void TestAdd()
        {
            Calc c = new Calc();
            Assert.AreEqual(30, c.Add(10, 20));
        }

        [Test]
        public void TestDiv()
        {
            Calc c = new Calc();
            Assert.AreEqual(5, c.Div(10,2));
        }

        [Test]
        public void ObjektTest()
        {
            ObjektA a = new ObjektA();
            ObjektA c = new ObjektA();
            a = c;
            Assert.AreSame(c, a);
        }

        //Condition Test
        [Test]
        public void TestResultIsTrue()
        {
            Calc c = new Calc();
            Assert.IsTrue(c.ResultIsTrue());
        }

        //Utility Method
        [Test]
        public void TestFail()
        {
            Calc c = new Calc();
            if (c.ReturnValue() > 2)
            {
                Assert.Fail("Fehler!");
            }
        }
    }
}
```
Listing 80: Project "TestSamples" - CalcTest.cs