VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

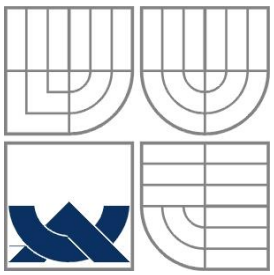
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

APPLICATION FOR CONTROLLING INELS INTELLIGENT ELECTRICAL-INSTALLATION FOR THE WINDOWS PHONE PLATFORM

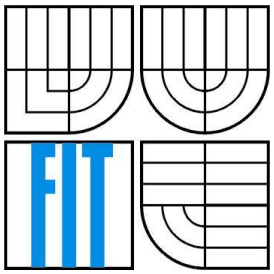
DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE Bc. David Bednář
AUTHOR

BRNO 2013



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

APLIKACE PRO OVLÁDÁNÍ INTELIGENTNÍ ELEKTRO-INSTALACE INELS PRO PLATFORMU WINDOWS PHONE

APPLICATION FOR CONTROLLING INELS INTELLIGENT ELECTRICAL-INSTALLATION FOR THE WINDOWS PHONE PLATFORM

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE Bc. David Bednář
AUTHOR

VEDOUCÍ PRÁCE doc. Ing., Dipl.-Ing. Martin Drahanský, Ph.D.
SUPERVISOR

BRNO 2013

Abstrakt

Tato práce se zabývá inteligentní elektroinstalací iNELS od společnosti ELKO EP a jejím ovládáním pomocí zařízení využívajících platformu Windows Phone. V textu je popsána komunikace, kterou využívá systém iNELS a multimediální systém iMM, především pak protokoly EPSNET a XML-RPC. Dále je popsána platforma Windows Phone a implementace samotné aplikace. V závěrečné části se práce zabývá zajímavými rozšířeními dané aplikace.

Abstract

This thesis talks about iNELS Intelligent Electrical-Installation, which is developed by ELKO EP, and its controlling using Windows Phone based devices. We describe communication, which is used within the iNELS system and the iMM multimedia system, especially focusing on the EPSNET and XML-RPC protocols. Windows Phone platform is discussed and the implemented application is presented. The last section talks about interesting extensions to the application.

Klíčová slova

Domácí automatizace, inteligentní budovy, iNELS, ELKO EP, protokol EPSNET, XML-RPC, Windows Phone

Keywords

Home automation, intelligent buildings, iNELS, ELKO EP, EPSNET protocol, XML-RPC, Windows Phone

Citace

Bednář David: Application for Controlling iNELS Intelligent Electrical-Installation for the Windows Phone Platform, diplomová práce, Brno, FIT VUT v Brně, 2013

Application for Controlling iNELS Intelligent Electrical-Installation for the Windows Phone Platform

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením doc. Ing., Dipl.-Ing. Martina Drahanského, Ph.D.
Další informace mi poskytli pan Jiří Konečný, Michal Mrnuštík, Michal Richter a Jakub Hrádek ze společnosti ELKO EP.
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
David Bednář
20. 5. 2013

Poděkování

Rád bych poděkoval svému vedoucímu diplomové práce doc. Ing., Dipl.-Ing. Martinu Drahanskému, Ph.D. za odborné vedení, připomínky a cenné rady, které mi během tvorby této práce poskytl.
Dále bych rád poděkoval za vedení a rady od společnosti ELKO EP, především panu Jiřímu Konečnému, Michalu Mrnuštíkovi, Michalu Richterovi a Jakubu Hrádkovi.

Table of contents

1 Introduction

Nowadays, intelligent buildings are no longer a domain of large commercial building complexes, but are becoming more important also for smaller family houses. Home automation puts together many previously separated systems such as light and heating control, air conditioning, security and access control and most recently also multimedia systems and household appliances such as washing machines and dishwashers.

As these systems grow in complexity and more devices and various systems need to be controlled, common switches and remote controllers are no longer effective and comfortable enough to control these complex and heterogeneous systems. As a result, controllers and switches were replaced by expensive in-wall touch screens and sophisticated remote controllers with programmable interface. However, these can be easily substituted by today's powerful smart phones and tablets. These devices have a great advantage of being already familiar to users from their daily activities. Also, most of available devices are based on modern mobile operating systems, which provide a foundation stone for an all-featuring home control application. In addition, modern devices also provide wide range of sensors and other equipment that can be used to further extend capabilities of intelligent buildings and their interaction towards residents. Lastly, thanks to these devices, features such as voice control or motion control can be implemented with minimal costs.

The aim of this thesis is to describe development of a home control application, which would be able to control the *iNELS* electrical installation system developed by *ELKO EP*, which is one of the largest producers of home automation systems in the Czech Republic as well as in Europe [18]. The platform, which has been selected for this project is the *Windows Phone 8* operating system produced by *Microsoft*. This system was chosen for its innovative approach, qualities, future potential and great development tools and support. Also, this platform shares components with the *Windows 8* operating system, which should support potential porting of the application to this platform as well.

In this project, we will firstly focus on the *iNELS* and *iMM* systems and their communication protocols. We will also describe important configuration files used within these systems. This section will benefit from the semestral project, which was preceding this thesis.

In the next section, the *Windows Phone 8* platform will be described. We will especially be looking into its features and APIs, which we can benefit from in our home control application, but also general concepts will be discussed.

Lastly, the design and implementation of the application will be presented. Approaches used within the application will be described and also description of implemented extensions and future development will be provided.

2 Intelligent buildings

Should there be a list of things, which have the largest number of different definitions, intelligent building, smart homes or home automation systems would be on that list. Over last decades, when those systems were formed and have grown in popularity, each researcher or producer has created its own definitions. For the purpose of our project, we will use a definition from [28]:

"An Intelligent Building is a building that integrates technology and process to create a facility that is safer, more comfortable and productive for its occupants, and more operationally efficient for its owners. Advanced technology—combined with improved processes for design, construction and operations—provide a superior indoor environment that improves occupant comfort and productivity while reducing energy consumption and operations staffing."

Using this definition, reasons to build an intelligent building rather than a regular one are obvious. Their occupants will have more comfort and most importantly, the operating costs and maintenance costs will be lower. Even though intelligent buildings might be much more expensive to build, the investment usually pays back thanks to the efficient energy saving mechanism and as mentioned before, also much lower overall subsequent costs. Nowadays, most of new commercial buildings are built with the intelligent building paradigm in mind.

Figure 2.1 - Burj Khalifa, Taipei 101 and Petronas Towers [27].

2.1 Evolution of intelligent buildings

The first generation of intelligent buildings dates back to the 1980s. Buildings used dedicated systems that only controlled single functionality of the building such as lighting or access control. These systems were not centrally connected and usually did not communicate with each other. [10]

The next generations provided a network connection between different systems, which enabled remote control functionality and cooperation of multiple subsystems. Buildings from 1990s were able to react on changing demands of the occupants and basically featured fully integrated automation systems.

Together with the evolution of information technology and communication technology, intelligent buildings became aware of its occupants. With the use of artificial intelligence and machine learning, it was possible for the buildings to automatically adjust individual subsystems based on the behavior of occupants. Also all systems became integrated within single one, which was not only able to control all automation subsystems, but also all network communication.

Sometimes, however, intelligent buildings can also have negative impact on its occupants. Due to very complex integration of all systems, even small changes can have a huge impact on the system. For example, accidentally opened windows could change the inner climate in a way that the system could not compensate. Therefore occupants' freedom is being limited and controlled, which might be discomforting. [10]

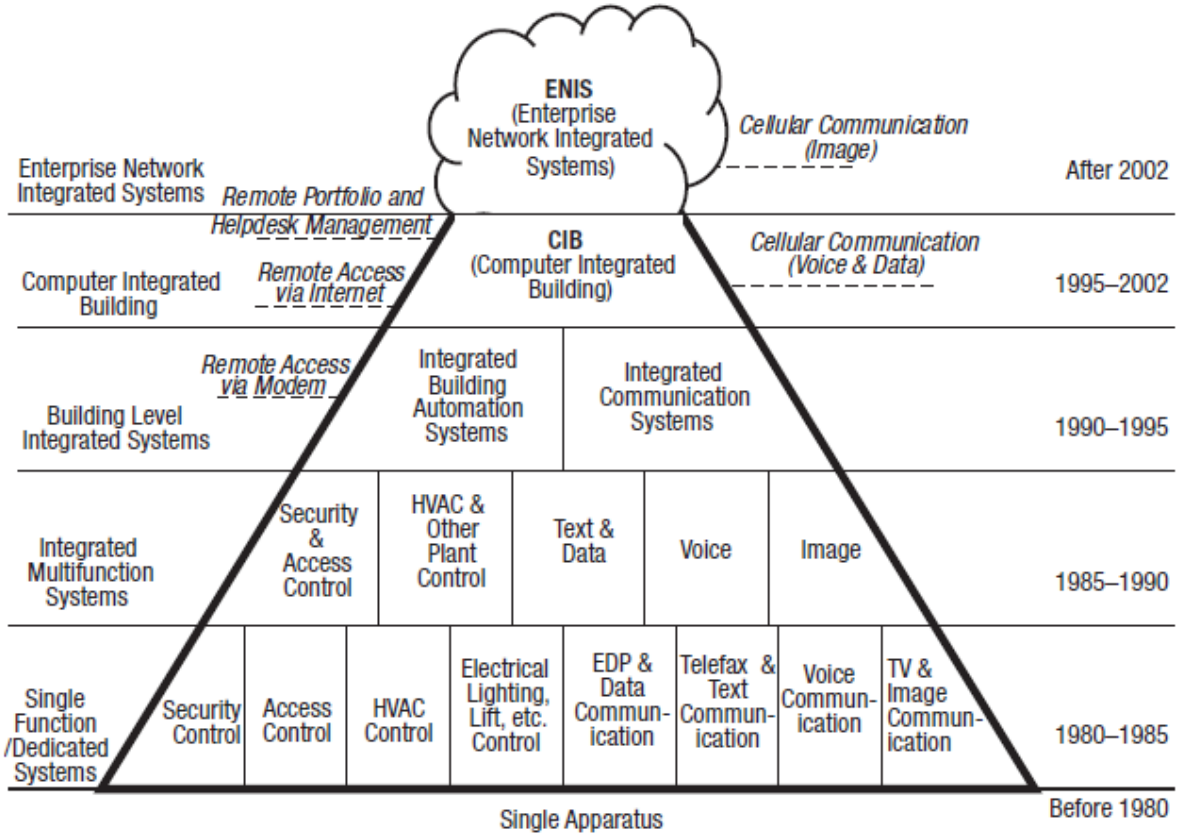


Figure 2.2 - Intelligent building pyramid [8].

2.2 Subsystems

It would be very difficult to describe all features and subsystems of today's intelligent buildings and home automation systems and not to forget any. We will therefore only briefly summarize basic subsystems that are usually present in most intelligent buildings.

Lighting

Lighting is one of the basic subsystems, which is present in almost every configuration. Where possible, the sunshine is effectively used and its amount is controlled by blinds. The aim is to use as little energy as possible, so when no one is present in an office lights are usually turned off.

Climate

Sustaining stable climate within the building is also very important for energy saving. These subsystems take care of the climate as whole, i.e. they control not only temperature, but also humidity, fresh air and gas and particle concentration. The climate subsystem is therefore very complex, as it must control not only heating, venting and air conditioning, but also has to take into account all the windows and doors, where heat exchange occurs, control blinds and communicate with other subsystems.

Security and access control

Security has been one of the most important systems even before the era of intelligent buildings. It features smoke and fire detectors, heat sensors and movement detectors, alarms and connectivity to security and rescue services. Nowadays is also monitors movement of occupants and property and features camera systems.

Maintenance

This subsystem is able to quickly recognize if some parts of the systems are not working properly. It identifies not only the exact location of the error, but usually also the probable cause. This helps the staff to remove the malfunction very quickly and helps maintaining stability and overall health of all systems.

Elevators

Especially in large commercial buildings, elevators can consume a significant part of energy resources. Modern elevator-monitoring systems try to maximize the number of transported occupants, maximize the comfort while being transported and at the same time, minimize costs and waiting times. This is usually achieved by monitoring not only buttons pressed by the occupants, but also, with cooperation with camera monitoring systems, obtain more precise numbers of the occupants.

Multimedia

Multimedia subsystems tend to centralize all media resources to one server. This helps save the energy and lower costs. Clients can then connect to the server and stream audio or video from various sources such as the internet, cable or satellite television to any device within the building. Such device can be a television, audio system, personal computer or mobile devices.

Appliance control

As many modern home appliances such as washing machines, coffee machines and dishwashers also feature network connection, they can also be controlled by the intelligent building. Washing machines can, for example because of the noise, be started when nobody is home, or during the night because of energy prices. Also, monitoring of the state of those appliances can greatly contribute to the comfort of occupants.

Communication

Modern buildings also use centralized communication control. Occupants can accept calls in different places, even using different devices. The system will take care of proper redirection. Also, costs can be reduced, as all calls can be directed through one connection, or even over the internet. Devices connected to these subsystems can be phones, computers or house bells.

Energy harvesting

As the one of the main purpose is to reduce costs and energy consumption, energy harvesting also plays a significant role. Firstly, energy monitoring subsystems provide information about excessive use of energy sources and can identify many problems. Nowadays, many buildings also feature photovoltaic panels, which can reduce energy costs, especially in sunny locations. Automatically adjusting solar panels can also increase the energy obtained.

Watering

Watering subsystems are vital in dry areas, where water resources are limited. These subsystems work autonomously and by monitoring the weather condition, or even weather outlook, watering during night and not during windy times, they can preserve large amounts of water supplies.

2.3 Categories

As there are many types of home automation systems available, we will use categories as mentioned in [10]. We will not mention custom solutions such as those described within [9], as these are not subject of our interest in this project.

Open systems

The systems are built upon publicly available standards and their specifications are available. The advantages of such system are that many manufactures can provide accessories and also academic researchers can help develop new functionality. This pushes prices lower and community can help with further development. On the other hand, the market can be fragmented and choosing best alternatives can be time consuming. *KNX*, *Lon*, or *BACnet* are a great example of such systems. [10]

Closed systems

Closed systems are usually produced by a single company and specifications are not available. On one hand, users are limited to use only devices and subsystems, which are provided by this company, on the other hand, configuration of those devices tend to be much easier and service is usually available. These systems are represented by *ABB Ego-n*, *iNELS* by *ELKO EP* or *Moeller Xcomfort*. [10]

Centralized architecture

In centralized architectures, only one or a small number of central units is controlling whole system. Parts of the system are connected directly to the central unit, or via a bus. Architectures without a bus are have the disadvantage of having long and costly wires for each separate device, while bus-based architectures can only use one or two wires for all devices. Centralized architecture does not require intelligent sensors; however, whole system depends on the central unit. *ABB Ego-n* uses such architecture. [10]

Decentralized architecture

In a decentralized architecture, the control is distributed across the whole building. All devices include some intelligence and are connected to a central bus. Since intelligence is distributed, the system tends to be more robust. These systems are represented by *LON* or *KNX* systems. [10]

2.4 iNELS

The *iNELS Intelligent Electrical-Installation* is a home automation system for intelligent buildings developed by *ELKO EP, s. r. o.*, which provides the possibility to control whole house or commercial building using centralized structure of control units. The *iNELS* system is able to control many aspects of an intelligent building, including light control, heating and air conditioning, security and alarms, window blinds, other electrical devices such as washing machines and dishwashers. With the use of *iNELS Multimedia extension (iMM)* the system can control also various multimedia devices such as televisions, audio systems and others.

The main part of the centralized structure is a central *PLC unit CU2-01M*, which is responsible for controlling all attached devices through *CIB (Common installation bus).* Attached devices are connected by two wires that form the bus. This is a big advantage over other common systems that sometimes use dedicated wires for each device or use a separate bus and power wires. The control unit is able to connect to up to 196 devices using extension modules and also connects to a local network using Ethernet port. [19]

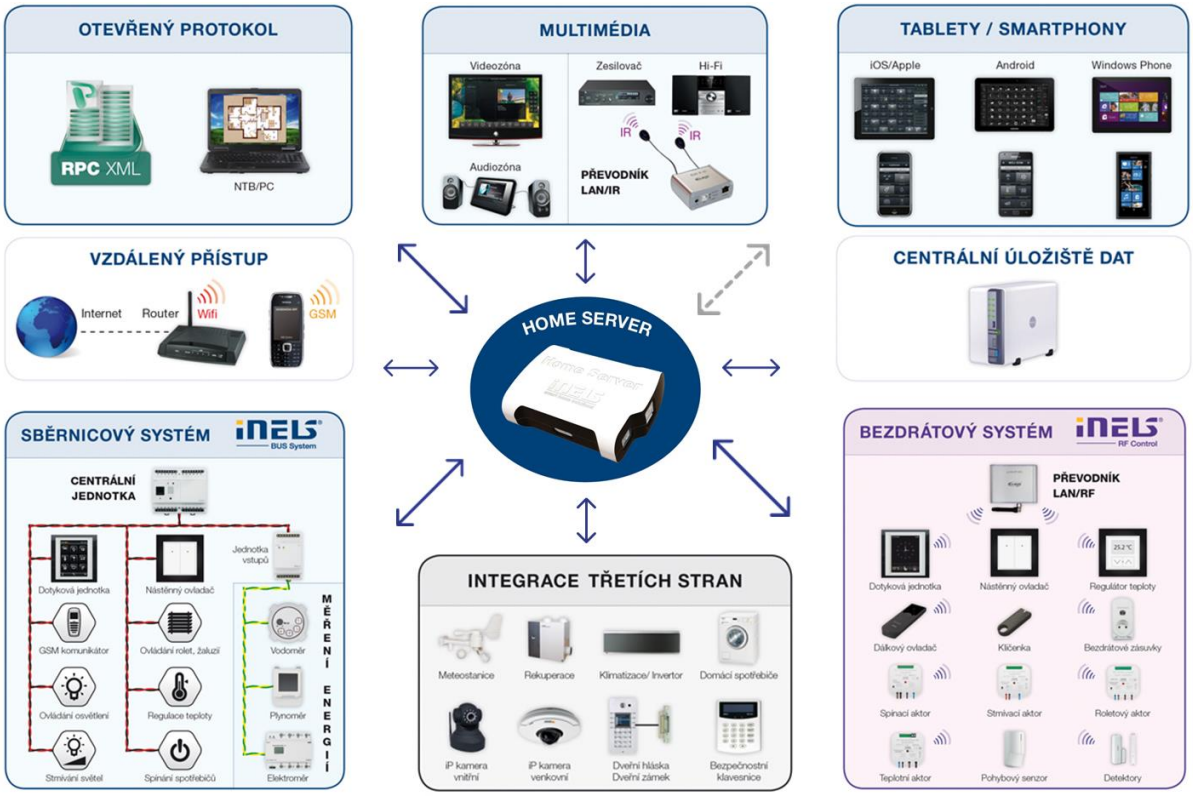


Figure 2.3 – The iNELS home control system [19].

2.4.1 EPSNET protocol

The *iNELS* system can communicate via Ethernet port with remote devices, such as remote control systems, using the *EPSNET protocol* [25], which is implemented over *UDP*. *UDP* is suitable for its simplicity and fast communication setup, but does not provide the certainty of delivered messages.

The *PLC* can communicate in various modes – *PC, PLC, UNI or MDB*. In this work, we will make use of the *PC* mode running on port 61682, which is used for basic communication. The structure of *UDP* packet is fixed and consists of a six-byte header and one ore up to five *EPSNET* messages, whose structure we will describe later. The header includes:

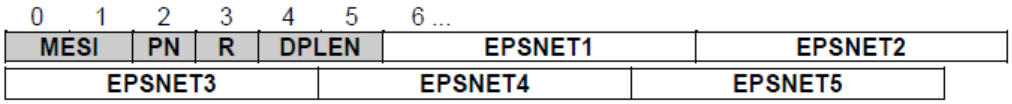


Figure 2.4 - EPSNET UDP packet structure [25].

MESI (byte 0 and 1) – number identifying the message, which is same for in the response

PN (byte 2) – code specifying whether PC (2) or PLC (3) configuration is used

R (byte 3) – reserved

DPLEN (byte 4 and 5) – length of all following data, where byte 5 contains the low part

The length must be even.

The *EPSNET* protocol specifies two kinds of devices – master and slave – and two types of basic configuration – *monomaster* (with only one master and multiple slaves present within the network) and *multimaster* (with multiple master and slave devices within the network).

The structure of each *EPSNET* message can have a variable length, so there the data are protected by checking the sequence of values, even parity (can be turned off) and by a checksum stored in *FCS* field. If these protections are not met, the message is discarded. There are also rules for silence on the line before sending additional messages, which are discussed in details in [25].

One of the common structures of the *EPSNET* message is with *DATA* field as follows:

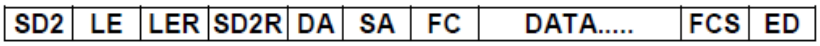


Figure 2.5 – EPSNET message structure example [25].

SD1 – start delimiter 1 ($10)

SD2 – start delimiter 2 ($68)

SD4 – start delimiter 4 ($DC)

LE – length of (DA + SA + FC + DATA) = (3 … 249)

LER – length repeat

SD2R – start delimiter 2 repeat

DA – destination address (0 … 126)

SA – source address (0 … 126)

FC – frame control byte – specific for each type of message

DATA – data specific for each type of message

FCS – frame check sum – byte sum of DA, SA, FC and DATA with neglect of overflow

ED – end delimiter ($16)

SAC – short acknowledge ($E5)

Where $XX stands for a number in hexadecimal formatting.

2.4.2 EPSNET communication services

The EPSNET network provides a set of communication services divided into two groups – system communication services and public communication services. In this section, we will focus on a subset of the public communication services that are essential for this project. This section is based on implementation details from [25].

CONNECT

The beginning of the communication, bound to communication structures initialization.

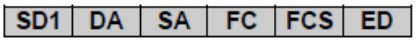


Figure 2.6 – The structure of CONNECT message [25].

FC = $49 or $69

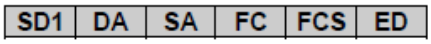


Figure 2.7 – The structure of CONNECT reply [25].

FC = $00

IDENT

Service used for gathering data and information about the connected system.

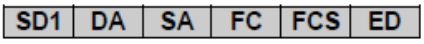


Figure 2.8 – The structure of IDENT message [25].

FC = $4E or $6E

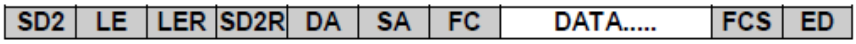


Figure 2.9 – The structure of IDENT reply [25].

FC = $00

DATA includes lengths and values of the identification string of the central unit, implementation protocol sign, structure version string and software version string.

GETSW

This service reads the status word.

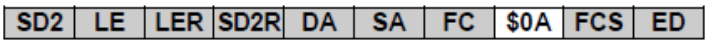


Figure 2.10 – The structure of GETSW message [25].

FC = $4C or $6C

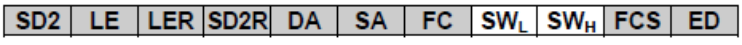


Figure 2.11 – The structure of GETSW reply [25].

FC = $08

SW fields contain low and high byte of the status word, which includes indications of errors and system configuration.

READN

Service is used for reading data from the register memory.

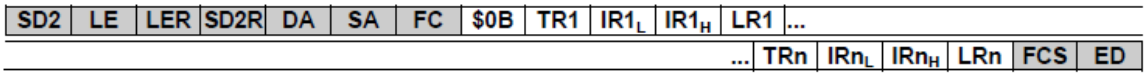


Figure 2.12 – The structure of READN message [25].

FC = $4C or $6C

TR defines the register, from which the data should be read.

$00 for X registers

$01 for Y registers

$02 for S registers

$03 – $05 for R registers

$80 and above for DataBox memory

IR fields stand for low and high byte of the first read register index.

LR is the count of read registers.

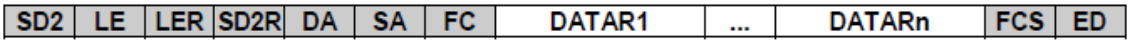


Figure 2.13 – The structure of READN reply [25].

FC = $08

DATAR fields contain the read values.

WRITEN

This service is used for writing data into the register memory.

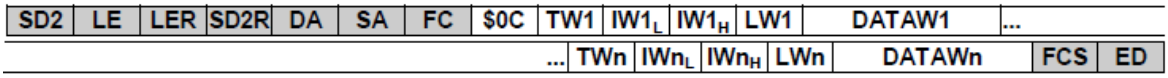


Figure 2.14 – The structure of WRITEN message [25].

FC = $43 or $63

TW, IW, LW and DATAW fields correspond to the TR, IR, LR and DATAR fields of the READN service.

The WRITEN reply message only consists of SAC.

READB

Service used for reading single bits from register memory.

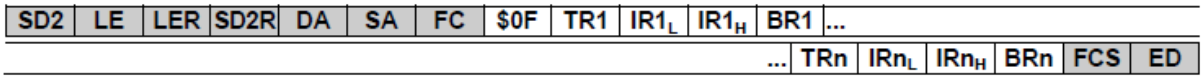


Figure 2.15 – The structure of READB message [25].

FC = $4C or $6C

TR and IR fields correspond to the TR and IR fields of the READN service.

BR fields include the index of the read bit (0 up to 7).

Figure 2.16 – The structure of READB reply [25].

FC = $08

BITR fields contain the values of read bits converted to bytes according to the pattern:

0 = $00

1 = $FF

WRITEB

Service used to write bits into register memory.

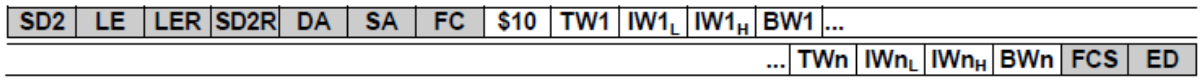


Figure 2.17 – The structure of WRITEB message [25].

FC = $43 or $63

TW and IW fields correspond to the TR and IR fields of the READN service.

BW fields stand for the index of the written bit and its value:

For writing value 0 – $00 up to $07 (index 0 to 7 within the register byte)

For writing value 1 – $80 up to $87 (index 0 to 7 within the register byte)

The WRITEB reply message only consists of SAC.

Other public or system services are not implemented in this project as they are not needed for its functionality. These include services for destructive read and write operations (READBD, READND and WANDRND) and others.

2.4.3 The iMM multimedia extension

The *iNELS Multimedia extension* provides even more functionality to *iNELS* systems. It is based on a linux machine, which is running the actual *iMM* server. This server can directly communicate with the *iNELS* central unit and control attached devices, but also provides many multimedia functions. Among these, the *iMM* can serve as a central storage for music and video files; it provides the possibility to control *iNELS* system using a TV screen and a special remote controller. Further, one satellite receiver can be shared for all zones. It supports IP cameras, including the record and remote control features. Also third party devices can be controlled, such as *Miele* [26] devices or air conditioning and energy consumption can be monitored. Other PCs or mobile phones can than connect to the *iMM* server using *XML-RPC* protocol and control other devices using the procedures offered by the server.

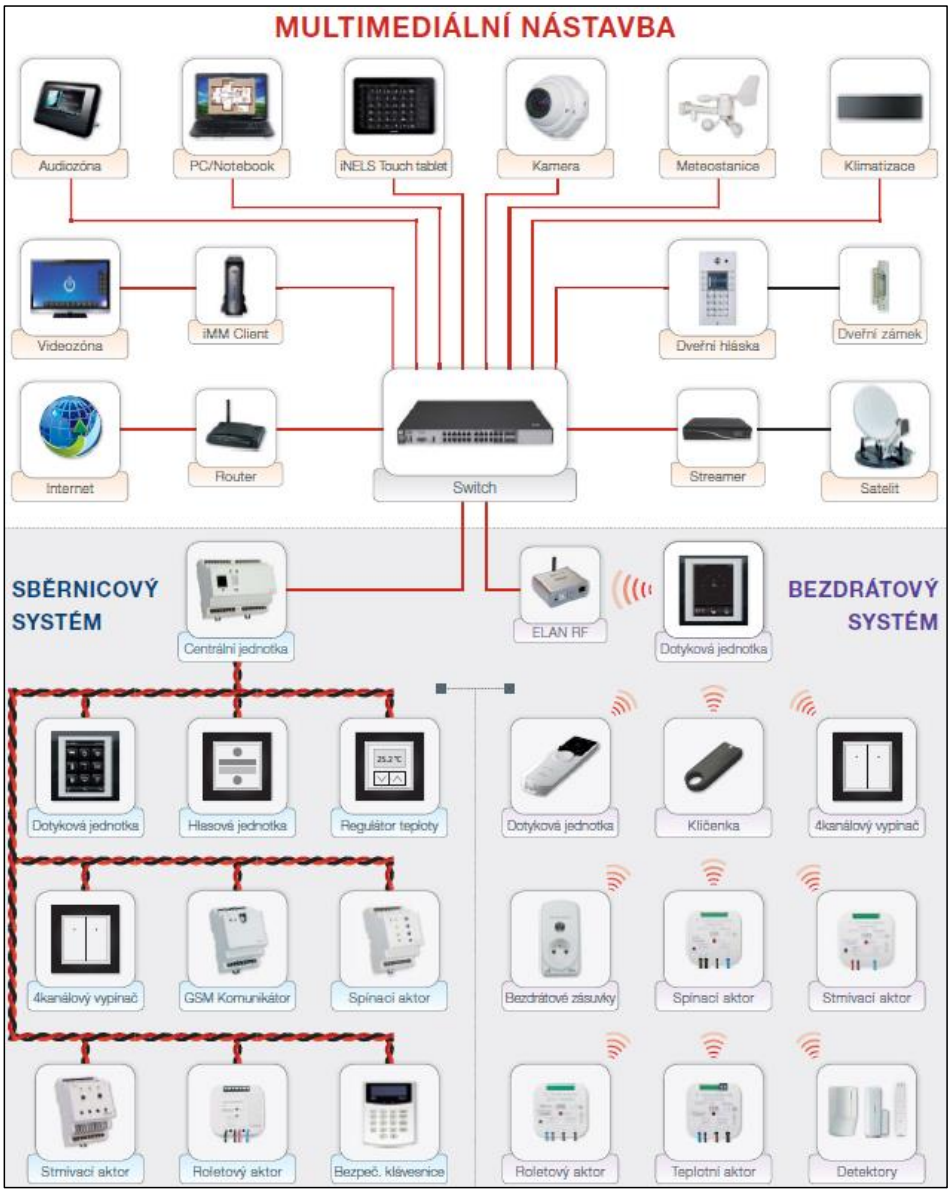


Figure 2.18 – The iMM multimedia extension [].

2.4.4 The iMM server and XML-RPC protocol

For the communication between *iMM* server and other clients, the *XML-RPC* [14] protocol is used. This protocol is actually a set of rules and implementations that allow software written in various languages and various operating systems to perform remote procedure calls over the Internet. The protocol uses *HTTP* as the transport protocol and *XML* for encoding the messages. This protocol should be as simple as possible, but at the same time should allow transfers of very complex data structures.

The *iMM* server provides several methods that can be used by the client devices. The most basic is the ping method, which is called without parameters and return True value if connection is successful. For obtaining the state of *iNELS* devices, the read method is used. This method accepts array of string values representing names of required devices and returns pairs of values representing the name of the device and its value. For setting values for devices, the *writeValues* method is used, supplying an object with key-value pairs, where key is the device's name and value is the value to be set, as a parameter.

Probably the richest method set is provided for multimedia. These include methods such as *getPlayersList, getVolume, setVolume, playIfPaused, pause, stop, jumpFf, jumpRw, repeat, shuffle* and others. These methods usually require IP address of the station as a parameter and for certain methods, also the type of station is required. The type of station is represented by a number:

0 - Audio client

1 - Audio squeezebox

2 - Video client

3 - Photo client

Other method sets include methods for energy manager readings such as the *eManTotalSumsAndPrices, eManTodaySumsAndPrices* or *eManWeekSumsAndPrices*. These methods are discussed within the extensions of our application in the Energy manager section. The rest of methods include methods for obtaining configuration data, *Miele* device information and methods for working with IP cameras.

For our project, we will, use the *XML-RPC.NET library* [14]which is described later in the text.

2.4.5 Public server and configuration

The *iNELS* system can be configured using two ways. If there is an *iMM* server present within the network, it can be configured by supplying the configuration files and editing them. However, if *iMM* server is not present within the configuration, public server can be used for configuration. These servers provide a web-based interface, which enables the user to modify both configuration files, which are discussed in the next section. The configuration consists of rooms, which contain certain devices. A room can, for example, include various lights and lamps, blinds and other devices.

For the multimedia extension, *zones* are introduced. Each *zone* represents a single multimedia device such as a television or audio player. Multiple *zones* can be assigned to a room. Air conditioning and energy modules can also be present based on the configuration.

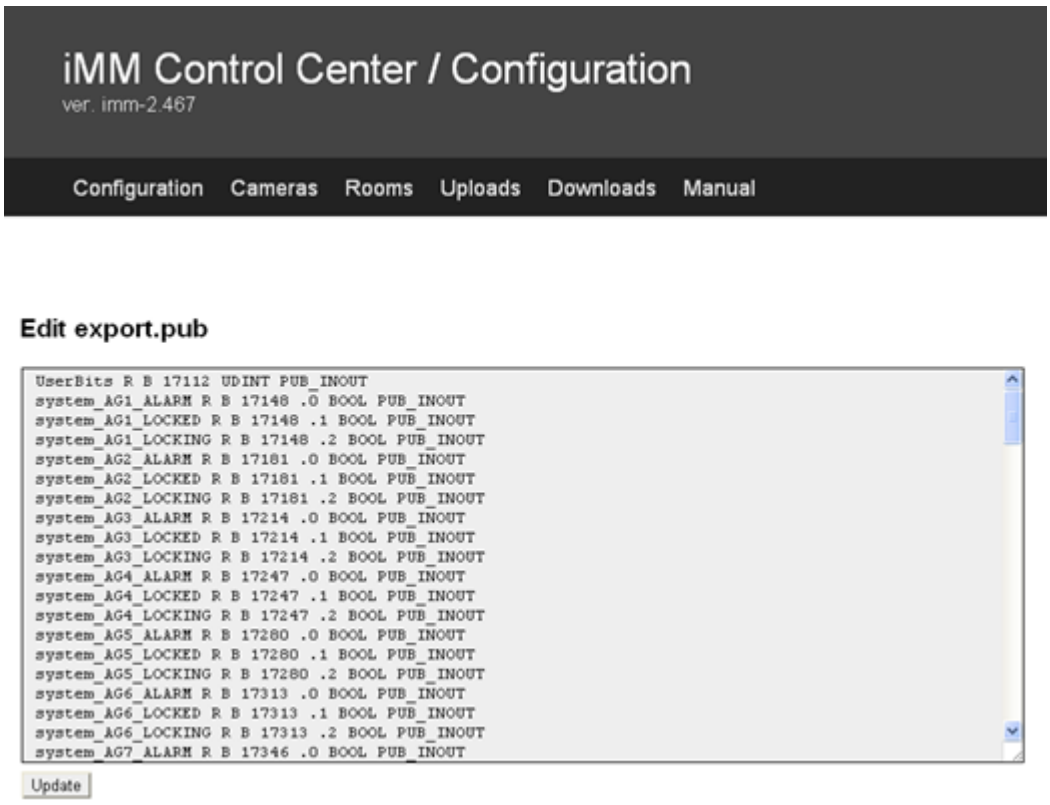


Figure 2.19 – The iMM Control Center configuration interface [19].

Export.pub

The *export.pub* configuration file contains definitions for all *iNELS* devices, which are present. Each device can be present in multiple forms, because its actions can be included as well. However, in our application, we will only use devices defined without any suffixes such as *_ON*, *_OFF* or *_TRIG*. The structure of each line is fixed and is described below.

Inels_item	REG	CF	ADDR	.B	TYPE	PUB_INOUT

Figure 2.20 – Formatting of export.pub.

Inels_item – item name generated by IDM

REG – defines type of register (X, Y, S, R values)

CF – compatibility field (B, F values)

ADDR – address of the register

.B – position of the bit within register for values of type BOOL

TYPE – type of the variable (REAL, BOOL, BYTE, UINT, UDINT, …)

PUB_INOUT – specifies input or output (PUB_IN, PUB_OUT, PUB_INOUT values)

```
da22_rs_stmivana_zasuvka_lampa_ON R B 18840 .1 BOOL PUB_INOUT
da22_rs_stmivana_zasuvka_lampa_OFF R B 18840 .2 BOOL PUB_INOUT
da22_rs_stmivana_zasuvka_lampa R F 18879 REAL PUB_INOUT
state_da22_rs_stmivane_osvetleni_halogeny Y F 12 REAL PUB_OUT
da22_rs_stmivane_osvetleni_halogeny_ON R B 18923 .1 BOOL PUB_INOUT
da22_rs_stmivane_osvetleni_halogeny_OFF R B 18923 .2 BOOL PUB_INOUT
da22_rs_stmivane_osvetleni_halogeny R F 18962 REAL PUB_INOUT
da22_rs_TERM X F 7 REAL PUB_INOUT
sa04_rs_1_SW1 X B 11 .0 BOOL PUB_INOUT
sa04_rs_1_SW2 X B 11 .1 BOOL PUB_INOUT
sa04_rs_1_SW3 X B 11 .2 BOOL PUB_INOUT
sa04_rs_1_SW4 X B 11 .3 BOOL PUB_INOUT
state_sa04_rs_1_roleta_nahoru Y B 16 .0 BOOL PUB_OUT
sa04_rs_1_roleta_nahoru_ON R B 19066 .0 BOOL PUB_INOUT
sa04_rs_1_roleta_nahoru_OFF R B 19066 .1 BOOL PUB_INOUT
sa04_rs_1_roleta_nahoru_TRIG R B 19066 .2 BOOL PUB_INOUT
sa04_rs_1_roleta_nahoru R B 19076 .0 BOOL PUB_INOUT
state_sa04_rs_1_roleta_dolu Y B 16 .1 BOOL PUB_OUT
sa04_rs_1_roleta_dolu_ON R B 19120 .0 BOOL PUB_INOUT
sa04_rs_1_roleta_dolu_OFF R B 19120 .1 BOOL PUB_INOUT
sa04_rs_1_roleta_dolu_TRIG R B 19120 .2 BOOL PUB_INOUT
sa04_rs_1_roleta_dolu R B 19130 .0 BOOL PUB_INOUT
```

Figure 2.21 - Sample of export.pub configuration file.

Rooms.cfg

The *rooms.cfg* configuration file contains structured information about *iNELS* devices, which belong to specific rooms. Each *room* element is listed under the root *rooms* element and contains the name of the room and also devices. There are various types of devices, such as *conditioning*, *garage*, *gate*, *lamps*, *lights*, *zones*, *scenes* or *shutters*. Each of these has specific attributes, but there are some that are common to all of them. The most important attribute is the *inels*, which identifies the device within *export.pub* configuration file. Each device also contains its *name*, which represents it within our application and also *column* and *row* in which it should be displayed. The rest is specific and varies across different devices.

```
<!--<?xml version="1.0" encoding="utf-8" ?>-->
<rooms>
  <room name="Room 1">
    <thermals>
      <item inels="Thermo_sensor_SA2_02B" placement="indoor">Teplota</item>
      <item inels="Thermo_sensor_WSB40" placement="outdoor">Teplota</item>
    </thermals>
    <scenes>
      <item column="0" dev_0="evnt_Zapnout_vsechny_svetla" row="5">Světla ON</item>
      <item column="1" dev_0="evnt_Vypnout_vsechny_svetla" row="5">Světla OFF</item>
      <item column="2" dev_0="Kontrolka_termohlavice_OFF" dev_1="evnt_Vypnout_vsechny_sve
    </scenes>
    <zones>
      <item audio="1" video="1">VIDEO ZONE</item>
      <item audio="1" video="0">AUDIO ZONE</item>
    </zones>
    <lights>
      <item column="0" inels="Halogenova_zarovka_1" read_only="no" row="0">Halogen 1</ite
      <item column="1" inels="Halogenova_zarovka_2" read_only="no" row="0">Halogen 2</ite
      <item column="3" inels="Zarivka" read_only="no" row="0">Zářivka</item>
    </lights>
    <lamps>
      <item column="2" inels="Svetlo_na_zdi_nalevo" read_only="no" row="0">kuchyn</item>
```

Figure 2.22 - Sample of rooms.cfg configuration file.

3 Windows Phone platform

In this chapter, we will describe the *Windows Phone* platform, focusing on the latest version of this platform – the *Windows Phone 8*. We will describe main concepts of this platform, as well as the most important features for this project. This chapter will also look into some of the provided APIs of *Windows Phone* platform and describe the development process together with publishing applications to the *Windows Phone Store*.

Windows phone platform is a relatively new mobile phone platform. It is a successor to the *Windows Mobile* platform, which was widely used between the years 2000 to 2010. *Windows Phone*, released in November 2010, is a complete makeover of the mobile platform in terms of design, structure and target user group. *Windows Phone* platform is not backward compatible with *Windows Mobile*, it is a modern mobile operating system, primarily controlled by finger touch gestures, opposite to the stylus touch input for *Windows Mobile*, and is targeted at wide public as well as enterprise market.

In October 2012, the new *Windows Phone 8* platform was introduced, again mostly restarting the ecosystem as it is not compatible with *Windows Phone 7* devices. Most of the *Windows Phone 7* applications may be converted for the new version though.

Although the *Windows Phone* platform is younger than *iOS* or *Android* platforms and there are not as many applications available [17], *Windows Phone* seems to be a progressive platform, which is especially appreciated for its speed as well as overall user experience.

3.1 Reference devices

As a reference devices, on which the application will be developed and tested, the *HTC Windows Phone 8S* and *Nokia Lumia 920* were chosen. Both these devices run *Windows Phone 8* operating system, but they were chosen due to different specifications, so that the application could be tested on different hardware. The *HTC* features 4 inch display with resolution of 800x480 pixels. It is powered by dual-core 1 GHz processor with 512 MB of RAM and has 4 GB of local storage expandable by MicroSD cards (memory cards cannot be used for installing applications as for now) [22]. The *Nokia Lumia 920* comes from the top spectrum of *Windows Phone 8* devices. It features 4.5 inch display with 1280x768 pixel resolution, has 1.5 GHz dual-core processor and 1 GB RAM. Its internal storage is 32 GB, but non-expandable [21]. The *Nokia Lumia 920* also supports *Near Field Communication (NFC)*.

Figure 3.1 – The HTC Windows Phone 8S [22] and Nokia Lumia 920 [21].

3.2 New Windows Phone features

From the users' prospective, *Windows Phone 8* adds many new features to the *Windows Phone* ecosystem. Even though most of these features are interesting, we will focus only on those that we can profit from in this project.

As first feature to come to *Windows Phone 8* is the ability to use more advanced hardware such as multicore processors. Even though this project is not aiming at implementing a demanding application, it will be useful in later stages of the application's lifecycle, where additional functionality like IP camera control will be added.

Additional feature is the *NFC (Near Field Communication)* support. We might benefit from this technology later, as it can be used to transfer the configuration data for the application. This would enable the user not to use a computer or other technology for transferring these into the device.

Other features extend the functionality of networking using *Sockets*[13]. This will also be useful in the later stages, as it permits listening and consequently establishing a connection, which was not initiated from the phone itself.

File and URI associations [13] will be useful for the configuration data as well. It provides the possibility to associate and application with specific extension. So if the configuration file is sent via email for instance, we can directly launch our application and load new *iNELS* [19] configuration.

Extended voice recognition and speech support will also be beneficial for our application's extension, as it greatly improved the way user can communicate with the application.

3.3 Architecture

Windows Phone 8 platform is no longer based on the *CE (Windows Embedded Compact)* architecture, but is based on the *Windows NT kernel*, like *Windows 8*. Also, starting with *Windows Phone 8*, the development for *Windows Phone* using *Windows Phone API* is now more flexible in terms of languages and technologies that can be used. The *Windows Phone API* consists of three parts available for development: the managed *.NET framework*, *Windows Phone runtime*, which provides the possibility to use *C++* native programming and also other low-level components such as *Direct3D* for gaming [13].

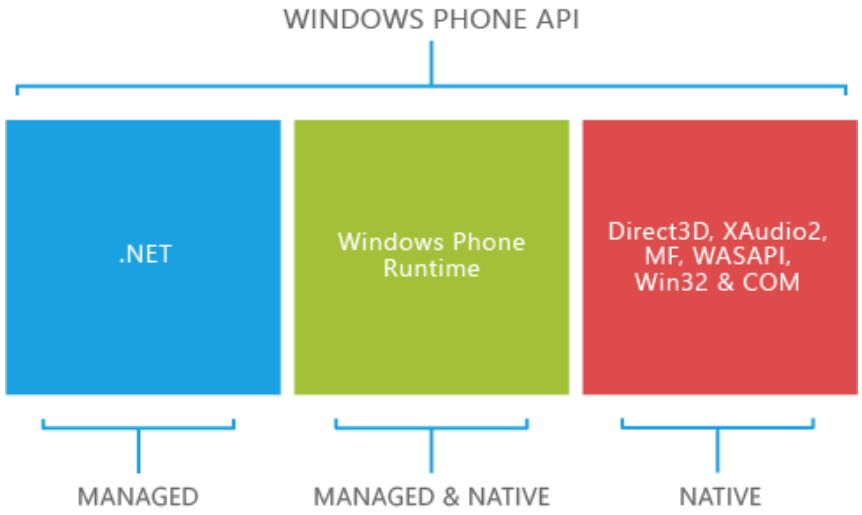


Figure 3.2 – The Windows Phone API [13].

In this project, we will make use of the managed API, as this project will be mostly based on *.NET framework* and the programming language chosen is *C#*.

3.4 Navigation

Windows Phone applications are not window-based like desktop applications. They are more like a web application, i.e. page-based. This is due to a limiting factor of the screen size and resolution and also touch –based controlling of the phone, where windows would not be user friendly and their non-content features would occupy too much space.

As the applications are page-base, the navigation through them is very similar to the one used on web pages. Upon starting, the application navigates to *App.xaml* page. This page hold application-wide resources and is bound to *App.xaml.cs* code file, which includes most of the global objects of the applications, as well as code for handling global lifecycle events of the application. [5]

Navigating between other pages can be done using *NavigationService.Navigate* method, which is supplied with the *URI (Uniform Resource Identifier)* of the target page. Parameters can also be appended to the base address, just like in web applications. The *Navigate* method will always create a new instance of the requested page. Visited pages are stored within a page stack. Every time a new page is navigated to, it is stored within the page stack and every time the back button is pressed or *NavigationService.GoBack* method is called, the top page is removed. If there are no more pages within the stack and back button is pressed, the application is navigated from. The concept of using back button to navigate back instead of using Navigate method again to the previous page is important as otherwise the page stack can be filled with circular page navigation and the user might be confused when using the back button. [3]

The back button function can be overridden and navigating to the previous page canceled by the developer. However, in order to publish the application and get it through the certification process, the function must be reasonable. For example, the back button press might be used for hiding a dialog, but not for a custom action such as taking a picture or starting calculations. [13]

3.5 Application life cycle

In *Windows Phone*, there can be multiple applications active, but only one is presented to the user in foreground. This application is the only one, which is able to navigate to a different page and present it to the user. Also only this presented page is active at that time. When pressing the *Start button*, user is navigated from the application, also, as mentioned before, upon pressing the *Back button*, user might be navigated to the previous application if there are no more pages in current application's *page stack*. After returning to the application, user should be navigated to the place, where he left off.

Applications can therefore be in certain states to give an illusion of multitasking. The concept in is called *Tombstoning*. When the application is in the foreground, it is in a *Running state*. After an interruption such as navigating away or invoking a *Launcher* or *Chooser*, the application moves to *Deactivated state*, where is can save its data, and subsequently to *Dormant state*, in which the application is not running and no processing is performed. Based on the available memory and number of *Dormant applications*, the operating system can fully terminate the application and put it to the *Suspended (or Tombstones) state*. When activating the application again from *Dormant*, the developer does not have to do anything, as the operating system resumes the state automatically. [3]

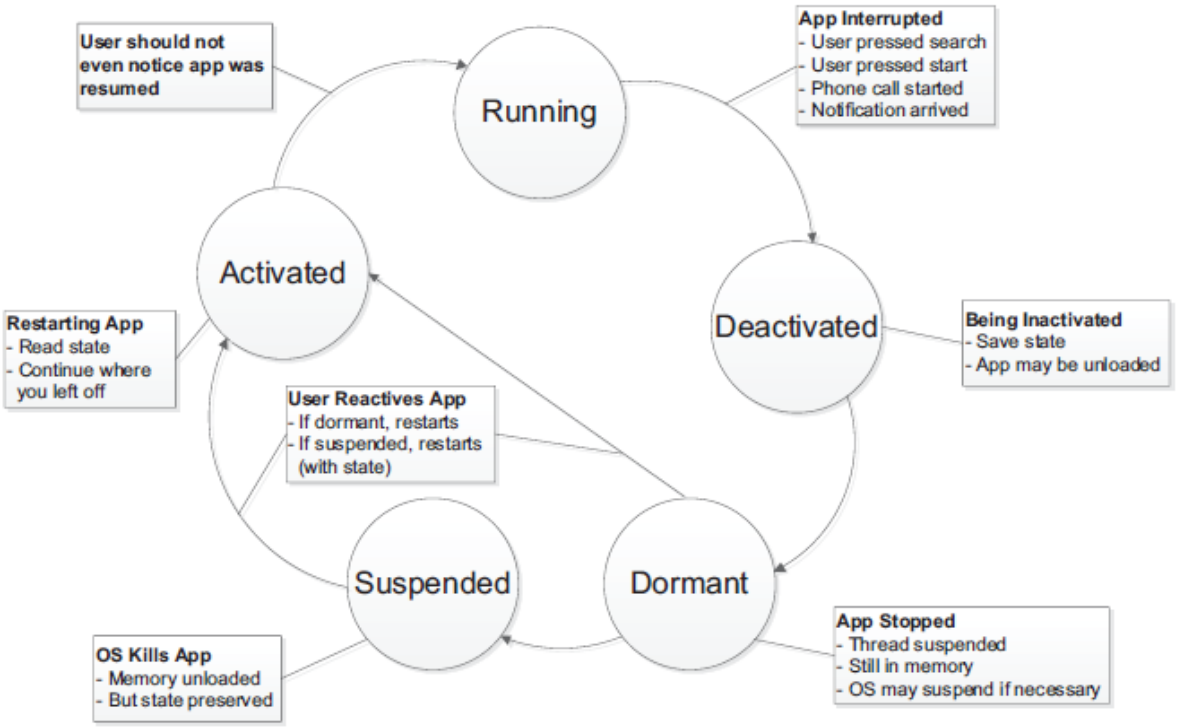


Figure 3.3 – The application lifecycle diagram [3].

Although the applications go through multiple states and data is being saved and loaded, the user should have the feeling as if the application was running in background: i.e. his position and state of game or visual elements must be preserved. Events are being provided to handle the transitions between states and also between pages. The most important are *OnNavigatedTo/OnNavigatedFrom* and *Activated/Deactivated* methods. These methods can be populated with code for saving state and data to achieve multitasking-like behavior for the user. The benefit of such approach is much better responsiveness and smoothness of in-foreground running applications as well as better battery life. [5]

If there is an interruption to the application such as an incoming phone call, instead of *NavigatedFrom* and *Deactivated*, the *Obscured* event is raised. The application continues running in the foreground, but is covered with a higher priority user interface, such as the phone call interface. In the Obscured event handler, the application may, for example, pause a game when relevant, since the user cannot see the applications content until the phone call is ended.

3.6 Multitasking

Although *Windows Phone* allows only one application to run in the foreground to maximize battery life and the overall performance of the system, there are some techniques that help the application perform certain actions in the background.

3.6.1 Background agents

Background agents in *Windows Phone* are parts of the application, which can run in the background without presenting any user interface. *Background agents* have access to some application's data, such as the *Isolated storage*, but also have many restrictions on the functionality they can provide. *Background agents* cannot use APIs like camera, radio, sensors, tasks, clipboard and others and they have limited memory available. Further, developer cannot precisely decide, when the background agent will run, as this is partially controlled by the operating system itself.

Periodic Background Agent

The *periodic background agent* is a *background agent*, which runs some code at most every 30 minutes. Another restriction to this kind of agents is the maximum running time of 25 seconds. If there are some other tasks, the background processing can be aligned, so the 30-minute period may vary. Also, if battery saver is enabled or there are more tasks than is the limit for a certain device, the background agent may not be run at all. [3]

Resource Intensive Background Agent

The *resource-intensive background agent* is can be used for longer lasting background tasks (up to 10 minutes), such as intensive synchronization tasks, but the system must meet tight conditions for executing it. The device must have external power source attached and battery must be over 90%, the device must be connected via non-cellular connection and must have locked screen. Also, no calls can be active during the execution. Other form of background execution is also *Background File Transfer service* or *Background Audio*.

3.6.2 Continuous background execution

The continuous background execution is another multitasking model since Windows Phone 8. This kind of execution is suitable for applications of navigation or run-tracker type. If a user navigates forward from this application, the application can still perform tasks in the background and keep the user informed using notifications or voice instructions. The system then balances resources for the application that runs in foreground and those that run in background, but in extreme conditions the foreground application is always prioritized. The user has an option to block applications from running in background. [5]

3.7 Graphical User Interface

The *Windows Phone* graphical user interface is very specific and is based on several principles. In order for the app to fit *Windows Phone* look and feel, it should stick to those principles. One of the most basic and innovative approach is the light feeling of all applications. Applications should focus on the content rather than on many graphical artifacts. The graphical user interface should be as simple as possible, without many sections on one screen, so the user does not get distracted. There should always be only one thing that is important and should be focused at in the given context. [1]

Windows Phone provides many controls that are customized so that the mobile user experience is as good as possible using touch navigation without stylus or keyboard. In this chapter, we will only focus on those that are very specific to the *Windows Phone* ecosystem and are representing the specific look and feel of this platform.

In order for all application to easily fit within the user interface, *Windows Phone* platform provides several fonts and styles that can be used as well as simple animations that should help the application visual responsiveness and fluidity. These animations can be used for elements as well as for transitions between pages.

3.7.1 Pivot and Panorama controls

Whenever there is a need to present large amounts of data, mobile applications are limited by their dimensions. Traditionally, scrolling is used on both desktop and web applications, or data is separated into different pages. *Windows Phone* has two built-in controls that should help present larger collections of data to the user. These controls are the *Pivot control* and the *Panorama control*. Both these controls are useful for organizing multiple components horizontally, so user can reveal extra content by swiping left or right. The main advantage of these controls can be taken when using the phone in *portrait mode*, since in *landscape mode*, there is not much space left for the content itself. [4]

The *Pivot control* is perfect for situations where a lot of information of the same type is presented. An example for such situation is the email client, where messages need to be filtered according to whether they belong to inbox, sent or archived folders. This could also be achieved using multiple pages, but the navigation would then become too complex. The *Pivot control* provides an easy solution for filtering large datasets and also and easy way of swiping between these filters.

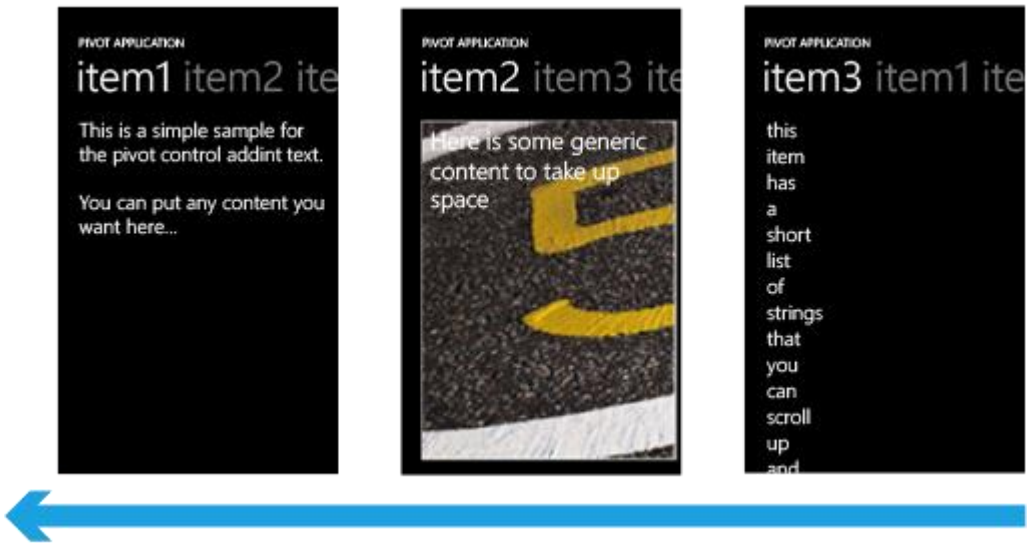


Figure 3.4 – The Pivot control [13].

The *Panorama control* is similar to the *Pivot control*, but is more media-oriented and also much more attractive looking. It is especially useful for presenting data of different kind on a single page. The *Panorama control* is basically a large canvas which can be scrolled horizontally and positioning its parts into the visible screen area. *Panorama-based* pages are usually used for main page of the application as they can present recent information as well as provide top-level access to deeply-placed pages of the application. [3]

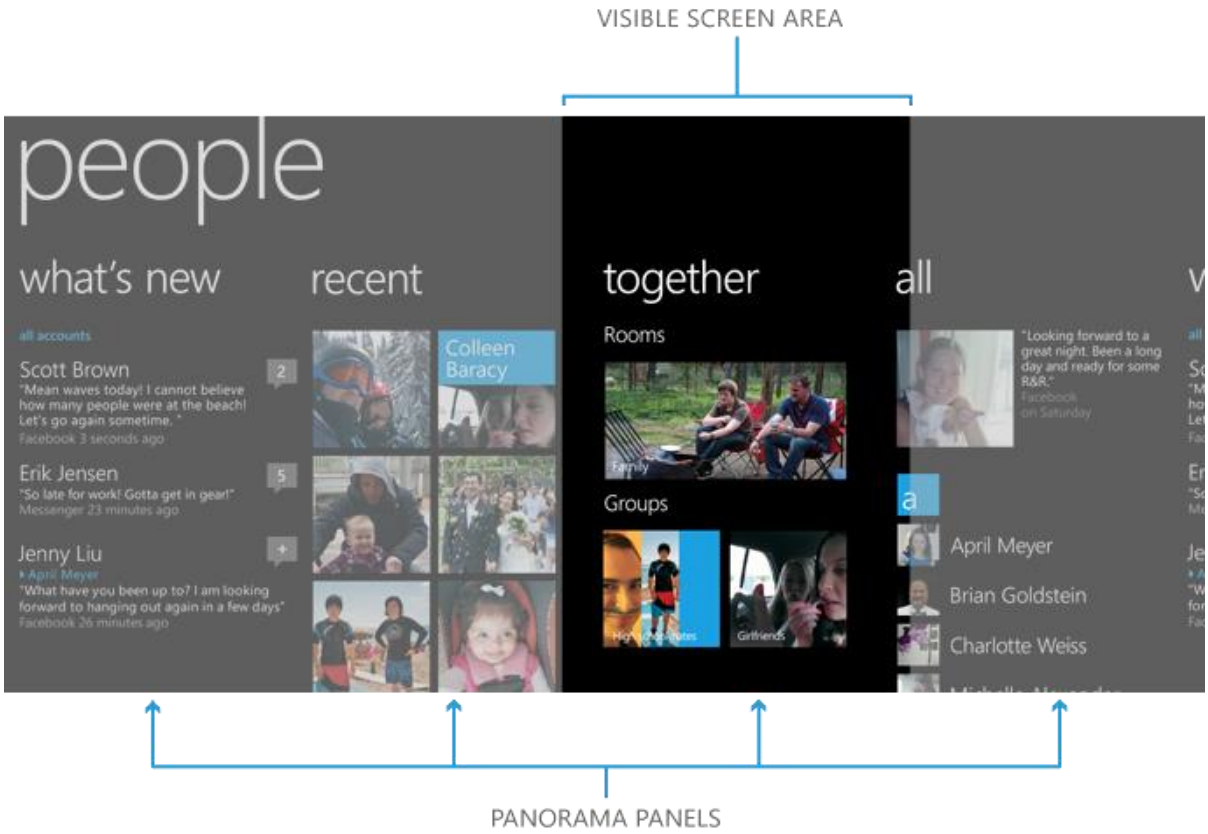


Figure 3.5 – The Panorama control [13].

3.7.2 LongListSelector control

This is a newly introduced control to *Windows Phone 8*, which provides a unified way of displaying list containing large numbers of items. For large lists, users had to scroll for a long time until they reached the relevant section. *LongListSelector control* provides a matrix of shortcuts that can be invoked upon clicking a group header item. List can be sorted by starting letters or by groups.

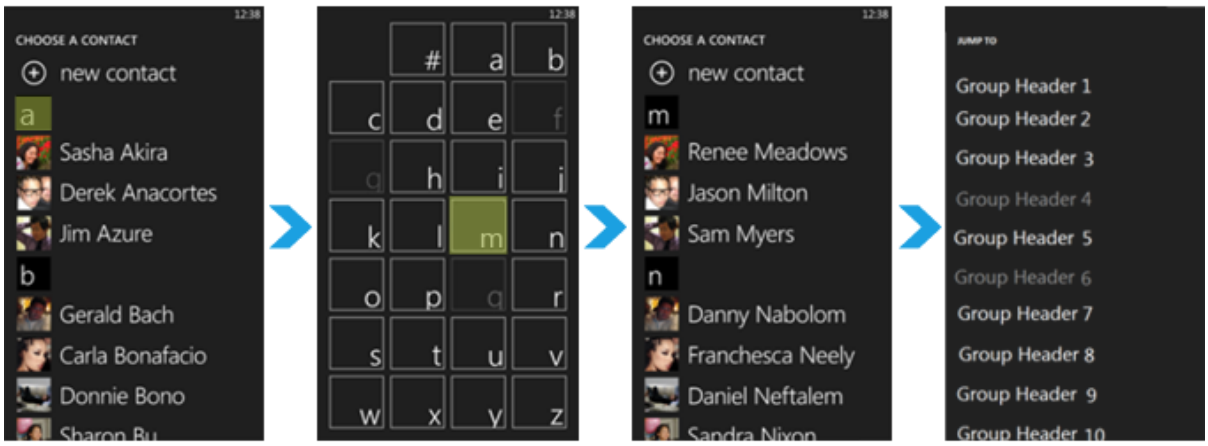


Figure 3.6 – The LongListSelector control [13].

3.7.3 Application bar

Since *Windows Phone* hardware does not provide any context button, menus can be placed in pages using the *Application bar control*. This control can be displayed minimized so that it does not take too much space, or can be viewed showing the menu buttons. Upon expanding the menu using the three-dot handle, more menu items can be viewed. The *Application bar control* is an essential part of most applications, since it enables the developer to further unify the look and feel of his application with the system. [3]

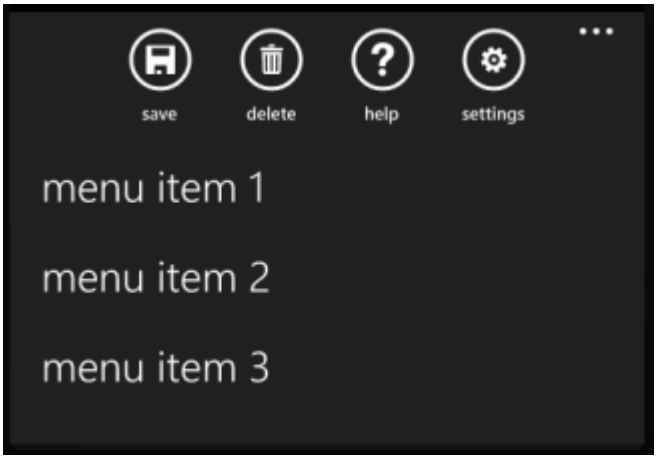


Figure 3.7 – Maximized application bar with menu items [13].

3.7.4 Storyboard

Storyboard is a concept for creating animations in *Windows Phone* applications. This concept provides an easy way of animating various properties of visual elements. There are four kinds of data that can be animated: *Color*, *Point*, *object* and *double value*. The *DoubleAnimation* is the most used as it permits to animate properties such as *Width*, *Height* or *Opacity*. [5]

The *Storyboard object* may be declared within *XAML code* and apart from the animation itself, also target object and its property must be set. For further simplicity, also *EasingFunctions* are provided, which help modify the animation not to have linear behavior, but to be more realistic for certain scenarios like sliding in or out. [5]

3.8 Resources

Applications in *Windows Phone* can use resources such as images, media files or fonts. These are usually referred to as *Binary resources*. Based on the *Build Action*, they can be embedded directly within *DLLs* as *Resources* or can be stored within application's *xap file* as *Content*. These resources can then be accessed using their URIs.

Another type of resources is *XAML Resources* [5], which represent stored objects such as *brushes* or *styles*. These resources are mostly stored within application's *App.xaml* file or inside page *xaml files*. Resources are *dictionary based*, so each resource must have a *key* under which it can be accessed. Resources from these dictionaries can be accessed by *StaticResource keyword* within xaml definitions, such as:

<TextBlock Text="Sample text" Style="{StaticResource SampleTextBlockStyle}" />

The resource is then applied upon element creation. When the resource is changed, in order to have the effect on the target element, the element would need to be updated manually. As resources can be defined within different resource dictionaries, such as in *App.xaml* or page's resources as well as in any other parent element's resources, applying the resource means that the most specific i.e. closest definition will be applied. Together with the possibility to use resources within other resource definitions, this means that powerful hierarchy of resource definitions can be made, reducing the amount of duplicate definitions. This behavior is similar to using *Cascading Style Sheets*.

3.9 Data Binding

As *Windows Phone* is based on latest technologies and modern programming frameworks, it also provides a way to present data within the user interface using data binding. Instead of handling the binding between UI elements (declared in *XAML*) and the actual objects by hand, there are several ways of automating the process to a certain degree, the developer might take advantage of.

3.9.1 Binding properties

For the basic scenarios, one can take advantage of the implementing *INotifyPropertyChanged* within his class. This approach enables the fields of a class to be bound to a certain UI element in a way that when these properties change, the change is propagated to the UI element.

For this setup to work, the class must implement *INotifyPropertyChanged interface*, properties must have public *getters* and *setter* and *setters* must invoke a *handler* that registered for the public *PropertyChnagedEventHandler event* [1]. Within the UI, elements' properties can use the binding syntax like:

<TextBox Text="{Binding PropertyName, Mode=TwoWay}" />

Such binding will then make sure that changes are propagated from the class to the UI as well as the other way, i.e. by changing the text value of this *TextBox*, the change will also affect the property within the respective object.

3.9.2 Binding collections

If collections need to be bound to lists within the user interface, objects must be present in an *IEnumerable collection*. By setting *ItemsSource property* of the respective UI element such as *ListBox* to this collection, the items will be set from it. For dynamic collections, it is advisable to use the provided *ObservableCollection*, which implement *INotifyCollectionChanged* and therefore items within the UI lists get dynamically updated. [1]

3.9.3 Converters

As the data sometimes are not represented in the way in which they should be displayed, the developer may make use of a class implementing the *IValueConverter interface* and implement *Convert* and *ConvertBack* methods. These can then be used for the conversion between data representation within the object and the visual representation. The converter must be declared within page resources to be accessible for visual elements.

3.9.4 Model-View-ViewModel approach

The *MVVM (Model-View-ViewModel)* is an evolution from *MVC (Model-View Controller)* pattern used extensively in *Windows Phone* applications. This pattern enables the developer to separate design and coding of the application. The View is represented by *XAML files* within the project and consists of user interface description. The *Model* is usually a collection of data objects or database data and is bound to the UI using *ViewModel*, which usually instantiates the *Model* and provides the *View* with a *data context* [1]. The *VMMV* approach is widely used in templates providing *Pivot* or *Panorama* pages.

3.10 Data storage

Windows Phone comes with a few concepts for storing and accessing data by the application. Since full system access to file storage is not supported, these scenarios are specific for target use cases.

3.10.1 Local folder (Isolated storage)

The concept for storing persistent data in *Windows Phone* application used to be called *Isolated Storage*, but as this name is still widely used [5], we will call reference local storage as *Isolated Storage* as well. *Isolated Storage* is a dedicated place for the application, which can only be used by this specific application. This approach has its advantage of being secure, so that no other application or the user can harm or steal data from the application, however, on the other hand prevents applications from sharing data this way. Also, applications are not allowed to directly access the phone's file system. In the *Isolated Storage*, application can store data in two ways: as a file or as a setting. The file behaves like a standard stream, so can be easily written or read using standard approaches.

The other possibility is to use *Isolated Storage settings*. This is a dictionary of key-value pairs that can be directly accessed using *IsolatedStorageSettings class*. If there are more complicated objects to be saved, the developer needs to make sure that they can be serialized.

The *Isolated Storage* also permits creating local database using *LINQ to SQL*. Relational data can then be accessed using an object-oriented approach. Proxy class needs to be implemented in this scenario. [5]

3.10.2 Other data storage locations

Applications can also access their installation folder; however, this access is read-only so no data can be stored within this folder. This is useful for bundled data that come directly with the application's installation or update.

Since *Windows Phone 8* supports SD cards, data can also be accessed in those locations. However, the access is again very limited and read-only. In addition, only file types that are explicitly registered by the application can be accessed from this location. These restrictions are a huge limitation of current platform release, as applications' data cannot be saved on SD cards, which limits devices with small internal memory capacity (but with CD card slot) to use applications such as navigation with large storage requirements.

The last data storage type is the *Media library*. This is a virtual storage container as media files can be stored both in the internal memory and SD card. However, users have no control over the location of media files. Since version 8, applications are allowed also to save songs within the media library [13]. Saving photos was available also in earlier versions of *Windows Phone*.

3.11 Tiles

Windows Phone platform introduced a new concept of tiles. The tiles are one of the key features, which distinguish *Windows Phone* from other platforms. Apart from having applications listed alphabetically, users can also create tiles from these applications and pin them to the Start screen. Tiles can be regarded to be a mix of a shortcut and a widget. They can be of three sizes and also three types, which will be discussed later. The concept of tiles is to allow users to customize their *Start screen* by arranging frequently used applications and also inform them about their state, as live tiles can provide pieces of information. Comparing them to widgets, they might not be as universal, but they have many advantages. Firstly, they have unified look and field, which is consistent with the user interface of the system, and, most importantly, refreshing images and other information within these tiles is controlled by the system, so it does not have negative impact on the speed and battery of the device [7].

All tiles types of tiles can be of three sizes, which the user can change on the Start screen: small, medium and wide. The medium sized tile takes place of 2 times 2 small tiles and the wide takes place as two medium sized tiles next to each other. The small tile is never dynamic, only can show a picture and numeric information.

Figure 3.8 – Type of tiles (Flip, Iconic and Cyclic) [13].

Flip tiles provide front and back face and flip randomly in intervals around 6 seconds. The front face has background image specified for all three sizes, title and count indicator, while the back face can contain title and text content. *Cyclic tiles* provide a possibility to show a series of images between which the tile randomly cycles by scrolling. The last type of tiles is the *Iconic tile*. These tiles only you two colors – white for the icon and text and the system theme color for its background.

Iconic tiles provide a place for icon and a counter for the small tile, extra title for the medium tile and another three content place holders for the wide tile.

3.11.1 Secondary tiles

Apart from the *primary tile*, an application may provide so called *Secondary tiles*. These tiles can be generated directly within the application and users can then pin them to their Start screen. Such tiles can represent a subsection of the application's content, such as a category in news application, a contact in social network application or a room or specific device in a home control application. *Secondary tiles* can be of any of kind similarly to the primary tiles. *Secondary tiles* can be unpinned programmatically, however, the standard approach is to let user unpin them from the *Start screen* manually [7].

3.12 Push notifications

Windows Phone devices can receive messages from the internet using *push notifications*. *Push notifications* can update a *Live tile* or can send data to a running application. In order to use this feature, there must be an accessible web server provided by the developer. The phone first registers with *MPNS (Microsoft Push Notification Service)* and obtains a unique URL, which can be used to send notifications to this particular device. Afterwards, the phone contacts the developer's server with running service and passes its URL. From now on, the server can send messages using the URL provided and *MPNS* will resend it as a push notification if possible to contact the phone.

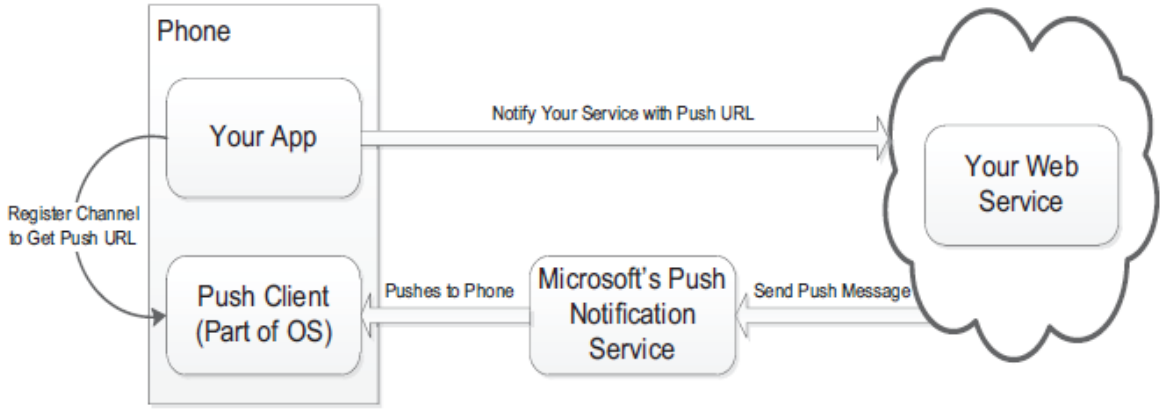


Figure 3.9 – The Push notification scheme [3].

Raw notifications are messages, which are sent directly to the active application. If the target application is not active, notifications are discarded. *Raw notifications* are simple *POST* messages sent to the *MPNS*. After contacting the *MPNS*, a response is obtained with the status of the notification, the status of the phone connection and subscription status providing information whether the phone is still subscribed to the service.

Toast notifications are messages simple messages appearing on the top of the phone's screen. They are only delivered when application is not active and upon tapping them, user launches the respective application.

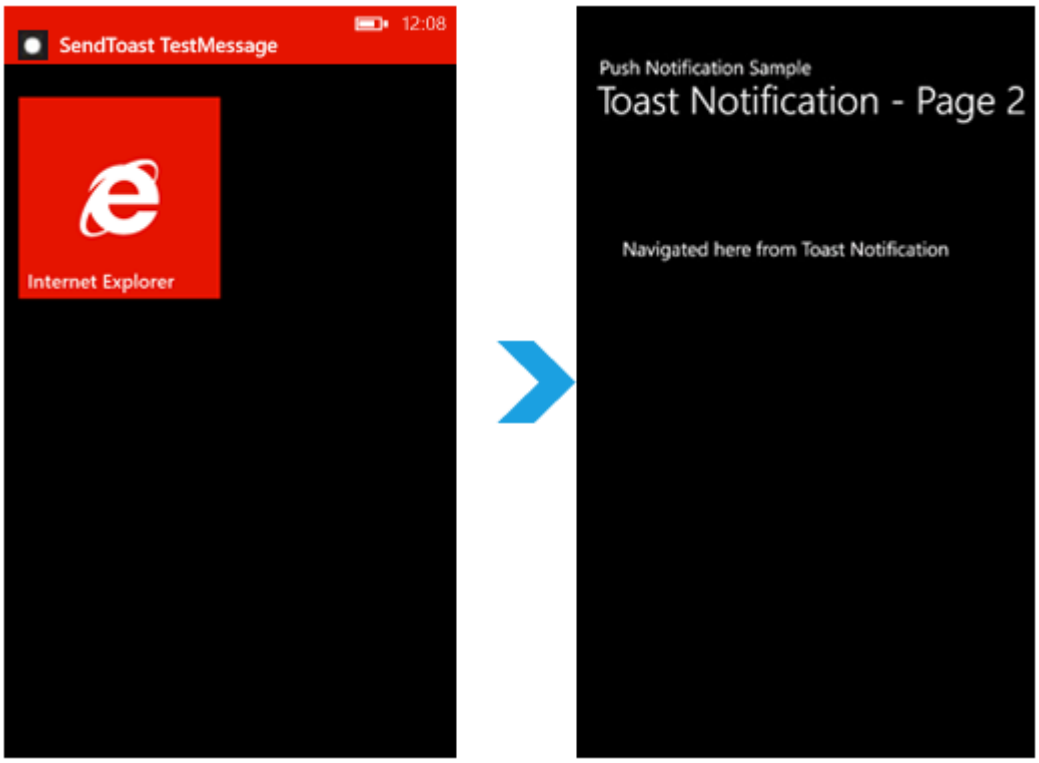


Figure 3.10 – Toast notification [13].

Live Tiles is the last usage of push notifications. It allows sending messages containing an XML document for Live Tiles on the start screen. The XML structure is strictly defined and provides the tile with update information such as title, count or background image. [1]

3.13 Alarms and reminders

The *Windows Phone* platform supports two ways of alerting user from within an application – *alarms* and *reminders*. Both these features are similar and allow an application to notify user event when not running. An *alarm* is a simple notification, which can be set on specific time including a custom sound to be played. After tapping the alarm pop-up the user is redirected to the application, but only to the initial page, as if the application was launched from the application list. [3]

The *reminder*, on the other hand, cannot be assigned a custom sound to be played. The default reminder sound is played instead. The biggest advantage of the reminder though is that after tapping it, the user can be redirected to any page within the application and also query string can be passed.

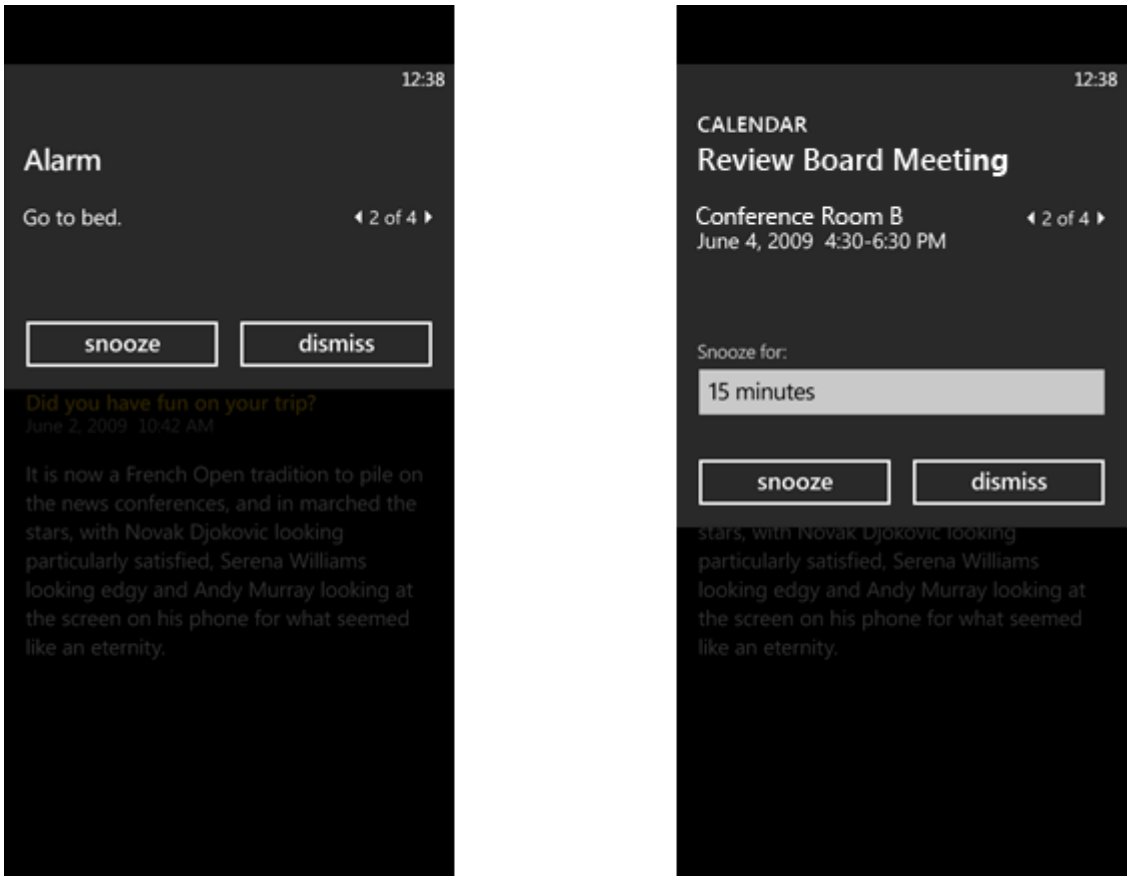


Figure 3.11 – The Alarm and Reminder notifications [13].

3.14 Speech

Windows Phone 8 provides an expanded set of features to support voice control of both the phone system and applications. *Voice commands* can be used either by directly launching the *Global Speech Experience* by holding the *start button* and saying the command, or can be invoked within a specific application.

3.14.1 Voice commands

This mode allows users to use voice commands to start an application, but since *Windows Phone 8*, also to perform actions handled by an application, such as playing a specific song, inserting new tasks or even to perform a completely custom behavior. [1]

To register voice commands, the *Voice Command Definition file* must be registered by the application. This file describes complete format of the expected voice commands and also the respective answers and target pages. The default template for *Voice Command Definition file* consists of *CommandSet* sections, which describe voice commands for given culture. Phone's speech culture must match these cultures and for each culture, separate *CommandSet* must be present [1].

Each *Command* within the *CommandSet* can consist of an example, which is displayed to the user, the sentence or sentences with optional words and tokens from *PhraseList* which the phone should listen for, the feedback the phone gives back and the target to navigate to, including optional parameters. An example of a simple *Voice Command Definition file* might look like:

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

<VoiceCommands xmlns="http://schemas.microsoft.com/voicecommands/1.0">
  <CommandSet xml:lang="en-GB" Name="RoomsCommands">
    <CommandPrefix>Home Control</CommandPrefix>
    <Example> turn on </Example>

    <Command Name="TurnOn">
      <Example> turn on </Example>
      <ListenFor> [and] turn on {device} </ListenFor>
      <Feedback> Turning on {device} ... </Feedback>
      <Navigate Target="RoomsPage.xaml" />
    </Command>

    <PhraseList Label="device">
      <Item> one </Item>
    </PhraseList>

  </CommandSet>

</VoiceCommands>
```

Figure 3.12 – The Voice Command Definition file from our application.

The *PhraseList* can be dynamically updated in the code to reflect desired set of commands. This can be done using *VoiceCommandSet.UpdatePhraseListAsync* method. Once, when the application is first launched, the *Voice Command Definition file* must also be initialized using *VoiceCommandService.InstallCommandSetsFromFileAsync* method. After the command is processed, recognized data is sent to the page specified within the *Target*. The parameters include *voiceCommandName* and also *reco*, which contains whole recognized text.

3.14.2 Speech recognition and Text-to-speech

All speech recognition features can be also used within the application itself. The advantage of using speech recognition this way is that the developer has higher customization possibilities of both the design and functionality of speech recognition. For example, small microphone can be used in a *textbox* to invoke speech recognition features. Recognizing can be invoked by calling *RecognizeAsync* method from *SpeechRecognizer* class.

"Windows Phone 8 includes support for pre-defined grammars for free-text dictation and web search, and also supports custom grammars that are authored using the industry-standard Speech Recognition Grammar Specification (SRGS) Version 1.0." [13]

Text-to-speech feedback features are available. These can be used by calling *SpeakTextAsync* method from *SpeechSynthetizer* class.

"Your app can speak a simple string of text, or a formatted string defined by the industry-standard Speech Synthesis Markup Language (SSML) Version 1.0." [13]

3.15 Location

In *Windows Phone 8, location* and *maps APIs* are based on *Nokia maps platform*. This is different from the previous version of *Windows Phone* and has many advantages such as easier implementation and higher performance, as well as richer functionality.

The location in *Windows Phone 8* can be determined from various sources such as GPS, Wi-Fi or cellular radio. The correct device for obtaining the position is determined automatically by the system based on the required accuracy set in the *DesiredAccuracy property* of *Geolocator object*. This approach helps to keep battery life longer when high accuracy is not vital. When obtaining the position, the developer may also set *maximumAge* for the same reason.

Location-tracking applications can also run in background as described in the chapter about multitasking. This is useful for voice navigation as well as for collecting location data.

Maps in *Windows Phone 8* were also enhanced. Applications can use the *Map control* directly, which enables displaying maps in the same manner as built-in applications. Both two and three-dimensional views are available and also different modes can be displayed. Further, this control supports *overlays*, so custom data can be displayed above the map itself.

As mentioned in the *tasks* chapter, there are also tasks available for location and maps. These include *launchers* such as *MapsTask* for searching specified items within current location or *MapsDirectionsTask* to get route directions for given destination.

If there is a need for navigation in the application and *Tasks* are not sufficient for this purpose, applications can make use of the *RouteQuery* and *MapRoute APIs* to obtain driving instructions or information about a computed route. [1]

3.16 Tasks

When users of an application are required to perform a standard task, such as making a phone call to someone on whose picture they are looking within a custom application, or when they wish to upload a picture in a custom application, it would be very difficult and sometimes impossible to include such feature within this application due to the restrictions of the *Windows Phone* operating system.

However, *Windows Phone* provides a way, how to achieve such functionality within third-party applications in a way that it is OS-wide consistent. Developer might make use of *Tasks*. *Tasks* are of two types, *Launchers* and *Choosers* and each of them has a slightly different concept of use [3]. When running a *task* from an application, the currently running application is put into dormant state and might be *tombstoned* as described in the chapter about application's life cycle. This means that different application is launched and brought to the foreground and when the task is completed, the calling application is activated again, but user should have the feeling as if he was still within the calling application all the time. As a subsequence, if the application is terminated during the task by the system and is not able to restore its state, it will not be made automatically by the system either.

3.16.1 Launchers

The first type of tasks is *launchers*. These *tasks* launch selected built-in application such as web browser or maps and let the user perform the selected task. User can, of course, choose not to perform any task and return to the calling application.

Examples of these tasks would be: opening a specific link within a browser, showing a location within maps, sharing statuses on social networks or writing and sending an email message or text message, search for a specific text, play music, perform a call or search a contact.

3.16.2 Choosers

Choosers are very similar to *launchers*, but after the task is launched and user performs some action such as taking a picture with camera, calling application is activated and supplied with data from the completed task. If the user, however, leaves the task by going to the Start Screen, the application may never be activated again, so must be prepared for this use case.

Examples of choosers are: obtaining address of a contact selected by the user, get a picture taken by the user, obtain phone number or email address from a contact selected by the user or save contact or ringtone to the device.

3.17 Globalization and localization

Since *Windows Phone* now supports many languages and applications are available in many different countries, *globalization* and *localization* play significant role in application development. In *Windows Phone managed applications*, most of the functionality is simplified and assisted by the *.NET framework*, so developers do not have to create their custom approached [7].

Globalization in applications helps display all data such as numbers, dates or phone numbers in a way that users from specific region are used to. A concept to help dealing with *globalization* is using *CultureInfo* objects. These objects can be assigned to *CurrentCulture* property of current thread. Instead of hard-coding values such as dates, *CurrentCulture* can then be used for formatting date or time values. *Globalization* also affects currency format and symbol and of course the sort order of items.

Localization of the application takes *globalization* to a next level, where all strings are presented in user's language. Special attention must be paid to the App bar and application title, as their localization process is slightly different. The most important is to move all text information from the code into resource files. Application can be easily modified by copying every string to the resource file and using a reference to it such as:

{Binding Path=LocalizedResources.ApplicationTitle, Source={StaticResource LocalizedStrings}}

instead of hard-coded text. For *localization* purposes, every single culture supported by the application needs to have its own *resource file*. The *App bar* can be localized by uncommenting the *BuildLocalizedApplicationBar* method call in basic application template's files. More cultures must be specified within project's properties.

With *Windows Phone 8*, there is the *Multilingual App Toolkit*. This toolkit helps localizing application to more languages if standard globalization and localization concept is used. If this condition is met, the toolkit provides an efficient way of localizing the application into different cultures, providing user interface for choosing languages, integrating with *Visual Studio 2012* and even providing connection to *Microsoft Translator* to suggest translations. This feature also permits machine translations as a starting point for more precise localization. [7]

3.18 Network

As *Windows Phone* devices are almost constantly connected to a network, networking plays a significant role in most applications. Due to the requirement of responsive user experience, *Windows Phone* networking features only support asynchronous calls. This way, event without complex programming, the user interface stays responsive as network calls are handled in different thread and cannot therefore lock the interface up.

The basic class for asynchronous networking is the *WebClient class*. Using the asynchronous pattern, most asynchronous calls end with the word *Async* and the events that occur upon finishing the call end with the word *Completed*. An example would be the *DownloadStringAsync* and D*ownloadStringCompleted*, which allow getting data from a webserver. Other classes that can be used instead of the *WebClient class* are *HttpWebRequest* and *HttpWebResponse*. Their combination can be used to take deeper control of the networking calls; however, it is worth noting that these classes also only support asynchronous calls unlike their desktop equivalents. [3]

As devices can be connected to the network using technologies with very different download speeds and pricing, *Windows Phone* provides an API, which can supply information about current connectivity attributes. The *DeviceNetworkInformation class* has several useful properties such as *IsNetworkAvailable*, *IsCellularDataEnabled* or *IsWifiEnabled.* Based on these information, applications can decide not to download some content, or for example download image previews only if no Wi-Fi connection is present. [3]

There are also other useful classes such as *NetworkInterfaceType*, *NetworkInterfaceSubType* or *NetworkInterfaceInfo*, which provide additional information about the network interface, so that the application can determine the network speed, characteristics or even *SSID* of a wireless network. These information can help the application determine not only what data it can download, but also if the device is present on home cellular network or even home wireless network and perform adequate actions.

3.19 Camera

An application can work with device's cameras in certain ways. The first approach is using *CameraCaptureTask chooser*. Once activated, the built-in camera user interface is launched and upon taking the picture, it is saved to the camera roll and also returned to the application as a result. This approach is suitable for most scenarios when a photo needs to be taken. It is both user friendly and memory efficient [6].

Another approach is using the camera directly. This way, application can directly access the camera stream for taking both pictures and videos. In *Windows Phone 8*, there are two sets of APIs – the *PhotoCamera* and the *PhotoCaptureDevice*. The *PhotoCaptureDevice* API is newly introduced in Windows Phone 8. It is available for both managed and native applications and is supposed have better performance in managed applications comparing to *PhotoCamera API*. [13]

Using these APIs, the application can set camera settings as well as capture the live feed and work with it. This can be useful for camera like applications or for augmented reality applications. However, *Windows Phone 8* also supports a new concept of using camera within applications called *Lenses*. *Lenses* are applications that can be launched directly from the built-in camera application and can provide extensions such as filters, instant sharing of pictures or overlaying additional information on top of the camera preview [13]. However, they are not limited to such functions and could be also used for interacting with other devices based on the camera image processing.

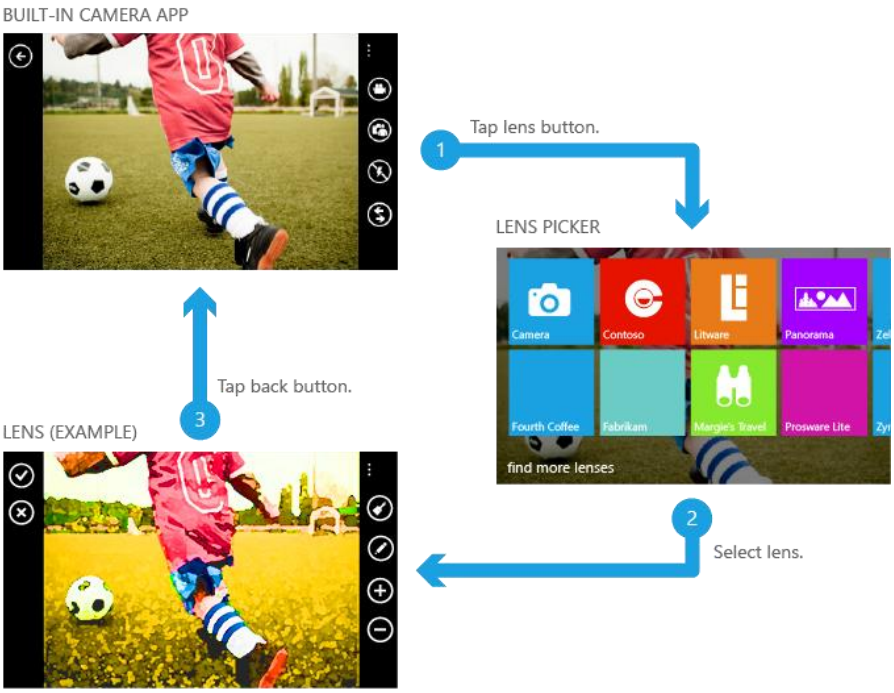


Figure 3.13 – Lenses scheme [13].

3.20 Sensors

Multiple types of sensors are supported in *Windows Phone*. These sensors can be used for obtaining device's orientation or even motion. *Windows Phone* supports *accelerometer*, *compass* and *gyroscope* sensors, but developers can also make use of the combined *motion API* [6]. Sensors cannot be used while applications run in background as mentioned in the multitasking chapter.

The first sensor is the *accelerometer*. *Accelerometer* sensor is required to be present in all *Windows Phone* devices and it is the basic sensor for determining device's orientation. The accelerometer can provide information about orientation in all 3 axes and expresses its readings as a three-dimensional vector, where each is represented in gravitational units. If the device is put on a flat surface, the readings will produce -1g in the Z-axis, which represents the gravity force.

Another sensor is the *gyroscope*. *Gyroscope* is not required in all devices. This sensor provides information about device's orientation is space. The *gyroscope* returns rotational velocity of the device in all axes. Last supported sensor is the compass. This sensor is also not essentially part of each *Windows Phone* device. The *compass* sensor provides information about angles between the device orientation and the Earth's magnetic field. As this sensor is able to detect changes in magnetic field around the device, it can also be used for detecting metals to certain extent. It is important to note that applications must count with the possibility of devices not having this sensor [2].

3.20.1 Combined motion

Since raw sensor data might be difficult to use, developers can make use of the *Motion class*. This class handles low level sensor calculations as well as geometrical transformations and presents data in more ready-to-use format. Applications can than easily use information about device's movement, orientation or acceleration without the need to combine different sensors and calculate the data themselves. *Normal motion API* uses only *accelerometer* and *compass* sensors, while the *enhanced API* also adds the *gyroscope* and is therefore more accurate [2]. However, as mentioned, not all devices support *gyroscope* sensors, so the application must be aware of such possibility.

3.21 Proximity

In *Windows Phone 8*, the *Near Field Communication (NFC)* functionality is based on the *proximity APIs*. Using these APIs, developers can include in their apps features such as establishing a connection to other device by putting devices close to each other or sending and receiving content from other devices or markers. Not all devices running *Windows Phone 8* do support proximity features, as it is not a required hardware accessory.

"Near Field Communication (NFC) is an international standard for short-range wireless connectivity that provides intuitive, simple, and safe communication between electronic devices. NFC is the technology on the phone that makes Proximity scenarios possible." [13]

Although *NFC* might be useful in certain scenarios, it also has its limitations. In order to communicate, devices must be very close. This range is usually around at maximum around 4 centimeters. Also, the maximum transfer speed is usually in tens of kilobits per second.

Given the abovementioned limitations, there are three possible usage scenarios for proximity in applications. The first scenario is using proximity for establishing connections via other wireless technologies such as Wi-Fi or Bluetooth, which provide much higher transfer rates. This scenario enables users to touch devices, which will then try to use Bluetooth connection if enabled on either devices; or Wi-Fi connection if both devices are connected to the same infrastructure and can ping each other. The second scenario is based on using the device as a reader of *NFC tags*. In this scenario, when device is moved within a range of such *NFC tag*, its information can be obtained. The last scenario is direct communication using *NFC* technology. This requires both devices to stay within close range and allows them to exchange messages directly. As states before, the transfer rate is very limited and is not therefore suitable for multimedia content [13]. However, for simple messages, such as configuration files in simple strings, this method might be the best choice as it does not require any further configuration on the devices other than running the application and have *NFC* enabled.

3.22 Windows Phone Store

Windows Phone applications can be uploaded to *Windows Phone Store*. *Windows Phone Store* is also the only official way for other users to obtain applications to their phones. There are also other ways such as loading apps to developer unlocked phone or company phones, but these are only targeting limited user groups.

In order to publish application within the *Windows Phone Store*, there are certain steps and requirements that will be described below. Firstly, registration within the *Windows Phone Dev Center* must be made and annual registration fee of $99 must be paid. This fee is not required for students enrolled in *Microsoft DreamSpark* program. After joining the *Dev Center*, *Windows Phone* applications can be submitted for the certification process and if successfully passed, application is signed and moved to the *Windows Phone Store*, so that users can download or purchase it. If the application cannot pass the certification process, failure results are sent back to the developer. It is worth mentioning that 70 percent of money paid by the user goes to the developer and 30 goes to *Microsoft*.

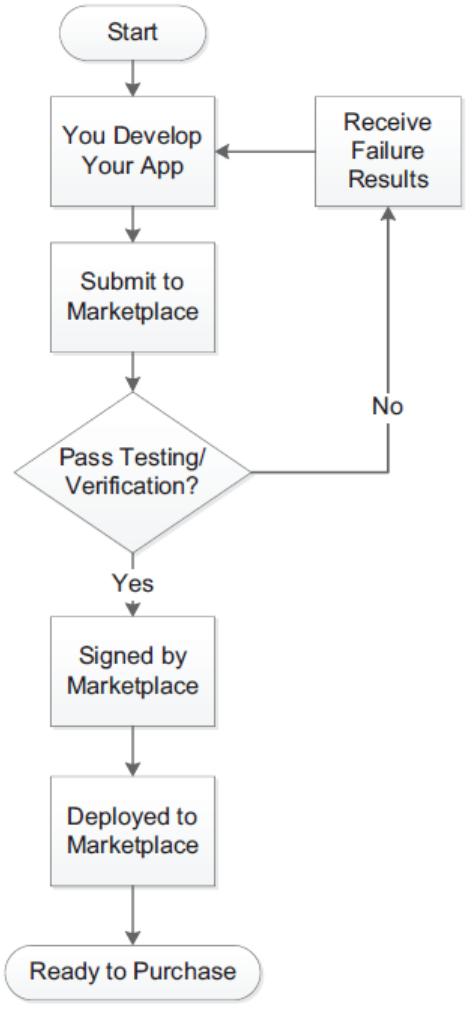


Figure 3.14 – The certification process diagram [3].

To ensure that the application passes the certification process, the developer has to make sure that his application meets certain requirements. *The application must be reliable* [3], so if the there is any possible instability, the app can be refused. Also, *applications must make efficient use of resources* [3]. If the app should fully discharge the battery because of non-efficient use of sensors, it might be another reason not to pass the certification process. Another requirement is that *applications must not interfere with the phone functionality* [3], meaning that if changing settings or working with other phone data, user must be notified accordingly. Lastly, *applications must be free of malicious software* [3], so it must be safe to use them.

These were general restrictions, but there are also many more rules and restrictions along the way while publishing the application. For example, the whole package cannot be bigger than 400 MB; also adequate images must be supplied for the application, which are used within the Windows Phone Store and on the device itself. Applications must have a title and information about version and technical support. [13]

Also, regarding the application itself, it cannot override or ignore any system notifications, cannot use forbidden APIs, promote or distribute any content through alternate stores. Also, applications which are not fully functional or use advertising and music sales through different channels, other than those provided by *Microsoft*, will not be accepted. Applications that send user data without notifying them will break the certification process as well.

It might seem to be very complex to publish the application while complying with all the rules and it certainly is for more sophisticated designs. However, Microsoft provides well written guides and also new version of developer tools for *Windows Phone* makes it much easier to fulfill those requirements.

One more thing worth mentioning in this context is the concept of *trial apps*. Although it is possible to develop both free and paid versions of an application, developers can also develop only one version and provide the possibility to download the application as trial. Then, within the application, it is possible to make use of the *IsTrial* method to check whether user has purchased the application and is therefore eligible to use extended features. [6]

3.23 Development tools

Applications for *Windows Phone* are developed using the *Windows Phone SDK*, which includes many essential tools, such as *Visual Studio Express 2012 for Windows Phone*, *Microsoft Blend for Visual Studio 2012*, *Windows Phone Emulator*, *Developer Registration tool* and others.

Visual Studio is the place, where most of the coding is done. It is well connected to other tools, so there is mostly no need to run other parts of the SDK separately. The *Express edition* is provided free of charge and includes all necessary features that are required for *Windows Phone* development.

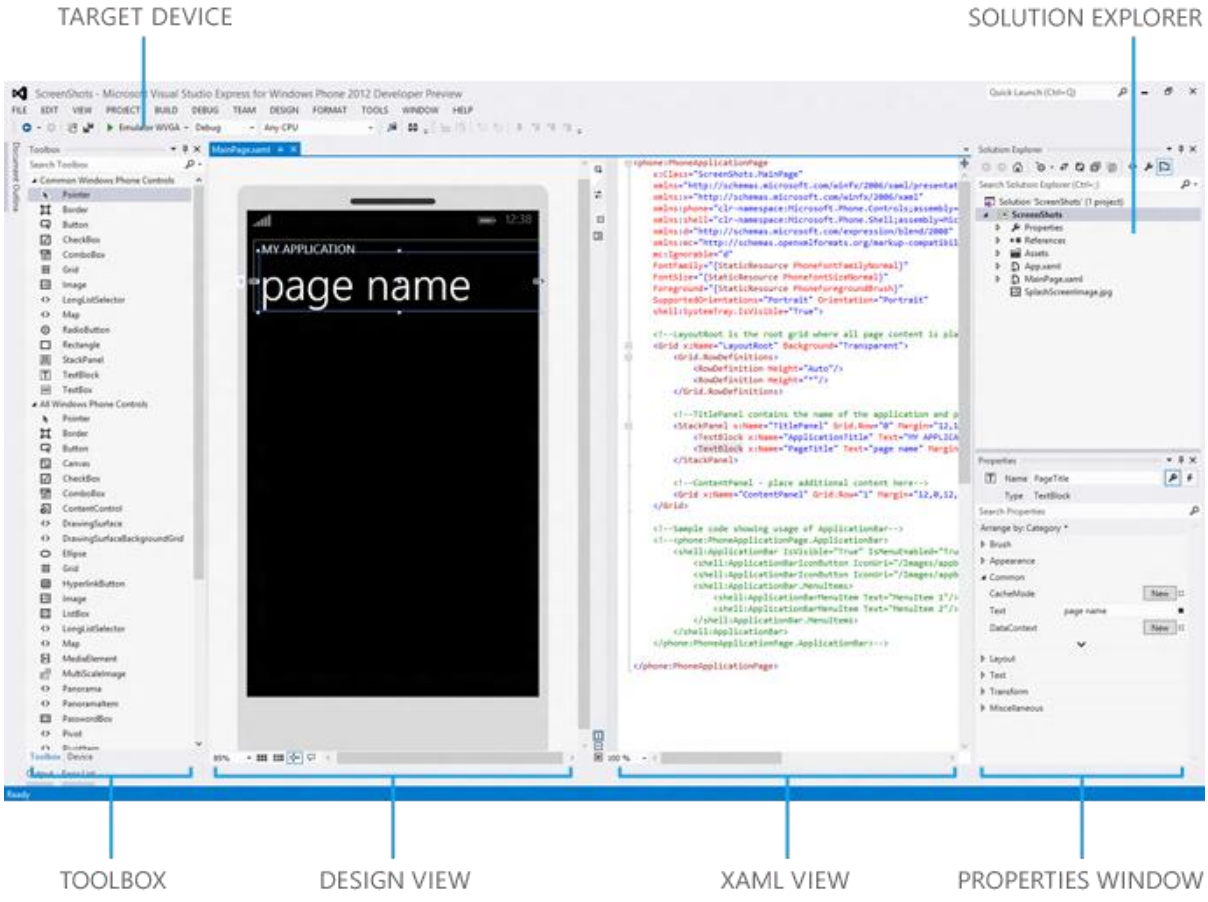


Figure 3.15 – The Microsoft Visual Studio 2012 for Windows Phone [13].

Another supplied developing environment is *Blend for Visual Studio 2012*. *Blend* is a design tool for designing interactive user interfaces based on *XAML* and provides many features of a graphical editor. It also supports importing of various formats and creating many graphical and animation effects.

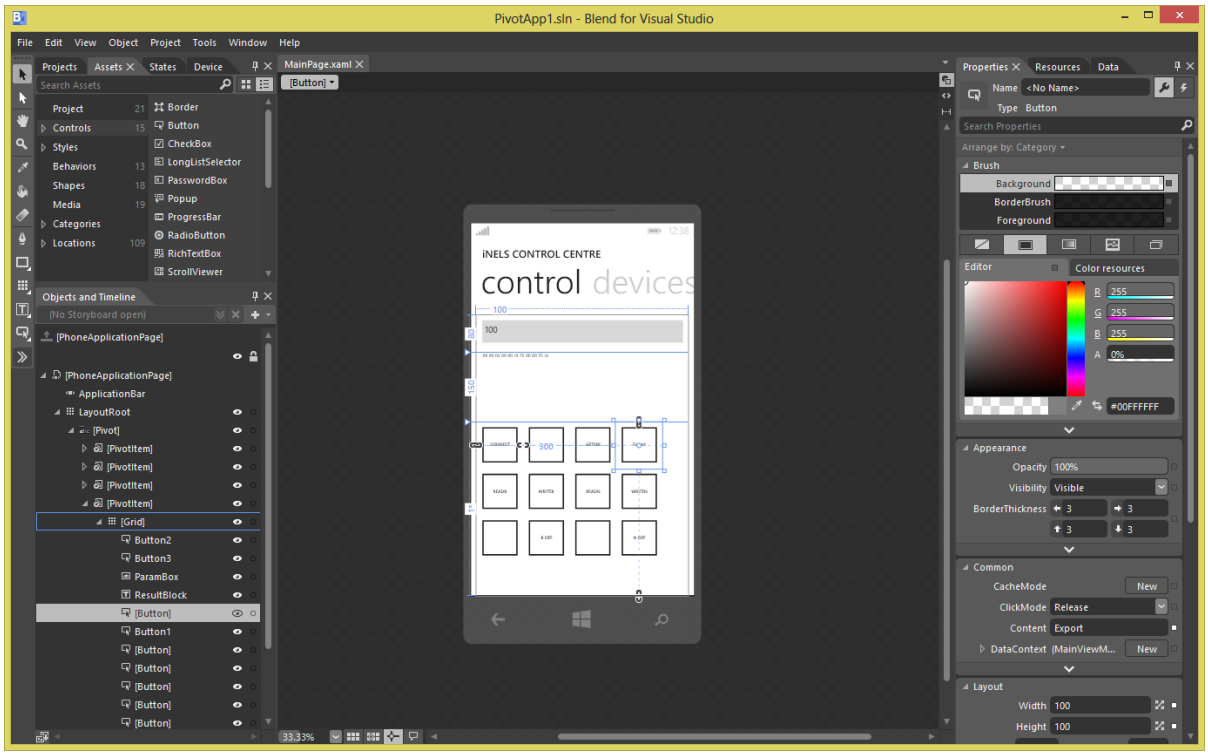


Figure 3.16 – The Microsoft Blend

As mentioned, one of the tools obtained is the *Windows Phone 8 emulator*. This new version of *Windows Phone emulator* has a new architecture and therefore different hardware requirements. As most of the other tools, *Windows 8* or *Windows Server 2012* is required. Further, as *Windows Phone 8 emulator* is based on *Hyper-V virtualization technology*, *Hyper-V* features must be enabled within the operating system. Also, *Hardware-assisted virtualization*, *Second Level Address Translation (SLAT)* and *Hardware-based Data Execution Prevention (DEP)* must be supported by the system's hardware and enabled in *BIOS*. In order to check the development system, Microsoft has provided a tool called *Coreinfo*, which can provide information about the system. More detailed guide about the system checking process is described in [23].

3.24 Third-party libraries

Windows Phone 8 comes with default components and controls. Some of the most phone specific controls were already described. However, there are also third-party libraries providing some of the controls that might be useful while developing *Windows Phone* applications. Libraries can be often downloaded as a source code or as compiled libraries. A great tools, which may help with downloading these libraries and including them within the application is called *NuGet*. *NuGet* is a *Visual Studio extension*, which provides easy way of downloading libraries from its gallery and also manages their updating and referencing within an application [24].

3.24.1 Windows Phone Toolkit

Probably the most used library for *Windows Phone* project is the *Windows Phone Toolkit*. This is an open source library provided by *Microsoft*, which includes many useful controls such as *WrapPanel*, *ContextMenu* and *DateTimePickers*, as well as *Effects*, such as *SlideInEffect* and *Tilt effect*.

These controls are unique as they perform similar functions like the *Windows Phone* itself uses and therefore applications using them fit nicely within the overall user experience concept. The library also provides sample codes within its source and can be modified. All parts are published under the *Microsoft Public License (Ms-PL)* [15] [Attachment C].

3.24.2 XML-RPC.NET

Another useful library is the *XML-RPC.NET*. Although this library does not provide any controls, it provides functionality for implementing *XML-RPC* services and clients. The interesting part for this project is the client side of this library and its features supporting asynchronous calls. Since most of the functionality of this project is based on *XML-RPC* communication with the server, *XML-RPC.NET* is an essential part, which cannot be omitted.

XML-RPC.NET is released under *MIT X11 license* [14] [Attachment B].

4 Home Control application

In this chapter, we will describe the application, which was developed during this project. The aim was to develop an application for controlling *iNELS* devices over *EPSNET* protocol and also over *XML-RPC* protocol. Our goal was also to build user interface, which would correspond to guidelines defined by *Microsoft* [13] and also fulfill common usability requirements as presented in [11]. Firstly, we will focus on the content of project's folder, subsequently on the user interface and also on the extensions that were implemented, or could be implemented in the future.

4.1 Project description

In this section, we will describe the content of the whole project and also functionality of the files included. All code files will be discussed in details, together with most of the included methods.

4.1.1 Properties

AppManifest.xml

No modifications were required to this file and it does not differ from the standard project.

AsseblyInfo.cs

AsseblyInfo is a file, where all assembly information can be specified. The most important are information such as title and description and also application version number and default language for resources. Assembly information must be specified in order for the application to pass the certification process.

WMAppManifest.xml

In *Visual Studio*, this file is opened using a custom user interface and allows specifying essential properties of the application. The first tab within this allows specifying UI details that identify the application such as its display name, the page the application should navigate upon launching and most importantly all the tiles used for this application and supported resolutions. Next tab sets the capabilities of the application such as *ID_CAP_ISV_CAMERA*, which allows the application to use camera-related APIs. Third tab can be used explicitly telling the user that the app requires certain hardware to be present. These are not set as our application does not explicitly require any specific hardware for its base functionality. On the last tab, default language together with all supported languages can be set and additional package information is present as well.

4.1.2 References

Apart from the standard references *(.NET for Windows Phone and Windows Phone)*, our application requires references to the following libraries:

CookComputing.XmlRpcPhone

This is a reference to the mentioned *XML-RPC.NET* library [14], which is essential for most of application's communication with *iMM* server.

Microsoft.Phone.Controls.Toolkit

Reference to the *Windows Phone Toolkit* library [15] is essential for some user interface components, such as the *WrapPanel*, which is used for displaying in-app tiles, the *TiltEffect* for tiles and also the *ContextMenu*, which allows system-like editing of places within settings.

4.1.3 Assets and images

These two folders contain all images used within our application. Images are supplied in white color with transparent background, so they can be used as an opacity mask. Instead of assigning our UI elements the image directly, we can set their foreground to use a static resource such as the system's currently selected foreground color, the background to use current system accent color and set their opacity mask to use our image. This scenario results in the application changing colors according to user selected system themes and the application therefore becomes more system integrated from the users prospective.

Images that should serve as icons for the *Application Bar* do no need to have the circle included, as when used within the *Application Bar* configuration, the outer circle will be added automatically. However, as our application also used some of these icons outside the Application Bar, the circles must be supplied.

4.1.4 Resources and other files

The *Resources* folder contains resource files for individual cultures supported by the application. The *App.xaml* and *App.xaml.cs* files are the base for our application and contain methods for application's lifecycle handling as described in the chapter Application life cycle. Apart from the methods, *App.xaml.cs* also contains a definition for static object of type Places, which is used in our application to access all model data. The *RoomsVCD.xml* file is used for speech recognition and will be described within the extension.

4.1.5 Devices

This folder contains the whole hierarchy of the application representation of *devices*, *rooms* and *places*. *Place* represents one location, such as home or an office, with one configuration. *Places* consist of *rooms* and *rooms* contain several *devices*.

Places class

This class lies on the top of the hierarchy. It contains list of *Place* items and also index of the active item. The *activeItem* property returns the active item based on the *activeItemIndex*. Places class is used as a container within application wide model.

Place class

The *Place* class is the one that holds all information related to a specific location. The main properties are *PlaceName*, *PlaceType* specifying the type of connection used for this place, *Host* and *Port* and *ConfigID*, which serves as an identifier for downloading configuration files from *public server*. All these properties are employing the concept of *INotifyPropertyChanged* as described within the data binding chapter to allow *two-way binding* with the user interface. The class also holds *PivotRoomCollection* object, which contains *rooms* as explained later.

PivotRoomCollection class

This class contains an *ObservableCollection* of *PivotRoom* objects. Other than that its constructor, which accepts *ExportPub* and *RoomsCfg* objects as its arguments, fills the collection with *PivotRoom* objects with devices according to certain rules. For each room in the *RoomsCfg* object, all types of items are iterated over and for each of these items; suitable device is added to the collection. Each of these devices is configured so that it contains a proper name and also link to the relevant icon.

PivotRoom class

The *PivotRoom* class represents a specific room. It contains an *ObservableCollection* with *devices* belonging to the *room* and also has a *RoomName* feature that is displayed within user interface. The class itself needs *KnownType* attributes with all types of *Devices*, so that it can be serialized when storing data within *isolated storage*.

Device class

The *Device* class is a basic class representing devices, i.e. iconic items within rooms, in our application. This base class implements *INotifyPropertyChanged* interface and contains three basic properties: *Value*, *Name* and *ImagePath* that are inherited by all child classes. Also, there is a virtual method *getStateDevice*, which should be overridden in child classes.

Device child classes

Classes that inherit from the *Device* class represent different types of devices within the application. *DeviceReal* and *DeviceBool* represent simple devices, while *DeviceScenes*, *DeviceShutters* or *DeviceThermals* bring specific methods.

4.1.6 EPSNET

This folder contains classes for complete *EPSNET* protocol and communication over this protocol with the central unit of *iNELS* setup. [25]

EpsnetClient class

This class is the essential part for communicating over the *EPSNET* protocol. Instance of this class is directly used to send and receive messages from the central unit.

Once the instance is created, it is essential to call the *StartClient* method. This method sets the endpoint for communication and creates a new instance of the *SocketAsyncEventArgs* object, which is used for asynchronous network communication using *sockets*, and also source and destination addresses for *EPSNET* communication. Both timers are also initialized. The *timeOutTimer* serves for the time out of sent messages and its interval is set using the *TIME_OUT_INTERVAL* constant. This timer is only started after a message is sent and when time out occurs, the client socket is closed. The second timer is called *tasksTimer* and is used for processing requests to send messages. This timer's interval is specified by the *REFRESH_INTERVAL* constant. The *tasksTimer* is started before the end of the *StartClient* method.

When the *OnTasksTimerTick* method is started as a result of *tasksTimer* tick event and the client is not currently processing any task, the peak of the queue called *tasks*, which contains all scheduled tasks, is checked and when a task is present, it is started. In case the queue is empty, the active *room* is searched for *devices* and the reading of their statuses is queued using *eREADB* and *eREADN* methods for *devices* with *bool* values and *real* values respectively.

The execution of a task starts one of the methods such as *EpsnetCONNECT*, *EpsnetIDENT*, *EpsnetGETSW*, *EpsnetREADB*, *EpsnetREADN*, *EpnetWRITEB* or *EpsnetWRITEN*. These methods correspond with types of *EPSNET* messages as discussed in the chapter about *iNELS*. Each of these methods creates an *EpsnetUDPPackage* with relevant *EpsnetMessage*, sets the handler for *EpsnetCompleted* event as discussed later and starts sending it by calling the *ConnectAsync* method, which setups the asynchronous socket and starts the *timeOutTimer*.

After every ansynchronous socket operation is completed, the event is handled by *Operation_Completed* method. This method checks the type of the completed operation and calls one of the *SendAsync*, which sends the actual data asynchronously, *ReceiveAsync*, which receives data asynchronously into a buffer, or *ReadClose* methods.

The *ReadClose* method stops the *timeOutTimer* and tries to get data from the buffer. Also, the socket is closed so that it can be reused again. When data is copied, the *OnEpsnetCompleted* method is called, which invokes any associated handler and passes the data as an array of byte values. The handler to be invoked is always set in advance by the calling method such as *EpsnetCONNECT* and other methods mentioned earlier. Handler methods are named in the same manner as the calling methods with the *_Completed* suffix, such as *EpsnetCONNECT_Completed*. These methods provide

textual feedback, or in cases like the *EpsnetREADB_Completed* and *EpsnetREADN_Completed* can directly set values of relevant devices. Before each of these methods is ended, the *Completed* event is fired by calling *OnCompleted* method with a string parameter. This is useful as by handling this event, we can monitor the incoming data.

EpsnetClientEventArgs class

This class defines two types of parameter for event arguments, which are used within *EpsnetClient*. The *EpsnetResult* provides data in an array of bytes, while the *Message* can be used for passing a string value.

EpsnetMessage class

The *EpsnetMessage* class resembles the structure of *EPSNET* messages as described in the chapter about *EPSNET* protocol. All common constants are defined such as start delimiters *SD1* and *SD2* or end delimiter *ED*. Properties such as the *length (LE)* and *frame check sum (FCS)* are automatically calculated from the object content. The *Message* virtual property is supposed to be overridden in child classes and its purpose is to return an array of bytes with relevant structure based on the type of *EPSNET* message.

EpsnetMessage child classes

Classes such as *EpsnetMessageCONNECT*, *EpsnetMessageGETSW* and *EpsnetMessageIDENT* only consist of two parts – the constructor and the overridden *Message* method. The constructor is simple and takes only source and destination *EPSNET* addresses. The *Message* method returns the complete *EPSNET* message of a specific type. *Messages* such as *EpsnetMessageREADB*, *EpsnetMessageREADN*, *EpsnetMessageWRITEB* and *EpnsetMessageWRITEN* are more complex as they contain two constructors. One constructor accepts *ExportPubItem* object directly and the other allows whole collection of *Device* objects to be supplied.

EpsnetUDPPackage class

This class is responsible for constructing complete *EPSNET* packages that can contain multiple *EPSNET* messages. The constructor accepts the package counter and array of *EpsnetMessage* objects. The structure is described in the *EPSNET* protocol chapter. The data represented as an array of bytes can be obtained by accessing the dynamic *Package* property.

4.1.7 XMLRPC

Classes in this folder serve for the communication over *XML-RPC* protocol [14] with the *iNELS* server. This protocol is described in the *iNELS* chapter.

XMLRPCClient class

This class is represents the communication client over *XML-RPC* protocol. Its constructor needs to be passed a string representing the URL of the server to connect to and also Dispatcher object, which is used for invoking methods on the UI thread and *NavigationService* object, which can be used for navigating from the current UI page. Within the constructor, new instance of the *XMLRPCCLientProxy* class is created. This server as a mediator for the communication and is described later in this chapter.

The client can be started by calling the *pingStart* method. This method calls the ping method of the proxy object and if the result is a true value, the *refreshTimer* is initialized and started. Otherwise, the navigation service object is used to navigate to *PlacesPage* in order to allow the user to choose a place with different configuration as the server could not be reached using the current one. The *refreshTimer* is set so that on the Tick event, the *readAll* method is called. The *readAll* method is used to obtain values of all devices from the currently selected room. The method processes the collection of devices and calls the read method, passing all relevant device names within a string array.

The read method calls the *proxy.read* method, passing the device names and also and specifies the required *AsyncCallback* object as *lambda expression*. This approach is used among the whole *XMLRPCClient* class. Within the declared callback, the *UIDispatcher* object is used to invoke an anonymous delegate on the UI thread. This delegate consists of converting the result of *proxy.general_completed* method to an *XmlRpcStruct* object, which is a type object define within the *XML-RPC.NET* library [14] and provides a way of storing key-value dictionary-like recursive objects. For each key, which represents a device by its *iNELS* name, we take its value and update the main collection stored within *App.places* and respective room based on searching for the device with given name. The concept of using *ObservableCollection* objects together with *INotifyPropertyChanged* based properties takes care of updating the UI.

Other methods work in similar way. The *writeValues* method accepts an *XmlRpcStruct* object consisting of key-value pairs, where key is the device name and value is the desired value to be set. If only one device is supposed to be set, the *writeValue* method can be used for creating the *XmlRpcStruct*. The method to be called after response a response from server is obtained is *XmlRpcStructDelegate*. This method serves for general data processing and can also handle null results. The *showMessage* parameter indicated, whether the user should be informed about the result within user interface. If the result is not null and messages should be presented, list of key and value pairs is shown. We use this method for null type results and basic result listings.

There are also other methods such as *playIfPaused*, *pause*, *volumeUp*, *volumeDown* or *eManTotalSumsAndPrices* and also other methods for processing results. However, as these belong to multimedia and energy manager extensions, they will be described later.

XMLRPCClientProxy class

This class is used as a proxy for the communication with over *XML-RPC* protocol and is required by the *XML-RPC.NET library* when using *Windows Phone* platform. As stated in the documentation [14]:

"For cases where a manually implemented proxy is required, such as with Windows Phone, it is possible to make asynchronous calls via the BeginInvoke and EndInvoke methods of XmlRpcClientProtocol.

In both cases two new methods must be implemented in the proxy class for each XML-RPC method."

Therefore, each method used is defined separately:

```
[XmlRpcBegin("writeValues")]
public IAsyncResult writeValues(AsyncCallback acb, XmlRpcStruct devNameAndValue)
{
    return this.BeginInvoke(MethodBase.GetCurrentMethod(), new object[] { devNameAndValue }, acb, null);
}
```

Figure 4.1 – A sample of XmlRpcBegin marked method.

However, most of *XmlRpcEnd* methods can be merged under one definition as the result can be passed in type of common object and casted and processed later:

```
[XmlRpcEnd]
public object general_completed(IAsyncResult iasr)
{
    return this.EndInvoke(iasr);
}
```

Figure 4.2 – A sample of XmlRpcEnd marked method.

Methods defined within the proxy class correspond to those defined in the *XMLRPCClient* class and mostly use the exact same names.

4.1.8 ExportPub

For parsing and working with the export.pub configuration files, which is described in the *EPSNET* section of *iNELS* description, two classes were implemented.

ExportPubItem class

Class is representing a single device, which corresponds to a single line of the configuration file. The constructor accepts a string and tries to construct the whole object. Firstly, the line is split by spaces which are present in the line string. As some configuration files have spaces on the beginning of some lines this issue must be fixed as well.

Subsequently, according to the structure of export.pub configuration file, each substring is handled individually. The first substring corresponds to the type of register. There was a custom enumerable type created called *REG_Options* with values corresponding to the possible values supplied in the substring. Second substring is matched in a similar way and this approach is also used for most of the following values such as *TYPE, PUB_INOUT* or *CF* properties. Also, in each enumerable type, there is a value of *UNKNOWN*, which is assigned when no match could be found.

For items of type *REAL* and *BOOL* have slightly different structure, if the *.B* substring is found, an offset is set so that all other values are read from the correct positions. The constructor can throw an exception if format is not valid.

ExportPub class

This class represents the whole configuration extracted from *export.pub* file throughout the application. The class holds a dictionary of *ExportPubItems*, which can be accessed by their names. If there are more items with same name supplied within the configuration file, the next item is ignored. Initially there was an exception thrown, but in order to provide better user experience, this exception is being ignored, so that the user gets to control as many items as possible even when the configuration file is not completely without minor errors.

The constructor of this class needs to be provided with *isolatedStorageFile* instance and also the name of the configuration file located within the isolated storage. Custom *FileUtilities.GetFileReader* method is then used to get the reader of the file and each line is subsequently supplied to the *ExportPubItem* class for processing.

4.1.9 RoomsCfg

The RoomsCfg consists of multiple classes that help with the deserialization of rooms.cfg configuration file, which uses XML format and is described in the chapter about iNELS.

RoomsCfg class

The RoomsCfg class is the basic class, which represents deserialized data of rooms.cfg file. It holds only one property which is of type Rooms. The constructor of this class must be supplied with isolated storage file object and also the name of the configuration file, which is usually rooms.cfg. Using FileUtilities.GetFileReader method and XmlSerializer.Deserialize method, the complete structure of Rooms is constructed.

Rooms class

The Rooms class represents the root node from the configuration file and contains list of child rooms.

Room class

The Room class represents the actual room node from the configuration files and contains definitions for all devices that might appear under this node. An example of such device is:

```
// HeatControl: heat-control / heat control
[XmlArray("heat-control")]
[XmlArrayItem("item")]
public List<HeatControlItem> heatcontrol { get; set; }
```

Figure 4.3 – A sample of a device type.

The comment line describes the names of the device used in the XML configuration file and also within the iNELS control centre. The first parameter specifies that the item should be considered to be an array with the respective name and the second that the items of the array are called item. The last line defines the representation within the application, which should be a List of one of the custom classes defined later.

BasicItem class and other item classes

The BasicItem class is used for all items with node that consist of the inels name, column, row and read_only attributes and text representing the name of the device. For other types of items, custom classes are defined, such as HeatControlItem for heat-control, MeterItem for meter, OnOffItem for on_off, ScenesItem for scenes, ShuttersItem for shutters, ThermalsItem for thermals or ZonesItem for Zones.

4.1.10 Helpers

This folder contains several classes providing methods used across whole project. Among these classes, there are classes with common functionality as well as converters, described in the chapter about data binding.

ArrayUtilities class

This class only provides one static method called concatAll. This method is used within various constructors of classes derived from EpsnetMessage class. The method allows concatenating any number of supplied arrays of objects into one array and returning it back.

ConverterClassToTemplate class

This class is one of the converters that are used within this project. Based on the class of the calling object, the Convert method return a template which is defined within the RoomsPage resources. This converter is used on the RoomsPage to select according template for each displayed device of the selected room.

ConverterPlaceTypeToEnabled

This converter is used within PlacePage.aspx and is described in PlacePage chapter.

ConverterValueToOpacity class

This converter is used for converting provided value to adequate opacity values. For zero values, which correspond to a device that is turned off, the opacity value is 0.5, whereas for values greater than zero the opacity is equal to 1. This converter is used within templates for displaying devices, so that devices, which are off are not as bright as those that are turned on.

ErrEventArgs class

This class is based on EventArgs and is used for handling errors across the application. These error arguments only contain one filed for a string error message.

FileUtilities class

FileUtilities class is essential for working with configuration files within our application. It provides two public methods DownloadTwoFiles and GetFileReader. DownloadTwoFiles serves for downloading two files one by one. The method must be passed the URIs and names under which the files should be stored, IsolatedStorage instance and also handlers for ready and error events. Downloading files is performed using asynchronous WebClient calls and files are stored within IsolatedStorage of the application. The GetFileReader method returns a StreamReader object for a file from isolated storage, specified by its name.

InitClient class

The InitClient class provides two static methods that are used for downloading and parsing configuration files. The DownloadFiles method calls internaly the FileUtilities.DownloadTwoFiles method. The GetFiles method return ExportPub, RoomsCfg and PivotRoomCollection objects based on the provided file names of configuration files.

4.2 Application's pages

In this section, we will describe the main application's pages. These are the main user interface parts that are presented to the user. There are also other pages and UI element within the application, but only pages from the basic application's functionality will be described in this chapter. The rest will be described in following chapters separately, as they belong to application's extensions and are therefore not part of the its core.

4.2.1 Main page

Our application only supports primary tile, which can be pinned to the Start screen. This tile is static and launches the application.

The *MainPage* is the default page of the application, which is navigated to upon launching. It consists of a title of currently active place, tiles that navigate to *RoomsPage* and various extension pages and also an application bar. The application bar is semi-transparent, so it does not take screen space and it holds two buttons – one for accessing the list of places (*PlacesPage*) and the other for navigating to settings. When a user navigates to the *MainPage* and there is no active place selected, the selected place has invalid configuration or server is not accessible with current network connection, he is automatically navigated to the *PlacesPage*, so that he could choose a different place configuration to be used.

Figure 4.4 – Start screen tile and main page of the application.

4.2.2 Rooms page

The *RoomsPage* is the most important part of our application. The page presents all rooms and devices that belong to the currently selected place within a Pivot element. The title of the page consists of the active place name and its connection type, while the title of each *PivotItem* is the name of the room. Within the content, devices are listed in form of icons. The *ListBox* uses *WrapPanel* element from the *Windows Phone Toolkit* as its *ItemsPanel* to enable advanced arranging options. Each icon type is defined as *DataTemplate* within the page resources. The types used within our application are *IconBool, IconReal, IconShutters, IconThermals, IconScenes* and *IconZones*.

IconBool is the basic type and represents a device, which can be turned on and off by a short tap on the icon. The icon consists of a *StackPanel*, *Rectangle*, *ImageBrush* and a *TextBlock*. The *StackPanel* wraps the icon and its *Tag* property is directly bound to the respective device object, its Tap property is set to *IconTap* method, *Background* property is got from the *PhoneAccentBrush*, which is the color selected by user in OS settings, and also *TiltEffects* is used, which provides a more responsive behavior of the icon. The Rectangle holds the icon image, which is set as *OpactityMask* using *ImageBrush*, whose *ImageSource* property is bound to *ImagePath* property of the device and the *TextBlock* is then bound to the Name property of the device. Upon the tap action, an animation is started to fade out and fade in the icon and also corresponding method is called based on the type of connection used. The animation is defined within page resources as a *Storyboard* with *DoubleAnimation* and *CubicEase* easing function. It animates the opacity of the icon,

The *IconReal* is very similar to the *IconBool*, however, it adds a *ProgressBar*, since these devices using this type of icon support progressive turning on and dimming. Also on *Hold* event, a pop-up control is displayed allowing changing the value of the device. While changing the value, timer is running to ensure that values are sent to the server every five hundred milliseconds. The method used is also different for each type of connection. Other types of icons behave in a similar way, although some do not have the *Tap* event enabled.

This page also includes application bar with *Places* and *Settings* buttons, which is minimized by default to provide as much screen space as possible for the icon listing. The *RoomsPage* contains instances of all the clients it can communicate with. In our application the *EpsnetClient* and *XMLRPCCLient* are used.

When navigated to the page, firstly, *App.places* is checked and when no data is present, the user is navigated to the *Places* page is a similar way to the *MainPage*. Also, handler for *SelectionChanged* event is set to notify adequate client about the room change. In this method, we also check the type of active place and start corresponding client by calling either *StartXMLRPC* or *StartEPSNET* method. Lastly, title is set to the name of room and also application extensions such as the *Speech* extension and *Reminder* extensions are handled by checking the *querystring* and searching for the device to be turned on.

The *StartXMLRPC* method has to check the *App.places* again in a more specific way. The *activeItem* must be set as well as the collection of rooms, which also must not be empty in order to start the client. Otherwise, user is once again navigated to the *PlacesPage*. The client is started based on the data contained within the active place by creating new instance of the *XMLRPCClient* class and calling the *pingStart* method, which tests the connection. If successful, the client starts periodically refreshing the state of all displayed devices.

The *StartEPSNET* methods works in a similar way, but creates an instance of *EpsnetClient* class and also sets handlers for events of the *EpsnetClient* object such as *Completed*, *Error* or *Log*. When navigating away from this page, all active clients are stopped.

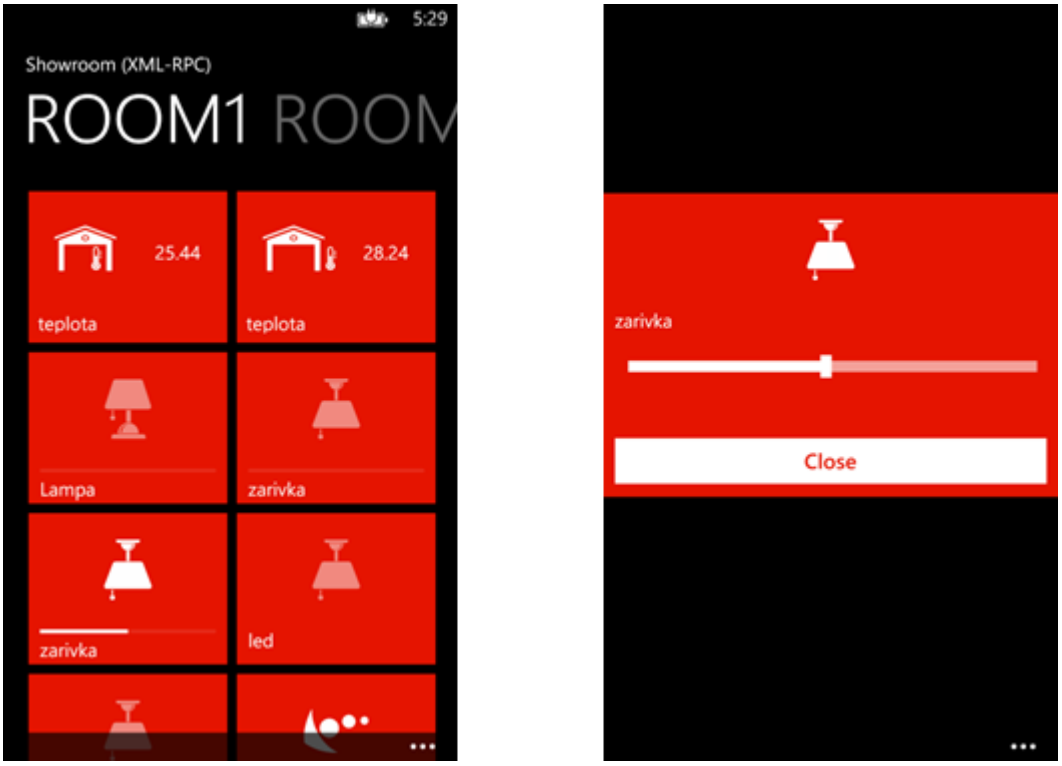


Figure 4.5 – The RoomsPage and pop-up for progressive value control.

4.2.3 Places page

The *PlacesPage* serves for selecting the active place, which defines connection to the server and also rooms with devices to be shown. The page lists places that can be selected, enables the user to delete or edit a place upon holding a finger longer on one of the places and also provides an application bar for adding a new place.

When navigating to the page, places are checked within *App.places* and if none are present, settings are loaded from the isolated storage settings. If there are no places within isolated storage settings either, new set of settings for the application is created. When navigating from the page, places are stored within the isolated storage settings.

When selecting a *place*, *active place index* is changed and user is navigated back. After pressing the application bar add button, user is navigated to the *PlacePage*. Meanwhile, new place is created and its index passed in the *querystring*. The Edit function works similarly and Remove simply removes the place from collection.

The *PlacePage* enables users to edit the properties of a place and also download the configuration files. Properties such as *Place name*, *Host*, *Port* and *ID* are two-way bound; the *Place* type uses *ConverterPlaceTypeToEnabled*, which sets the radio button according to the *Place type* property string. The *Download configuration* button gets configuration files from the public server based on ID specified and processes them into *pivotRoomCollection*, which is then stored within *App.places* and also *isolated storage settings*.

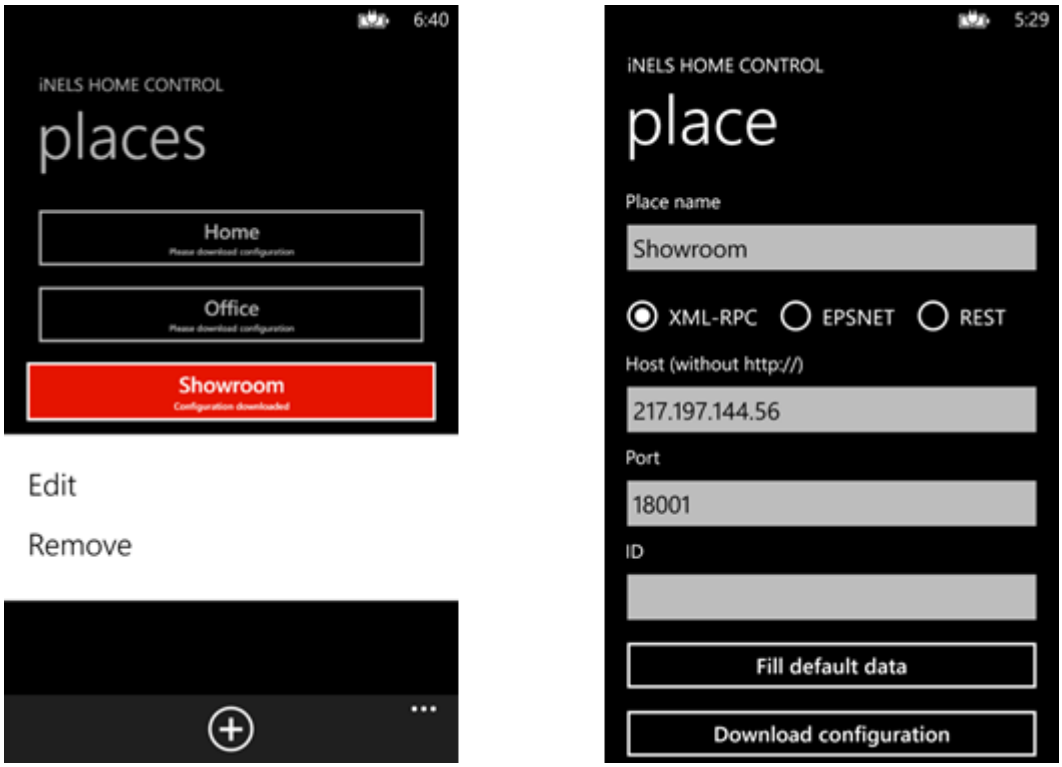


Figure 4.6 – List of places and place editing.

4.2.4 Settings page

As the configuration of places was moved out of settings, the *SettingsPage* itself currently only consists of three buttons.

The *Clear Isolated Storage* button calls the *IsolatedStorageSettings.ApplicationSettings.Clear* method. This method clears settings, not files stored within *isolated storage*. Clearing these settings enables the user to put the application into a state similar to the one when installed without the need to reinstall the application from the *Phone Store* or via developer kit.

Other two buttons are related to the *Speech extension* of this application. The first registers the command set by calling *VoiceCommandService.InstallCommandSetsFromFileAsync* and supplying the *RoomsVCD.xml* file, which is described later. The other enables the user to manually update the voice command definition. As the *Speech extension* only works with a specified room, the function looks for that room in *App.places* and updates the command set phrase list with list of devices from this room by calling the *VoiceCommandSet.UpdatePhraseListAsync*.

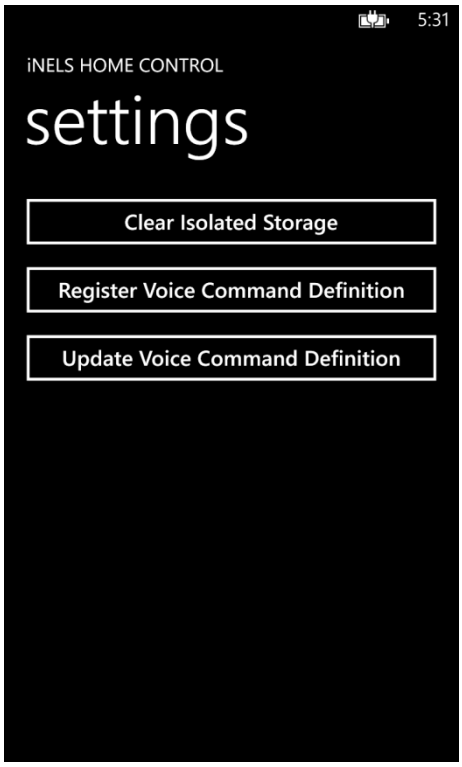


Figure 4.7 – The page with application's settings.

4.2.5 Extension pages

As mentioned earlier, other pages, which are directly related to the application's extension functions, such as *CameraPage*, *EManPage*, *MediaPage*, *MielePage*, *PhotoPage* and *ReminderPage*, are discussed separately in the following section.

4.3 Extensions and suggested features

In this section, we will discuss extensions that can provide more functionality to the application than controlling standard *iNELS* devices. In the first part of this section, we will discuss extensions that were implemented into the user interface; in the second section, we will focus on possible features that would create added value to the application, but would need further cooperative development with *iNELS* server.

4.3.1 Multimedia

The multimedia extension is implemented within the *MediaPage*. It consists of buttons that launch media methods such as *playIfPaused, pause, volumeUp, shuffle* and others and also of three fields for entering text. This test environment can communicate with the *iMM* server over methods specified in *XMLRPCClient* (and *XMLRPCCLientProxy* respectively). Two first fields are used for setting the parameters of connection, which can be obtained by using the *getPlayersList* button. Not all functions are meant to be used with all types of devices, so in the graphical user interface of possible multimedia extensions, different types of device will need to be shown separately.

The multimedia extension also has to be integrated with zones that are obtained from the *Rooms.cfg* configuration file. This way, media devices can be assigned to certain rooms. However, by having the multimedia separately, the user experience might become better.

Figure 4.8 – The multimedia extension page and list of connected players.

4.3.2 Voice commands

Voice commands can be a very useful feature within *Windows Phone* applications, especially for home automation. Not only can a user start the application, but also more sophisticated commands can be constructed as described in the *Speech* chapter. This way, it is even possible to control single devices within a room.

Our application has been provided with such functionality, where devices from the sample room can be controlled using voice commands. It is important to download configuration for showroom and also register voice commands and update definition at the *SettingsPage*.

Subsequently, the voice command feature can be invoked by holding the *Start button*. Once the *listening screen* is shown, the command if form specified within the *RoomsVCD.xml* file can be used. The format is like: *Home control, turn on, device_name*. *Home control* is the name of the application for voice command purposes. For testing purposes, only the turn on action is used. Device can be any of those that are present within the *ROOM1* of *Showroom*.

After saying the phrase, our application is launched, user is automatically navigated to the *RoomsPage* and device passed in the *querystring* is looked up and turned on.

The voice command functionality could be also extended by using speech within the application itself, but from our point of view, the voice command feature is more useful at the beginning, as it can also be used in a car's handsfree for opening a garage or unlocking the house even before exiting the car itself.

Figure 4.9 – The voice command initialization from the Start screen.

4.3.3 Reminder

The *reminder* extension allows users of our application to set an action, such as turning on lights, to be executed in the future. As *Windows Phone* multitasking model, which is described in the Multitasking chapter, is not able to provide a scenario, where the task would be executed automatically, the feature was implemented using reminders. After setting a device from the sample *ROOM1* and selecting the duration in minutes, the reminder is set using *ScheduledActionService.Add* method. Title, content, time of the action and also page to which the application should navigate (including *querystring*) can be set within the *Reminder* object. The application can be exited and when the time comes, reminder is shown to the user with default alarm sound. After tapping on the reminder, the application is launched and navigated to the *RoomsPage*, where the preprogrammed action is launched in a similar way to the voice command feature.

Also scenes can be launched this way and custom slow process of turning devices, which support continuous turning on could be added. Another approach would be using other forms of execution; however, none seems to be much more suitable for this scenario.

Similar approach could be used for combining this feature with applications that function as alarm clock with sleep monitoring features. Those could benefit of this approach and instead of playing sounds, it could be interesting to slowly open blinds and play music from the home audio system in the morning, or automatically turn on the coffee machine.

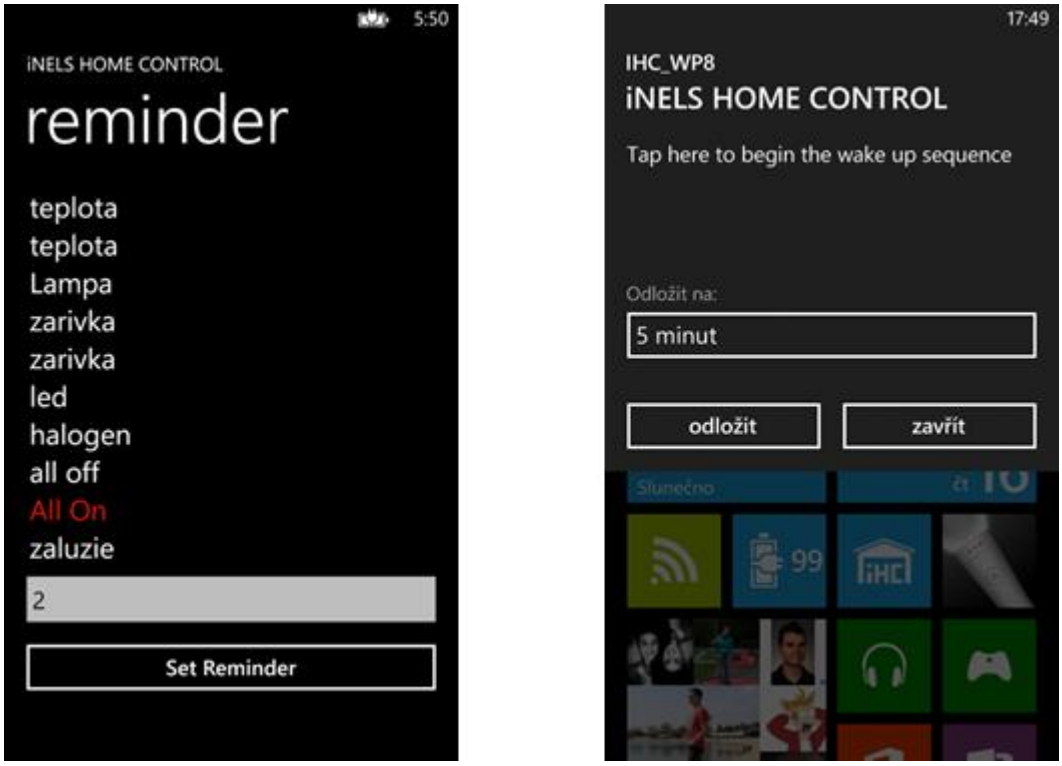


Figure 4.10 – The reminder setup page and the reminder notification.

4.3.4 Intelligent camera

Intelligent camera feature provides an experimental way of adjusting the lighting conditions automatically in intelligent buildings. Our feature uses the device's camera and based on the average lighting, can turn lights in a room on and off dynamically. This can be used in situations, where the room's lighting conditions are not sufficient enough for taking pictures. The device can then tell lights to slowly turn on, so that higher quality pictures can be taken.

PhotoPage is the sample implementation of this feature. After pressing the *Go* button, a timer is started, which processes the camera's image. *GetPreviewBufferY* method is used to obtain the image from camera. Only luminance data is necessary for this application. Average is then calculated and if the level is below or above empirically obtained thresholds, lighting is changed.

This feature could also be integrated into a Lenses application, which is described in the Camera chapter. This way the application could be launched directly from the *Camera* built-in application and provide light-adjusting capabilities in more user friendly way.

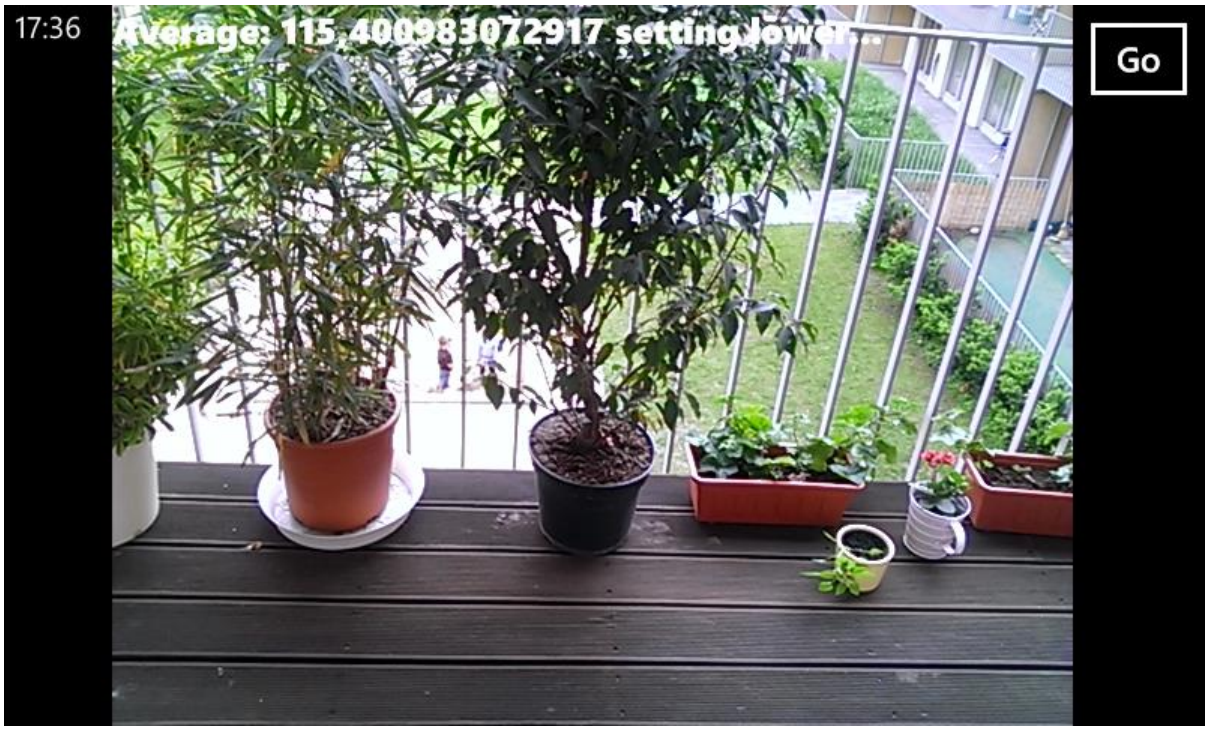


Figure 4.11 – The user interface of the intelligent camera extension.

4.3.5 Wi-Fi identification

Another extension to our application is Wi-Fi based place identification. In the *WifiPage*, we provide a code for the application to identify, whether there is active Wireless connection. This is done using the information from *NetworkInterfaceList* object. This object includes information about all network connections, so by choosing those that are of type *Wireless80211*, subtype *WiFi* and are connected, we can obtain the name of the currently active Wi-Fi connection, as two cannot be active at a time.

Using this approach, places within our application could be extended to include the Wi-Fi connection name and upon launching the application, we could check the active connection name and change active place if matching name is found. This could greatly increase user experience, as in most scenarios for end users; only one server (and also place) will be associated with one network.

For final implementation, however, further functionality would have to be implemented for scenarios, where more places include the same network name. For those scenarios, list of those places should probably be shown.

Figure 4.12 – The Wi-Fi identification page.

4.3.6 Accelerometer

As most *Windows* Phone devices provide accelerometer support, we implemented its support within the application. The sample implementation can be found on the *MotionPage*. This page mimics the behavior of devices with values of type real, which can be smoothly controlled. Using the accelerometer, when device is turned around the *X axis*, the *progress bar* values can be changed. This behavior is achieved using *Accelerometer* object and handling the *CurrentValueChanged* event. Within the handler, we check the *SensorReading*.Acceleration value of the *X axis* and update the *progress bar* values accordingly. For better user experience, threshold value is set, so that the user does not have to hold the device steadily.

Apart from using this feature in the way presented, *accelerometer* could also be used for switching between single rooms, or zones within multimedia. However, since these functions use the *Pivot* UI element, which is standard element for *Windows Phone*, users would have to be notified about this inconsistent behavior. Other possible use could be to use the shake motion to turn on or off all devices within active room.

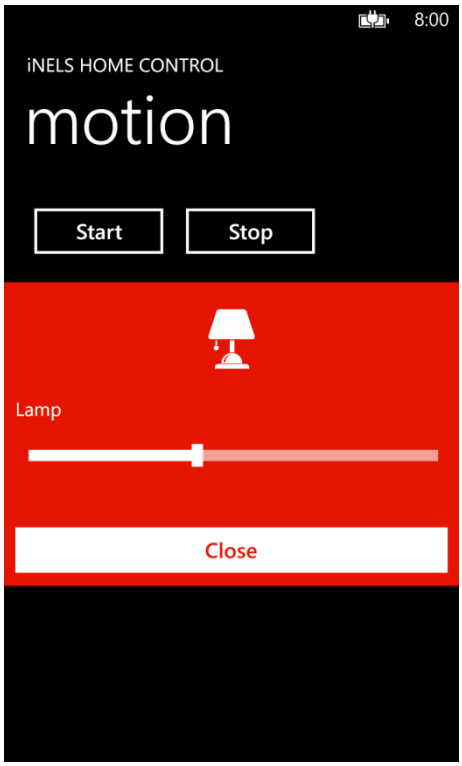


Figure 4.13 – Example of the accelerometer usage for controlling a progress bar.

4.3.7 Energy manager

This extension enables the user to get information about energy manager readings. We have implemented a *Pivot-based* page, which displays information of the usage of energy, water and gas consumption.

The page creates an instance of the *XMLRPCClient* class and uses energy manager related methods from that class. These methods include the *eManTotalSumsAndPrices* for all-time data, *eManTodaySumsAndPrices* for today's data and also respective methods for reading week, month and year data. As some of these methods require the current year, month or week as a parameter, the *DateTime.Now* is used.

For obtaining the current week, however, more complicated approach needs to be taken. Current week can be obtained using the *CultureInfo.CurrentCulture.Calendar.GetWeekOfYear* method supplied with *DateTime.Now*, *System.Globalization.CalendarWeekRule.FirstDay* and *DayOfWeek.Monday* parameters. This should, however, be changed for other cultural settings.

As another extension to this feature, graphs could be added. In the time of implementing this feature, however, the *Silverlight Toolkit*, which contains charts and graphs, could not be used for *Windows Phone 8* projects.

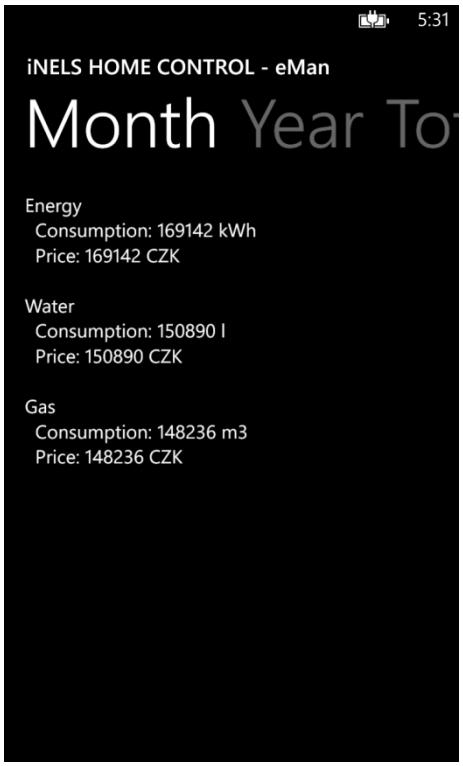


Figure 4.14 - Energy manager user interface.

4.3.8 Additional extensions

Miele devices

Our application supports reading the list of *Miele* [26] devices connected to the server. These devices can be of various types such as washing machines or dishwashers. However, more functionality can be added, such as reading the action that can be performed for each device. In our application, *MielePage* is the sample implementation. It uses *getMieleInfo* method from the *XMLRPCClient* class.

Online data

As mobile devices are usually connected to the internet, either using Wi-Fi or using the cellular network, they have access to various sources of information. In our application, we could benefit of the fact, that devices also feature GPS. We could, for example, extend our place definition to include GPS position information and use online data source, such as weather forecast from sources as yr.no, which provide weather data service. Combining these functions, we could obtain weather forecast for our place (defined by GPS) and based on the data, we close blinds or disable garden watering in case it is going to be rainy.

Near field communication

Within our application, we could also make use of the *proximity API*, which is described in the *Proximity* chapter. A great scenario would be sharing place definitions with other phones by touching devices. This would enable new phones to be added easily as control devices, without the need for manual configuration.

Secondary tiles

As our application uses tile-like environment, it would make sense, to make use of the new feature of *Windows Phone* platform – the *secondary tiles*. Using *secondary tiles*, users could select any device or *scene* and pin them directly to their *Start screen*. This would greatly simplify the process of turning on or off the most used devices. After tapping the secondary tile, user would be redirected to the *RoomsPage* and action would be performed automatically like in the *Speech* and also *Reminder* case.

Push notifications

The application now uses polling method to get states of all devices. Using the *Push notifications* feature, as described in the chapter *Push notifications*, could make the network communication much more effective. Using *raw messages*, the application could be informed directly and using the *tile update messages*, *secondary tiles* could reflect the real state of the device they represent. This feature, however, would require setting up a custom server, which would create such notifications for all registered devices and communicate with the *Microsoft Push Notification Service*.

IP Camera control

The *iMM* system also supports IP cameras. Their configuration is provided via *iMM* server, but the stream is access directly. So far, all attempts to capture the *MJPEG stream* using available libraries failed on the *Windows Phone* platform. Future research needs to be done in this field and alternatively, custom implementation of the streaming functionality must be created. This will, however, require longer period of time.

Intelligent action predictions

Our application could also make use of intelligent prediction of user's actions. In order to achieve such functionality, our application would need to be extended with action logging features. For each action, current state, desired change, time and also possibly other features such as connection type and GPS position could be logged. Based on analysis of the log file and current conditions (again time, state, …), actions that the user might want to perform would be presented on the *MainPage* for example, to be quickly accessible. Actions could be selected based on all the logged information and also be rated by the count of successful predictions, i.e. how many times they were actually executed when offered to the user.

Appointments integration

Since *Windows Phone* supports read access to the calendar appointments, our application could be also extended with similar functionality to the one used for setting ringtone profiles on some mobile phone platforms. This scenario also counts on some of the in background running features. This way, our application could check the upcoming appointments and perform set actions.

Device-based presence

When the application is launched on a device and connected to the server, we could assume that the user of that device is located at the same position as the device itself. Therefore, we could extend this assumption and merge the device and its user. This would allow scenarios like locking the home automatically when no more devices (users) are present within the wireless network. Also adequate scenes could be launched when users were leaving or coming back home. This could feature functions such as automatic light switching, securing the building or setting the heating system.

5 Conclusion

The aim of this thesis was to examine the *iNELS* intelligent electrical installation and *Windows Phone 8* platform and also to develop an application, which would be able to control the *iNELS* intelligent electrical installation using the *Windows Phone 8* platform-based device.

The application we have developed can control various *iNELS* devices by communicating with the central unit over the *EPSNET* protocol, which was fully implemented. Also, we have implemented the ability to communicate with the iMM server over XML-RPC protocol. For this communication, the XML-RPC.NET library was used. This communication method proved to be much more efficient and reliable.

The basic functionality has been tested on multiple configurations. However, in order to publish the application, more testing will be necessary, so that all scenarios are covered. So far, there were no performance issues even on the *HTC Windows Phone 8S* [22], which has low hardware specifications comparing to other available devices.

Apart from controlling *iNELS* devices, we have also implemented several extensions to our application. These extensions are proof of concept of various functionalities and give an indication of the possible future development and some of these extensions would certainly create added value to the final production application. Among these extensions, there are: *voice commands*, which enable the user to turn on a device from start screen; *reminder*, which enables users to set up turn on action for a specific time; *intelligent camera*, which enables adjusting lighting using device's camera; *wi-fi identification* to select appropriate place to be controlled based on the *SSID* of connected wireless network; *accelerometer*, which provides alternative way of setting values; or *multimedia* and *energy manager* sample pages.

As discovered during the implementation, *Windows Phone* platform has several limitations regarding deep feature integration within the OS, especially when it comes to multitasking. Even though relatively wide range of multitasking scenarios is provided, for more advance functionality, the full multitasking experience would be beneficial. Another limitation is the impossibility to communicate with self-signed servers. This is a limitation made for security reasons and requires changes on the server side of *iMM* system. Either non-secured communication can be used (as in our scenario) or the certificate must be verifiable.

As future development, the application will be extended with additions discussed in the previous chapter, such as *Multimedia* and *Miele* devices control or *Secondary tiles*, as well as with other features of the *iNELS* home control system like controlling IP cameras and air conditioning. Some of those features require further testing and also more reference *iNELS* setups in order to be published.

Bibliography

[1] WHITECHAPEL, Andrew, MCKENNA, Sean. *Windows Phone 8 Development Internals*. S.l.: Microsoft, 2013. ISBN 0-735-67623-2.

[2] NATHAN, Adam. *101 Windows Phone 7 Apps:Volume I: Developing Apps 1-50*. Indianapolis: Sams, 2011. ISBN 978-0-672-33552-5.

[3] WILDERMUTH, Shawn. *Essential Windows Phone 7.5: Application development with Silverlight*. Upper Saddle River, N.J: Addison-Wesley/Pearson Education, 2012. ISBN 03-217-5213-9.

[4] PETZOLD, Charles. *Programming Windows Phone 7*. Redmond, WA: Microsoft, 2010. ISBN 978-073-5643-352.

[5] FARRACCHIATI, Fabio Claudio, GAROFALO, Emanuele. *Windows Phone Recipes: A problem-solution approach*. 2nd ed. New York: Apress, 2011. ISBN 978-143-0241-379.

[6] LEE, Henry, CHUVYROV, Eugene. *Beginning Windows Phone App Development*. 3rd ed. New York, N.Y.: Apress, 2012. ISBN 978-143-0241-348.

[7] CAMERON, Rob. *Pro Windows Phone 7 development*. Berkley, CA: Apress, 2011. ISBN 978-1-4302-3219-3.

[8] SHENGWEI, Wang. *Intelligent Buildings and Building Automation*. London: Spon Press, 2010. ISBN 0-203-89081-7.

[9] RILEY, Mike. *Programming Your Home: Automate with Arduino, Android, and Your Computer*. Dallas, TX: Pragmatic Bookshelf, 2012. ISBN 978-1-9343-5690-6.

[10] JANOUŠEK, Vladimír. *Intelligent systems: Intelligent Buildings Lecture Notes*. Brno, Brno University of Technology, 2013.

[11] ANDREWS, Keith. *Human-Computer Interaction: Lecture Notes*. Graz, Graz University of Technology, 2012.

[12] MINÁŘ, Michal. Creation of Multimedia Control System in GNU/Linux. Brno, Brno University of Technology, 2010.

[13] *Windows Phone Dev Center* [online]. [cit. 2013-02-20]. Accessible from URL: <http://developer.windowsphone.com>

[14] *XML-RPC.NET* [online]. [cit. 2013-03-14]. Accessible from URL: <http://xml-rpc.net>

[15] *The Windows Phone Toolkit* [online]. [cit. 2013-01-04]. Accessible from URL: <http:// http://phone.codeplex.com>

[16] Home automation. In: *Wikipedia: the free encyclopedia* [online]. San Francisco, CA: Wikimedia Foundation [cit. 2013-04-05]. Accessible from URL: <https://en.wikipedia.org/wiki/Home_automation>

[17] Windows Phone. In: *Wikipedia: the free encyclopedia* [online]. San Francisco, CA: Wikimedia Foundation, [cit. 2013-04-05]. Accessible from URL: <https://en.wikipedia.org/wiki/Windows_Phone>

[18] *Nová Prezentace společnosti ELKO EP 2013* [online]. Holešov. ELKO EP, s.r.o., 2013, [cit. 2013-05-15]. Accessible from URL: <http://www.elkoep.cz/downloads/promotion_materials/Prezentace_spolecnosti.ppsx>

[19] *iNELS BUS Systém Prezentace* [online]. Holešov. ELKO EP, s.r.o., 2012, [cit. 2013-05-15]. Accessible from URL: <http://www.elkoep.cz/downloads/promotion_materials/PREZENTACE_iNELS_2012_CZ.ppsx>

[20] *iMM Prezentace* [online]. Holešov. ELKO EP, s.r.o., 2012, [cit. 2013-05-15]. Accessible from URL: <http://www.elkoep.cz/downloads/promotion_materials/Prezentace_iMM_FINAL.ppsx>

[21] *Nokia* [online]. [cit. 2013-05-18]. Accessible from URL: <http://www.nokia.com>

[22] *HTC* [online]. [cit. 2013-05-18]. Accessible from URL: <http://www.htc.com>

[23] BENNET, Jay T. Windows Phone 8 Emulator runs as a virtual machine on Hyper-V. In: *Windows Phone Central* [online]. 2012 [cit. 2013-01-05]. Accessible from URL: <http://www.wpcentral.com/windows-phone-8-emulator%E2%80%99s-hardware-requirements>

[24] *NuGet* [online]. [cit. 2013-01-20]. Accessible from URL: <http://nuget.org>

[25] *Sériová komunikace programovatelných automatů Tecomat – model 32 bitů* [online]. Kolín. Teco a.s., 2012, [cit. 2013-01-15]. Accessible from URL: <http://www.tecomat.com/wpimages/other/DOCS/cze/TXV00403_01_Comm_Serial32_cz.pdf>

[26] *Miele* [online]. [cit. 2013-05-12]. Accessible from URL: <http://miele.com>

[27] *Wikipedia: the free encyclopedia* [online]. San Francisco, CA: Wikimedia Foundation, [cit. 2013-05-15]. Accessible from URL: <https://en.wikipedia.org>

[28] *Intelligent Building Dictionary* [online]. [cit. 2013-05-01]. Accessible from URL: <http://intelligent-building-dictionary.com>

List of attachments

Attachment A: Content of the CD

thesis.pdf	
thesis.docx	
src folder	A folder containing source code of the application.
doc folder	A folder containing supplied documentation.

Attachment B: The MIT License (MIT) for the XML-RPC.NET

Attachment C: Microsoft Public License (Ms-PL) for the Windows Phone Toolkit

This license governs use of the accompanying software. If you use the software, you accept this license. If you do not accept the license, do not use the software.

1. Definitions

The terms "reproduce," "reproduction," "derivative works," and "distribution" have the same meaning here as under U.S. copyright law.

A "contribution" is the original software, or any additions or changes to the software.

A "contributor" is any person that distributes its contribution under this license.

"Licensed patents" are a contributor's patent claims that read directly on its contribution.

2. Grant of Rights

(A) Copyright Grant- Subject to the terms of this license, including the license conditions and limitations in section 3, each contributor grants you a non-exclusive, worldwide, royalty-free copyright license to reproduce its contribution, prepare derivative works of its contribution, and distribute its contribution or any derivative works that you create.

(B) Patent Grant- Subject to the terms of this license, including the license conditions and limitations in section 3, each contributor grants you a non-exclusive, worldwide, royalty-free license under its licensed patents to make, have made, use, sell, offer for sale, import, and/or otherwise dispose of its contribution in the software or derivative works of the contribution in the software.

3. Conditions and Limitations

(A) No Trademark License- This license does not grant you rights to use any contributors' name, logo, or trademarks.

(B) If you bring a patent claim against any contributor over patents that you claim are infringed by the software, your patent license from such contributor to the software ends automatically.

(C) If you distribute any portion of the software, you must retain all copyright, patent, trademark, and attribution notices that are present in the software.

(D) If you distribute any portion of the software in source code form, you may do so only under this license by including a complete copy of this license with your distribution. If you distribute any portion of the software in compiled or object code form, you may only do so under a license that complies with this license.

(E) The software is licensed "as-is." You bear the risk of using it. The contributors give no express warranties, guarantees or conditions. You may have additional consumer rights under your local laws which this license cannot change. To the extent permitted under your local laws, the contributors exclude the implied warranties of merchantability, fitness for a particular purpose and non-infringement.