INTEGRATING AND TRANSFERRING LEGACY CODE TO .NET / MONO ENVIRONMENT ON MULTIPLE PLATFORMS.

Kamil Wojciech Szarek

Bachelor's Thesis
May 2011

Degree Programme in Information Technology
Software Engineering

JYVÄSKYLÄN AMMATTIKORKEAKOULU
JAMK UNIVERSITY OF APPLIED SCIENCES

JYVÄSKYLÄN AMMATTIKORKEAKOULU
JAMK UNIVERSITY OF APPLIED SCIENCES

DESCRIPTION

Author(s) SZAREK Kamil Wojciech	Type of publication Bachelor´s Thesis	Date 03.05.2011
	Pages 47	Language English
	Confidential ( ) Until	Permission for web publication (X)

Title
INTEGRATING AND TRANSFERRING LEGACY CODE TO .NET / MONO ENVIRONMENT ON MULTIPLE PLATFORMS.

Degree Programme
Information Technology

Tutor(s)
SALMIKANGAS, Esa

Assigned by
Upload Image Processing Oy

Abstract

Today it is very important that the applications are platform independent and sufficient. It was and still is very common that whenever some operations take part on server side they are created by a script language, like PHP, bash, or Ruby. This is a very convenient approach because a code can be easy modified, and the whole application does not require recompilation after that. What is more the results are immediately visible. Sometimes, however, when such an application has to perform heavy operations, those languages are not sufficient. At some point it might be very wise to consider the code’s migration to some other environment or language that can be run efficiently on almost every platform.

This Bachelor’s thesis shows the whole process of such a migration. The legacy PHP script is refactored from Linux environment to C# language and cross-platform Mono environment. The author presents all most important modifications and improvements, and also focuses on the encountered problems and further changes that can be done in the future.

Keywords
PHP, C#, .NET, mono project, image processing, FTP client, mail client

Miscellaneous

CONTENTS

FIGURES

ABBREVIATIONS

API Application Programming Interface

APS Active Server Pages

ASCII American Standard Code for Information Interchange

CLI Common Intermediate Language

CLR Common Language Runtime

COM+ Component Object Model

CPU Central Processing Unit

FTP File Transfer Protocol

GPL GNU General Public License

GUI Graphical User Interface

JIT Just In Time

LGLP GNU Lesser General Public License

MSIL Microsoft Intermediate Language

OS Operating System

PID Process Identifier

SDK Software Development Kit

SMS Short Message Service

SOAP Simple Object Access Protocol

UDDI Universal Description, Discovery and Integration

UI User Interface

WSDL Web Services Description Language

XML Extensible Markup Language

DEFINITIONS

iSync – name of the PHP legacy script which is the main subject in the thesis. Application with the same name is created in .NET environment.

1 INTRODUCTION

For many years new businesses and companies have grown and the infrastructure has grown according to the available supplies and existing demands on the market. Within all these years with the very fast electronics development, the same has happened with the software used by different companies. Nowadays it is very important to follow up new technologies in the IT sector. Buying and using the latest hardware, however, is very often not enough. What makes work easier is also the software, which within the years may become inefficient or just too old and that is why it can be incompatible with the modern hardware. In that case there are few solutions:

1. buying new software if there is an available one which meets all expectations and needs of potential user
2. creating a completely new software
3. recreating an existing software, for example by implementing it in a new environment.

Many companies, which own their own software do not want to buy or create a new one because it can result in using the other's company's licenses, which they do not want to do. In this case, the best solution is to adapt the old program to the current needs and this is what Upload Image Processing company decided to do.

The author's main task during a practical training was convert PHP script running on the Linux environment to the C# cross-platform Mono environment. The heart of the original script was a very long procedure that had to be split and re implemented in the new environment. The base routine used many technologies like: file managing, file transfer protocol, MySQL database connection, email client for sending notifications, and image processing. The large variety of technologies described above caused that the whole project work was divided into several stages according to the used technologies to keep the code clear and to easily manage the process of refactorization.

At the beginning the author had to familiarize himself with the whole old code, which was kept in four files to separate at least some parts of the code from each other. When the first stage was finished and the author was briefly introduced into the whole script, integration had to be started. However, it was impossible and unwise just to rewrite the whole PHP code to C# line by line; that is why that solution was not even taken into consideration. Instead of that everything had to be done precisely, carefully and almost from scratch based only on the functionality which PHP legacy code provided. First of all the knowledge about certain issue related to the project had to be collected. A very important phase was also creating documentation of the whole code because the original source code was not very well commented. The aim was to create software that would be compatible with the current hardware and would meet the client’s expectations. What is equally important in the future is that then code would be much easier to maintain and change by someone else. To reach this goal detailed documentation was needed as well as many descriptions and explanations how and why thing was done in that particular way.

This Bachelor’s Thesis shows the whole process of software migration between two different IT environments. Its main goal is to show how the functionalities should be properly separated and encapsulated into the components. The other very important subject taken up into this document is very clear explanation of how all needed information was gathered and how it was implemented.

The first part of this thesis is theoretical and is contains a short description of a routine which was used to perform the script. It also introduces briefly some technologies that are used by the original script, which are described more detailed later in each of the separate subsections. It aims at familiarizing the reader with the basic technologies, which allows further understanding of how the script works and how it is modified.

In the third chapter, the author discusses the whole process of implementation with usage of the new application’s structures of classes for better illustration of the task he has accomplished. Some of the classes were used in the simple test applications with graphical interface, which is also presented in this part of the thesis.

The task of re-implementation can seem line straight forward, only it requires a great deal of time and a workload, and most often there are many difficulties that each developer needs to face in order to create a fully working application. That is why chapter 4 presents exemplary problems that the author experienced when a script was re-implemented and the ways he resolved them.

2 THEORETICAL BASICS

2.1 Overview of the technologies related to the script

The figure below shows most of the technologies, techniques and a system related to the iSync script and they are described more detailed in the further chapters. What can be noticed here is that iSync relies mostly on the open source free solutions.

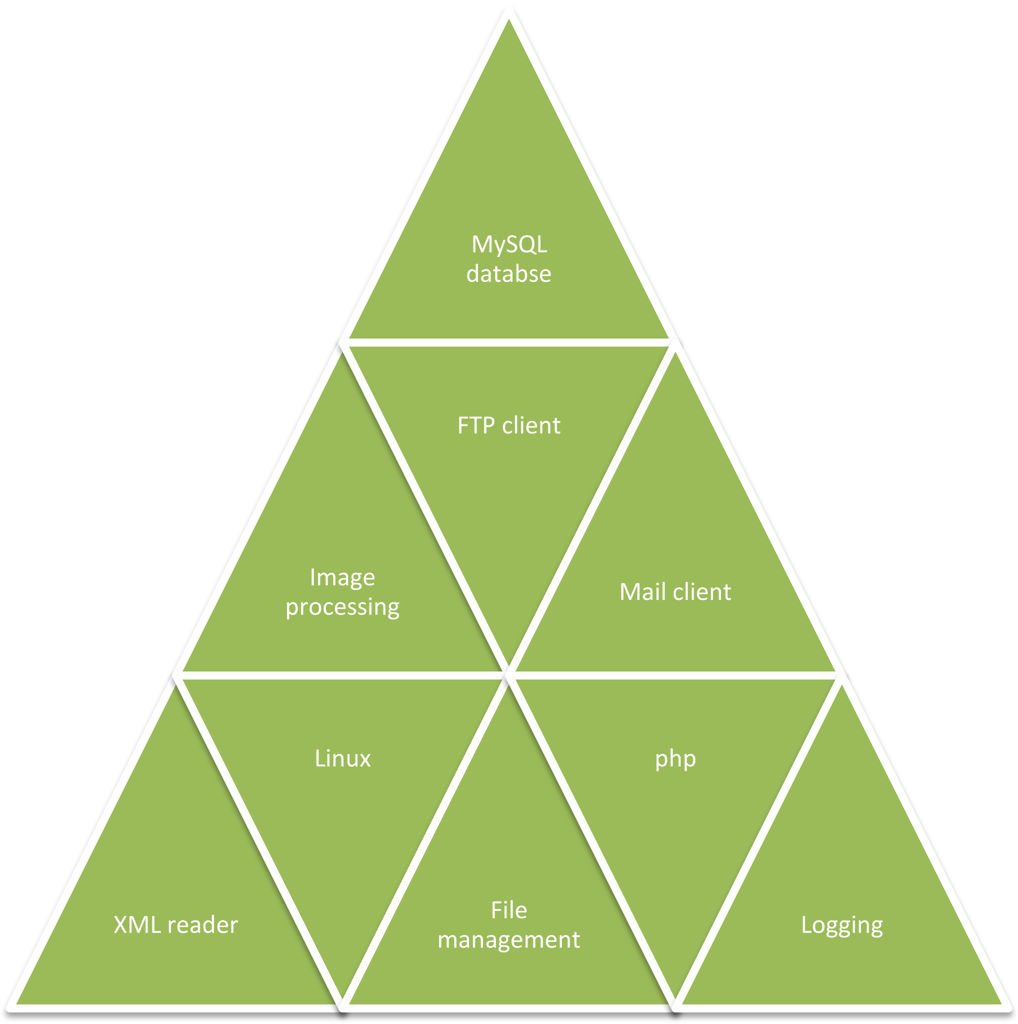


FIGURE 1. Technologies and techniques related to the script

The script is run under Linux operating system which is at the same time a web server. The FIGURE 2 below illustrates a routine of the script.

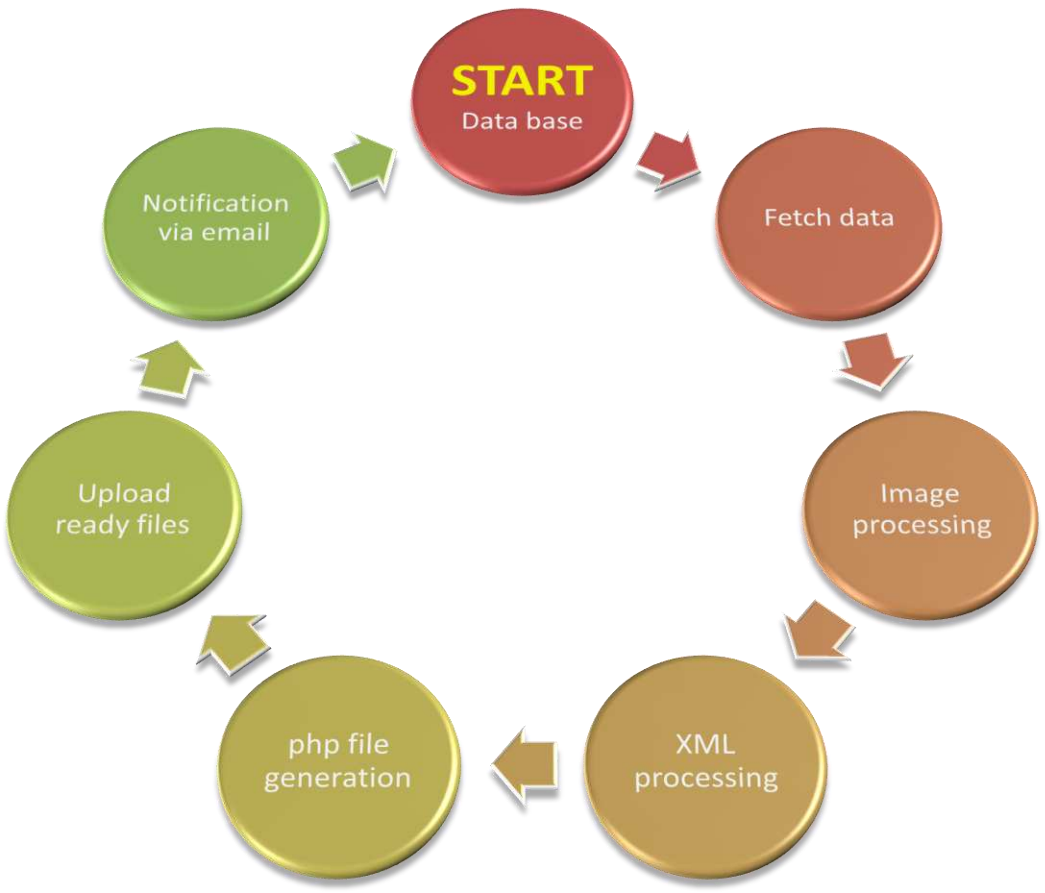


FIGURE 2. Overview routine of the application

At the beginning MySQL database is used as a source of the information and a data needed to further process. Whenever there are ready entries to work with them in the database, the script fetches data from the FTP server or from a locally mounted storage and then processes them. In most cases the script has to work with the images; usually at least two of them, from which the first one is used as a background image and the second one as a tagging image, which is placed on the background image according to the coordinates stored in the XML file. After that the script creates a PHP file containing proper data and later it is used by a web server. When the whole procedure is finished, all required files are uploaded to the location specified in a configuration file. Afterwards iSync sends an email with a notification to the email address fetched from a database. At that point if there are no problems one taken job is finished and the proper field in a database is changed.

Within all these activities the script writes previously defined information and errors' descriptions to a log file. In case of any failure or obstacle this information helps the developer or administrator of the server to solve a problem much quicker instead of starting investigating what has gone wrong and when it has happened.

2.2 .NET and Mono

.NET initiative was announced by Microsoft in July 2000. .NET platform was a new development framework with a new programming interface to the Windows services and APIs, integrating a number of technologies that emerged from Microsoft during late 90s. .NET, however, was not distributed alone but there were following elements incorporated with it:

- COM+ component service
- ASP web development framework
- commitment to XML
- object – oriented design
- support for new web services protocol as SOAP, WSDL and UDDI.

The most important component of the .NET framework is CLR. At a high level CLR activates objects, performs security, checks on the objects, lays them out in a memory, executes them and garbage-collects them. Conceptually, the CLR is a runtime infrastructure that abstracts underlying platform differences. CLR supports all languages that can be represented in the Common Intermediate Language. (Thai & Lam, 2002; Albabari & Albabari, 2010)

.NET, however, is also a collection of tools, utilities and components that aid developers in building a console, GUI, and web applications, abstracting much of the lower level detail involved in building high-performance software. In the sense of environment the most important components of .NET are:

- The Microsoft .NET Framework, a collection of core classes and libraries to perform common tasks
- The C# programming language, an object-oriented compiled language
- The Common Language Runtime (CLR), an MSIL interpreter and just-in-time runtime that converts MSIL to native machine code
- ADO.NET, a data access library.

(Liberty & Xie, 2008)

FIGURE 3 shows the evolution of the .NET Framework from version 2.0 to version 4.0.

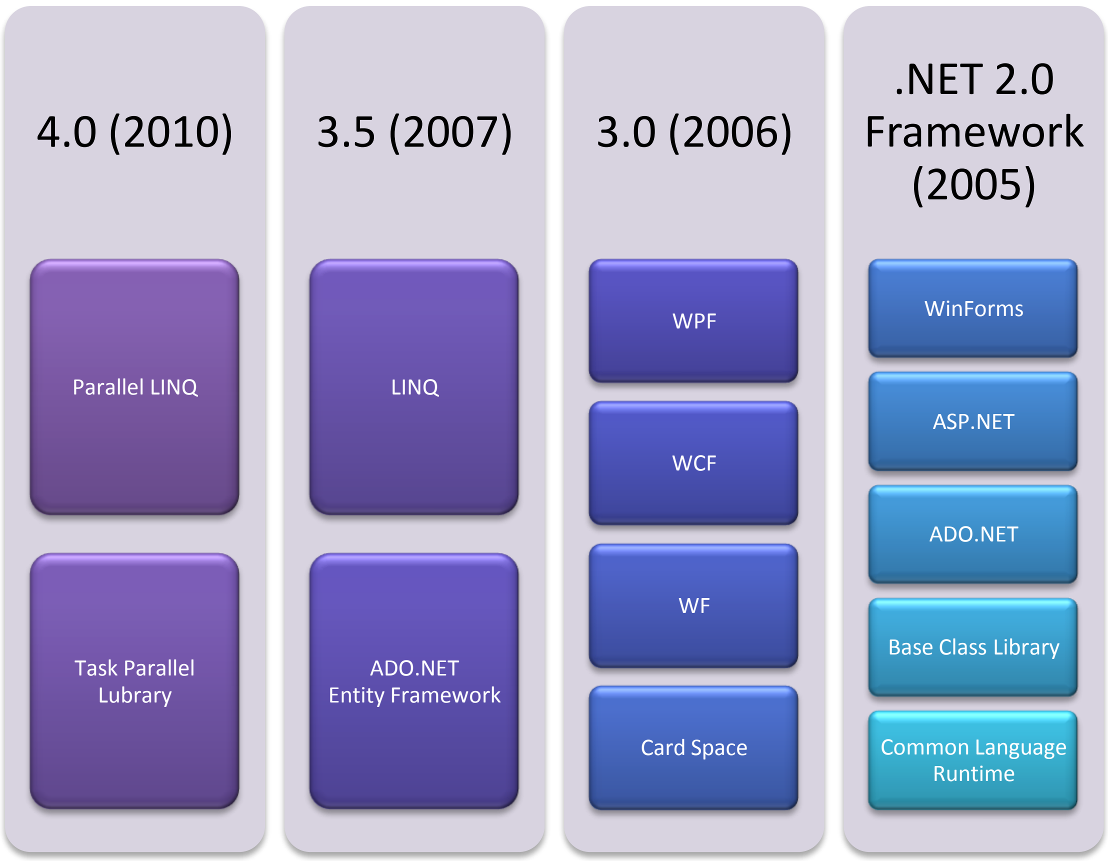


FIGURE 3. .NET Framework evolution (Nash, Accelerated C# 2010, 2010)

The application refactoring to a .NET environment, in case of this thesis, is strongly related to a Linux environment. C# as a language does not have any support on the other operating systems than Windows. This is a very crucial issue, although there is a way to use C# compiled code on the Linux OS. This is provided by Mono, which is an open-source initiative that brings .NET technologies to operating systems other than Microsoft Windows. (Mamone, 2006)

Mono provides the necessary software to develop and run .NET client and server applications on Linux, Solaris, Mac OS X, Windows, and UNIX. The project’s objective is to completely port the Microsoft .NET development platform to UNIX, thereby allowing UNIX developers to build and deploy truly cross-platform .NET applications regardless of the operating system and machine architecture. The project, although sponsored by Novell, is open-source. The current version of Mono available during application refactoring - Mono 2.4 - supports most of the common functionality offered within the .NET environment. Mono contains the core development libraries, as well as the development and deployment tools. At the time of writing, Mono API coverage is limited to the. NET 3.0 API, with spotty support for the version 3.5.

Mono contains the following components:

- A Common Language Infrastructure (CLI) virtual machine that contains a class loader, a just-in-time compiler, and a garbage collector
- A class library that can work with any language which works on the CLR. Both .NET compatible class libraries as well as Mono provided class libraries are included
- A compiler for the C# language. Future work on other compilers that target the Common Language Runtime is planned.

(Swaminathan, 2007; Schonig & Geschwinde, 2004)

2.3 MySQL and its .NET binding

MySQL is a relational database management system which acts as a server providing multi user access to a number of databases. It is available as either a binary or a source-code download. MySQL is covered under GPL and LGPL licenses.

MySQL is available on many different operating systems on variety of computer architectures. Currently it has versions for Linux, Windows, Solaris, FreeBSD, MacOS X, HP – UX, AIX, SCO, SCI, Irix, Dec OSF and BSDi. The Linux version of MtSQL runs on a range of architectures. The availability of a cross – platform versions has enhanced the popularity of MySQL. (Suehring, 2002)

Anyhow, .NET environment does not natively support MySQL database. Hence there was a need to use external components to provide the missing functionality. Such a library is available and provided by Sun Microsystem on MySQL webpage – it is Connector/NET.

Connector/NET enables developers to easily create .NET applications that require secure, high-performance data connectivity with MySQL. It implements the required ADO.NET interfaces and integrates them into ADO.NET aware tools. Developers can build applications using their choice of .NET languages. Connector/NET is a fully managed ADO.NET driver written in 100% pure C#.

Connector/NET includes full support for:

- Features provided by MySQL Server up to and including MySQL Server version 5.5
- Large-packet support for sending and receiving rows and BLOBs up to 2 gigabytes in size
- Protocol compression which enables compressing the data stream between the client and server
- Support for connecting using TCP/IP sockets, named pipes, or shared memory on Windows
- Support for connecting using TCP/IP sockets or Unix sockets on Unix
- Support for the Open Source Mono framework developed by Novell
- It is fully managed, and does not utilize the MySQL client library.

(MySQL :: MySQL 5.1 Reference Manula :: 20.2 MySQL Connector/NET, 2010)

2.4 Logger

Logging is a commonly used technique in software development, which allows to check if expected behavior happened in proper moments while the program was running. The most important benefit of this technique, however, is to track down a problem using logged data. When some problem occurs during runtime it is written to a log file with all associated descriptions.

The original iSync has written only the information about the progress. In practice it means that whenever some error occurred it showed only the last valid entry in a log file. There were no data saying what problem had occurred. This approach was very problematic and caused extra work for the attendant and that is why it had to be changed.

In the new implementation, the application writes three types of data to a log file:

- information
- warnings
- errors.

Information tells about normal operations done during a runtime. Warnings point that part of necessary verifications in a code which fail whereas the errors indicate unexpected behavior or exceptions.

With the help of superior of the project it was decided that logging class would be reconstructed as a similar standalone class and it would implement the singleton pattern.

The intent of a singleton pattern is to ensure that a class has only one instance and to provide a global point to it. The model for a singleton is very straightforward. Usually there is only one singleton instance and clients access a singleton through one well-known access point. The client in this case is an object that needs access to an instance of a singleton. The previously described relationship is showed in the FIGURE 4 below.

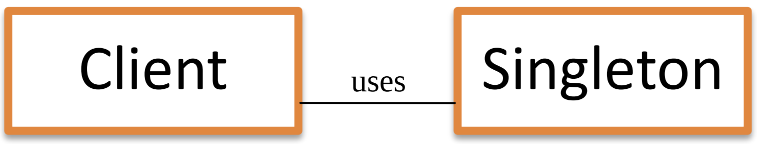


FIGURE 4. Logical model of a singleton pattern

(Exploring the Singleton Design Pattern, 2010)

Singleton
-static uniqueinstance : Singleton -sindletonData

+ static Instance() : Singleton
+ SingletonOperation()
+ GetSingletonData()

FIGURE 5. Singleton pattern physical model from design patterns (Gamma, Helm, Johnson, & Vlissides, 1994)

The physical model for the Singleton pattern is also very simple. The FIGURE 5 above shows a simple class diagram. There is a private static property of a singleton object as well as a public method Instance() that returns this private property.

2.5 FTP client

The File Transfer Protocol is one of the oldest and most often used protocols in the internet. FTP specifications state that by default, all data transfers should be done over a single connection. An active open is done by the server, from its port 20 to the same port on the client machine as was used for the control connection. The client does a passive open. For better or worse, most current FTP clients do not behave this way. A new connection is used for each transfer, to avoid exceeding TCP’s TIMEWAIT state, each time a client picks a new port number and sends a PORT command announcing that to the server. Nowadays FTP client uses two parallel connections. One connected to the server’s port 21 which is called control connection and it stays open for the duration of the session, and the second one is a data connection opened by the server from its port 20. (Bellovin & Laboratories, 1994)

The objectives of the FTP are:

- To promote files’ sharing (computer programs and/or data)
- To encourage indirect or implicit (via programs) usage of the remote computers
- To shield a user from variations in the file storage systems among the hosts
- To transfer data reliably and efficiently.

(Postel & Reynolds, 1985)

2.6 XML handling

Extensive Markup Language, abbreviated XML, describes a class of the data objects called XML documents and partially describes a behavior of the computer programs which process them. XML is an application profile or a restricted form of SGML, the Standard Deserialized Markup Language. By construction, XML documents are conforming to SGML documents. XML documents are made up of storage units called entries, which contain either parsed or unparsed data. XML provides a mechanism to impose constraints on the storage layout and a logical structure.

The design goals for XML are:

- XML shall be straightforwardly usable over the Internet
- XML shall support a wide variety of applications
- XML shall be compatible with SGML
- It shall be easy to write programs which process XML documents
- The number of optional features in XML is to be kept to the absolute minimum, ideally zero
- XML documents should be human-legible and reasonably clear
- The XML design should be prepared quickly

- The Design of XML shall be formal and concise
- XML documents shall be easy to create
- Terseness in XML markup is of minimal importance

(Extensive Markup Language (XML) 1.0 (Fifth Edition), 2008)

2.7 Control version system

System version control is used in the most of the software development projects to maintain a source code and to easily follow all the changes that have been done during the development process. Refactorization of the iSync PHP script was also kept in the SVN to easily track all improvements. TortoiseSVN application, which is built against Subversion was used for that purpose. This source control software enables easy access to the project from Windows operating system. It integrates itself with Windows shell, which allows to use it conveniently and intuitively. (tortoisesvn.tigris.org, 2010)

The repository was created locally – there was no need to create a remote one. The structure of its folders was as follows:

- testApps
- iSunc
    - trunk
    - branches
    - tags

In the testApps location a small application was created, which used the developed classes and tested them. After the class was well tested, it was enclosed to the final project, which was placed in the iSync/trunk. The experimentally developed alternatives were kept in the branched location, while fully working copies of trunk branch containing certain functionality were stored in the tags.

3 IMPLEMENTATION OF THE RESEARCH

3.1 Overview

The main functionality of the original application is in operating with the images and copying them to the FTP server when their processing is finished. The author tries to explain all necessary information needed for understanding what happens in the heart of the PHP script.

Starting from the beginning, the script is run every second minute. It requests data from MySQL database and if it receives any data, it proceeds with them. The first data are copied to the local temporary folder from a mounted network space. Afterwards a necessary checking part is executed and if everything is valid a script starts working with the images. When the images' processing is ready, the results have to be copied to the FTP server and the email notification is sent to a proper person.

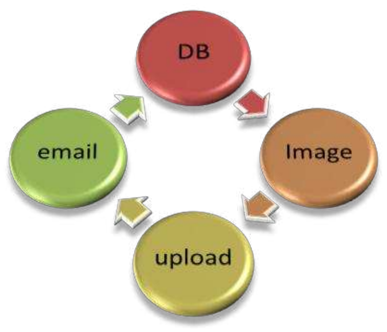


FIGURE 6. Data flow in the script

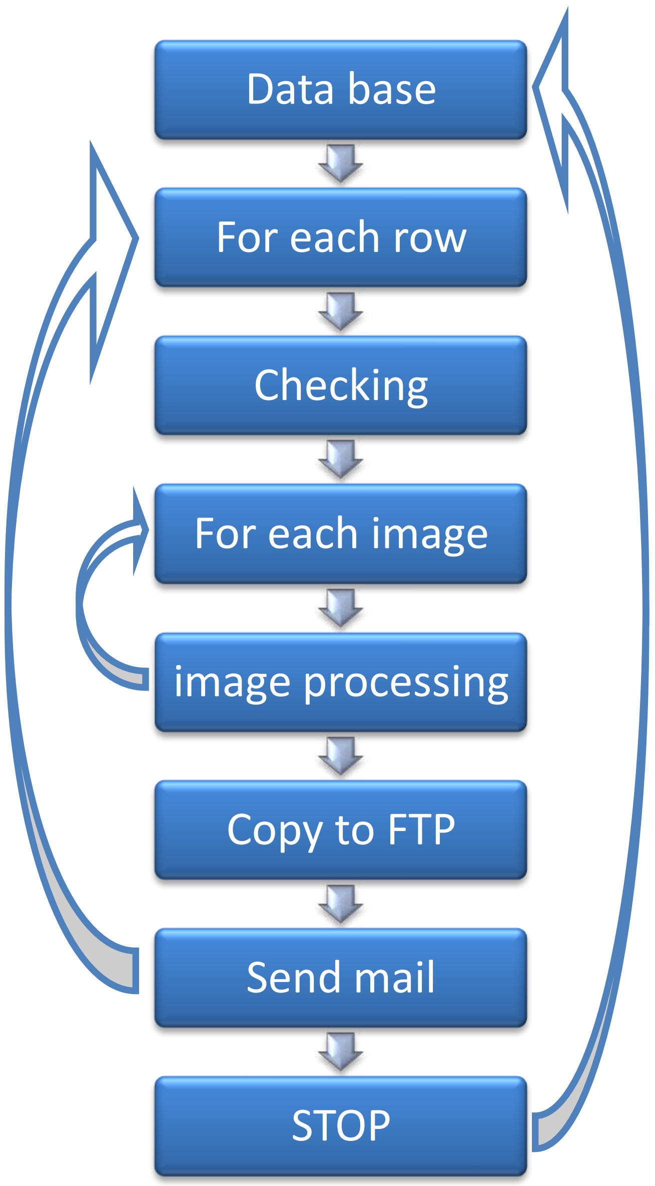


FIGURE 7. Data flow in main routine

Before anything was coded it had to be checked if everything involved in script could be easily moved to .NET environment and could be run fully compatible with the Mono. The project was started by reading a documentation on the Mono webpage and checking actual supported frameworks. It turned out that Mono have already implemented .NET 3.0 and also with some part from version 3.5. It was enough to handle all functionality in a new code. (Mono, 2004)

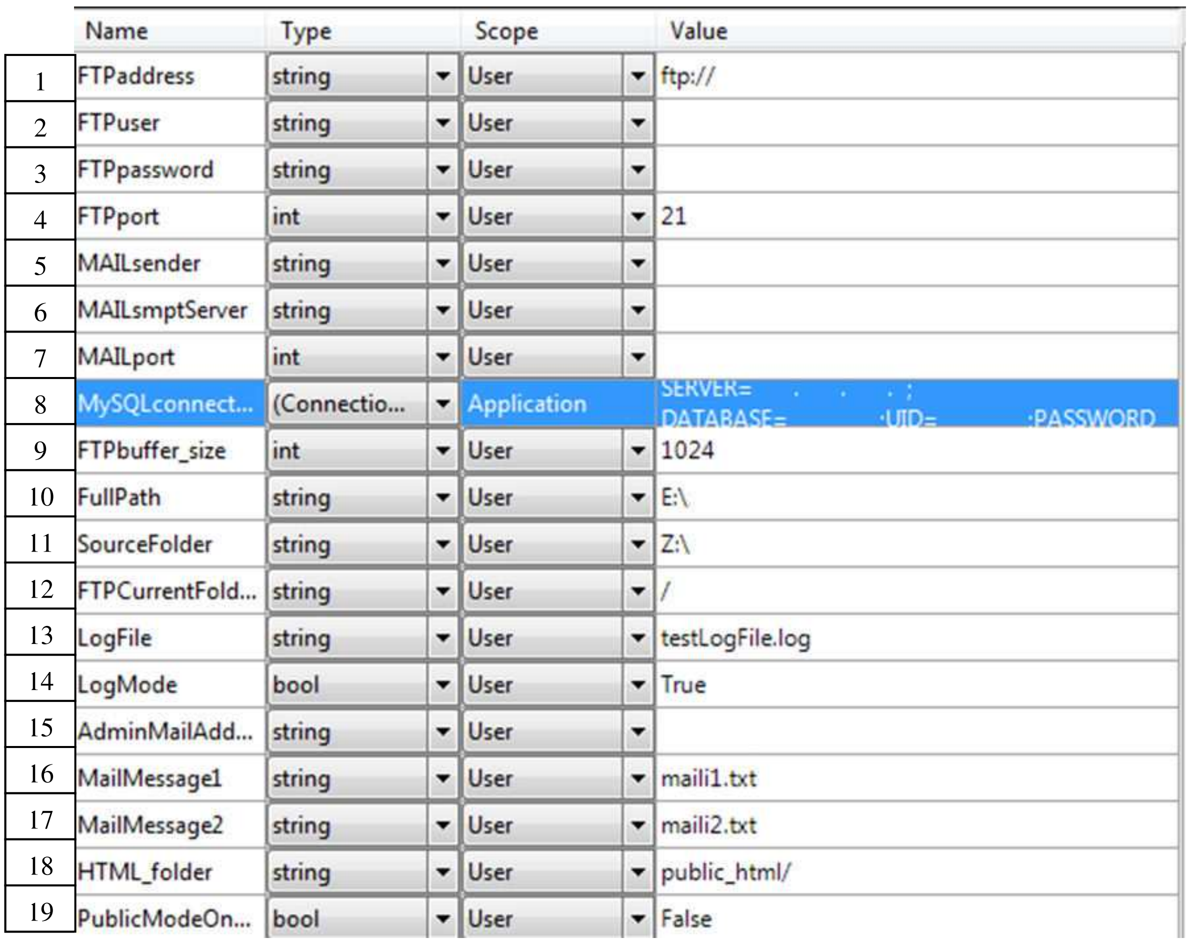
Afterwards the most important matter was to check the source code of a script. Neither clear convention was used in a PHP code nor division between different parts of a code. The comments had to be made everywhere and the order in a code was familiarized with by the author. One of the files containing logging functionality had a suggestion that this should be done as a singleton pattern. At that point it was understandable what the reason to do so is. When creating a log file it should be opened and handled only by one instance of a class. That is not only safe from the operation on files point of view but also it provides correct timestamps in a log file. It was decided that this pattern would be applied to a new application.

When it was known how the script worked it had to be decided how to start converting it. The approach that was taken based on creating classes providing proper functionality related only to the one picked part of the program, for example: FTP database, email client, XML handling, image processing. That approach also allowed easier application testing part by part or segment by segment.

3.2 Implementation

As presented in the previous chapters iSync consists of pieces from different technologies. To make the application flexible and easier to be modified, a configuration file was created to keep all data that might be changed in the future outside of the source code. A bad approach would be to use hard coded settings of FTP client or mail sender. That would prevent developers from changes of those properties without compilation, whereas PHP script allowed to do that freely and verify them almost immediately. The configuration file has given the same quality to binary, compiled application. The FIGURE 8 below illustrates the configuration fields. The screenshot was made from Visual Studio 2008. The first four options and options 9 and 10 are related to the FTP client object configuration while fields from 5 to 7 are for email sender object settings. Others, e.g. selected on blue *MySQLconnectionString* is for *DBHandler* object, which contains all required data like address of the server, database name and credentials. Fields 10, 11and 18 point to the related folders' destinations. The log mode field and the log file name can be seen respectively in the fields 13 and 14. The 15th row contains the administrator's mail address used in case of some unexpected actions. The next fields 16 and 17 contain the mail's body schemas and the last 19th. field specifies whether the results should be copied into the path from row 18.

The presented configuration enables customization of the components without recompiling sources. This brings to the new application the same quality as PHP script has. Moreover, it clearly splits and shows these customized variables from the source code and program itself.

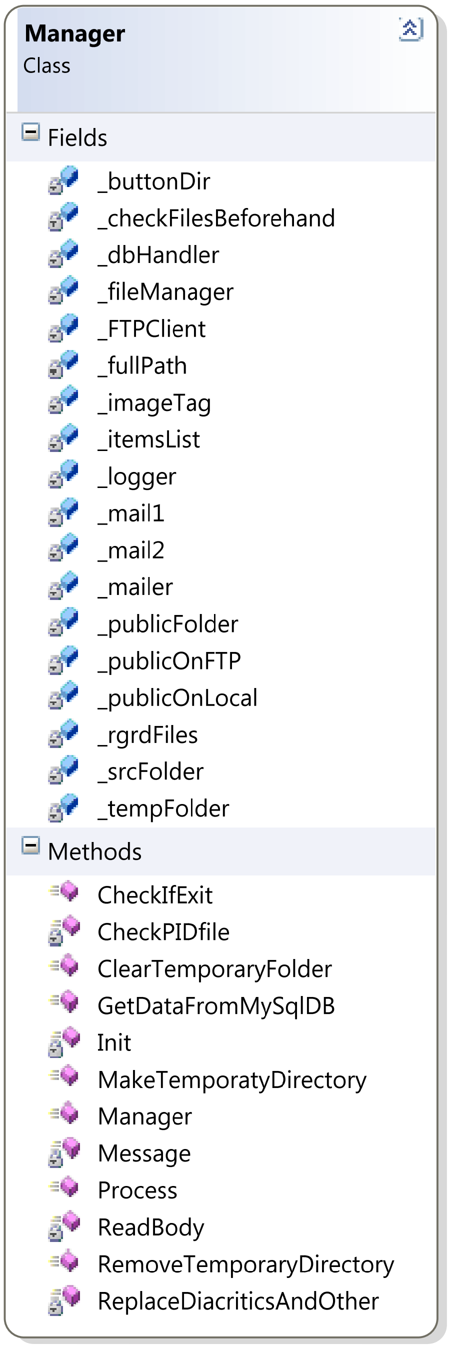


	Name	Type	Scope	Value
1	FTPaddress	string	User	ftp://
2	FTPuser	string	User	
3	FTPpassword	string	User	
4	FTPport	int	User	21
5	MAILsender	string	User	
6	MAILsmptServer	string	User	
7	MAILport	int	User	
8	MySQLconnect...	(Connectio...	Application	SERVER= . . . ; DATABASE= ;UID= ;PASSWORD
9	FTPbuffer_size	int	User	1024
10	FullPath	string	User	E:\
11	SourceFolder	string	User	Z:\
12	FTPCurrentFold...	string	User	/
13	LogFile	string	User	testLogFile.log
14	LogMode	bool	User	True
15	AdminMailAdd...	string	User	
16	MailMessage1	string	User	maili1.txt
17	MailMessage2	string	User	maili2.txt
18	HTML_folder	string	User	public_html/
19	PublicModeOn...	bool	User	False

FIGURE 8. View of XML configuration file fields

3.3 .NET

The FIGURE 9 below shows all fields and methods of the main class called *Manager*. The object of this class is responsible for and triggers all activities in iSync. Only one instance of this class is created in the final application. While reviewing fields of this class, it can be easily noticed that some of them match fields from the configuration showed in the previous chapter. Indeed these fields are filled with configuration data provided in configuration file. The core method of the application is called *Process.* It contains refactored routine from PHP script and within this function all important operations occur.



Manager
Class

Fields

- _buttonDir
- _checkFilesBeforehand
- _dbHandler
- _fileManager
- _FTPClient
- _fullPath
- _imageTag
- _itemsList
- _logger
- _mail1
- _mail2
- _mailer
- _publicFolder
- _publicOnFTP
- _publicOnLocal
- _rgrdFiles
- _srcFolder
- _tempFolder

Methods

- CheckIfExit
- CheckPIDfile
- ClearTemporaryFolder
- GetDataFromMySqlDB
- Init
- MakeTemporatyDirectory
- Manager
- Message
- Process
- ReadBody
- RemoveTemporaryDirectory
- ReplaceDiacriticsAndOther

FIGURE 9. Manager class

The *Manager* class contains also a function called *CheckPIDfile*. The application has to mark somehow which item in a database is currently processed to avoid processing the same item twice. iSync creates a PID file, which is in that case nothing more than an empty file with a proper item's name. The application uses *CheckPIDfile* function to check whether an item, which it is interested in, is actually in a progress. If not it proceeds with it, otherwise it simply skips this item.

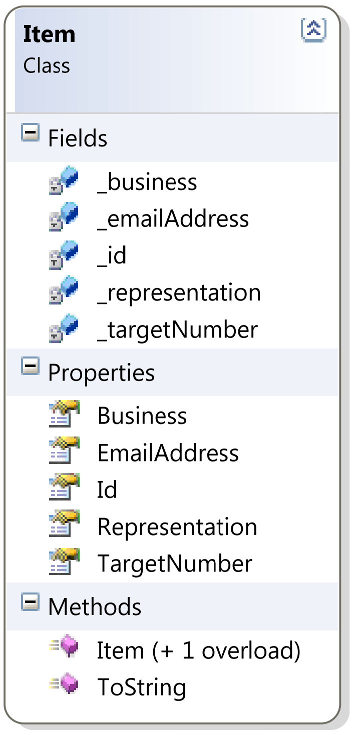


FIGURE 10. Item class which represents fetched data from database.

One of the most important classes in the application is shown in FIGURE 10. It encapsulates all needed data and provides a convenient way to access all class's fields. It also allows printing out information about an item by overloading a *ToString* function. In the original PHP class the data received from a database were kept in the separate variables without obvious connections. The object representation with proper fields allows keeping related data in the one place and provides an easy access to them. It is also very easy to store more items in any container.

3.4 MySQL

The FIGURE 11 below shows class responsible for connection, fetching data, modification and updating fields in a database. Its function called *FetchData* acts as a data input for an application. It fetches entries from a database and next parses them to an object of *Item* class showed in the previous chapter. All valid entries from a database are stored in the following container *List*<*Item*> *_itemsList*.

Nevertheless *FetchData* is not the only function which uses connection with a database. The others are *CheckIfWholePathExist*, *SetPath* and *SetStatusDownloaded*. The first function is responsible for checking if there is at least one cell containing certain string value in a database, the second function takes care of setting a path to an item with a specific id and the last itemized procedure sets fields of an item with a specific id as downloaded and done.

The other functions in *DBHandler* class are also related to a database and they check whether a connection to a database is active or if it was already closed.

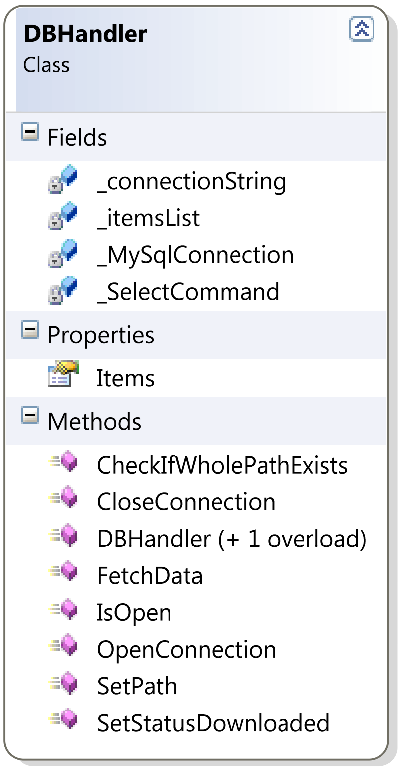


FIGURE 11. Data base handler class

3.5 File management

The application operates with different types of data. Some of them are received from a database and represent processes' ids which iSync handles, the other are XML file containing points' coordinates or picture files. Software, however, not only processes the content of all received files but it also manages the files themselves, for example it moves or copies them between different locations. For that purpose a file manager is used. At the beginning and the end of the application's routine, the files are moved here and back. Firstly, they are copied from a mounted network location to the temporary folder and when the most of the processes are done, the results have to be placed in a specified location as well.

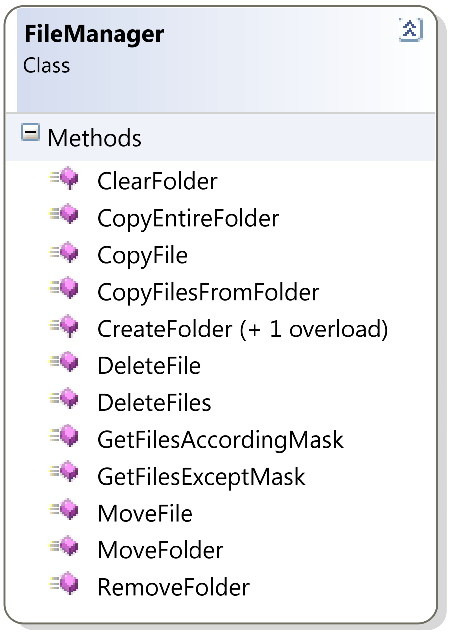


FIGURE 12. File manager class

The *FileManager* class contains also the method responsible for creating and removing folders. This is required for iSync since it processes data copied locally into a temporary folder. After all processes are finished, it copies data to a destination place. In case a destination path does not exist, it creates one and then processes with data copying. iSync also deletes the content of a temporary folder which is created whenever it is needed.

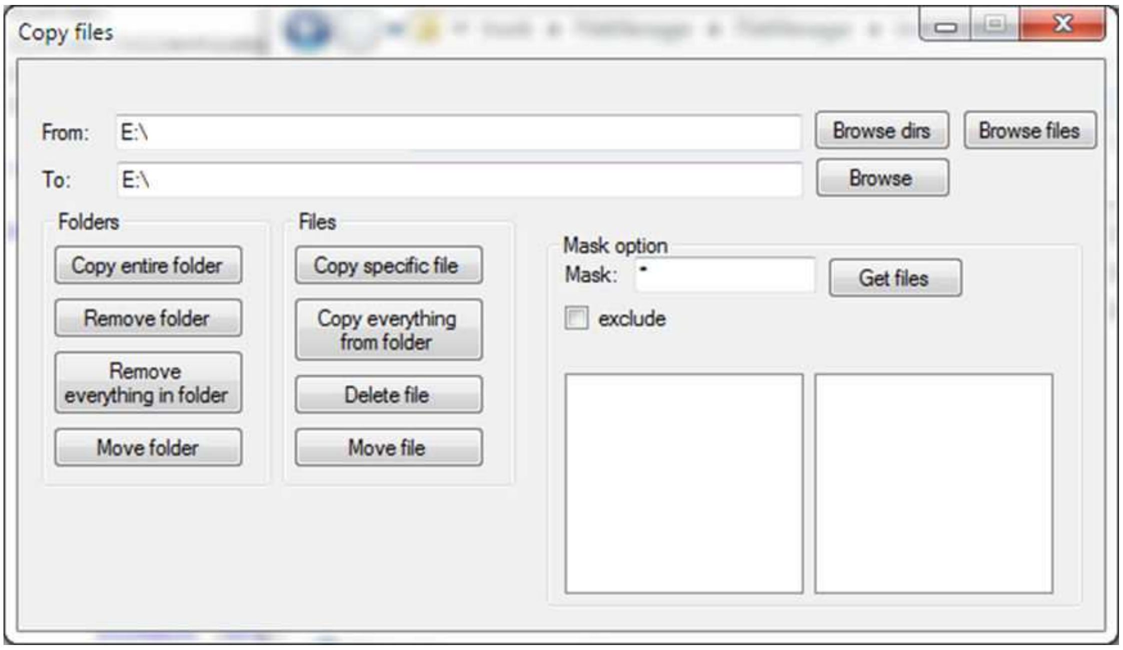
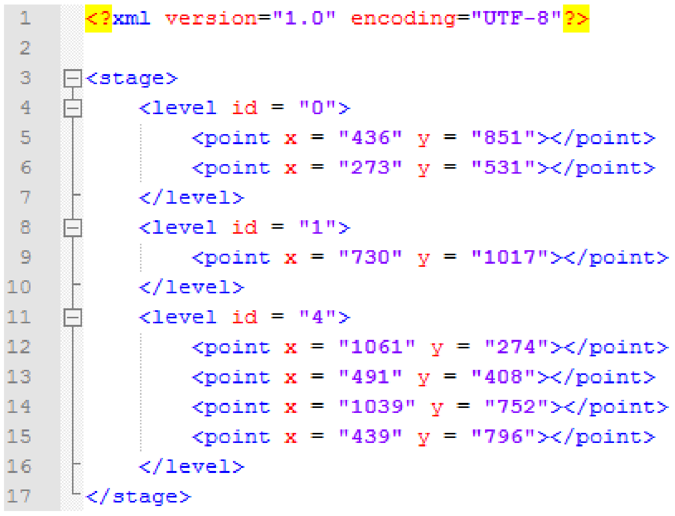


FIGURE 13. UI of test application using *FileManager* class

The figure above shows the graphical user interface of a test application. The application wraps all functionality provided by *FileManager* class and provides an easy way of testing it.

3.6 XML handling

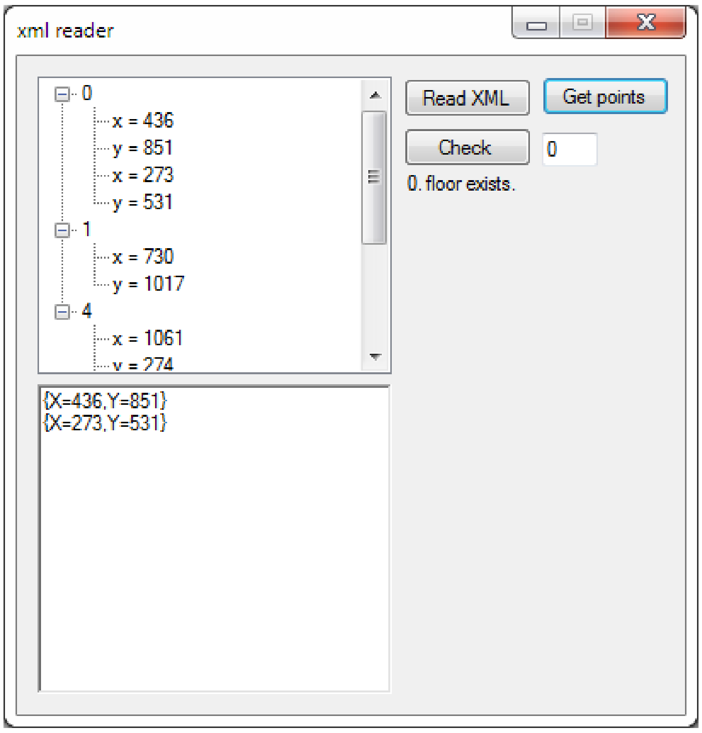
FIGURE 14 below shows an exemplary XML file which is processed by the application. The file always includes only one stage element which may contain from zero to few level elements, whereas each level element contains at least one point entry. This structure is very clearly visible in FIGURE 15.



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

<stage>
    <level id = "0">
        <point x = "436" y = "851"></point>
        <point x = "273" y = "531"></point>
    </level>
    <level id = "1">
        <point x = "730" y = "1017"></point>
    </level>
    <level id = "4">
        <point x = "1061" y = "274"></point>
        <point x = "491" y = "408"></point>
        <point x = "1039" y = "752"></point>
        <point x = "439" y = "796"></point>
    </level>
</stage>
```

FIGURE 14. XML file structure



```
0
    x = 436
    y = 851
    x = 273
    y = 531
1
    x = 730
    y = 1017
4
    x = 1061
    y = 274
```

```
{X=436,Y=851}
{X=273,Y=531}
```

0

0. floor exists.

FIGURE 15. UI of the test application which presents and process an XML file. It wraps *XMLCoordinatesReader* class

FIGURE 16 below shows a structure of *XMLCoordinatesReader* class. The object of this class is responsible for parsing the XML file showed in FIGURE 14. From the mentioned FIGURE 14 can also be seen that the levels' ids are not consecutive numbers – there are levels with following ids: 0, 1, 4. That is why *XMLCoordinateReader* class has a function *CheckIsFloorExists*, which checks if a parsed XML file contains the floor id, which the application requests. The function *GetPointsOnSpecificFloor* takes as an argument floor number and returns an array of points corresponded to this number.

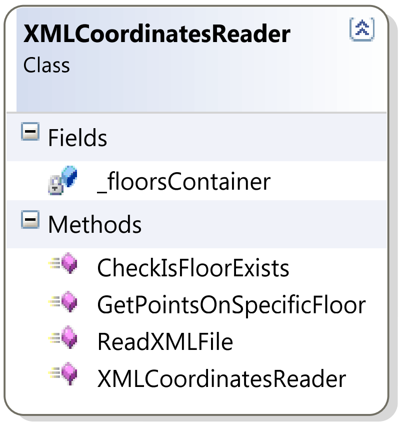


FIGURE 16. XML handler class.

When an application calls a *ReadXMLFile* function with a path to an XML file as a parameter, an object reads this file and stores data in a dictionary `Dictionary<int, List<Point>>`. It provides a convenient way of getting points from a specific level.

3.7 Image processing

The image processing is a very important part of the original PHP script. The processed images with an associated PHP file, which is also created by an iSync, bring required functionality to the web server which uses them. The operations that are performed on each image are as following:

1. Reading points' coordinates from XML file – presented in the chapter 3.6
2. Loading a background image and resizing it in the way that larger dimension from height and width is 360 pixels and the second dimension is scaled respectively
3. Calculating ratios for height and width of a background image
4. Applying the scales to the read coordinates from the 1. point

5. Creating a PHP file called *id_level.php* which contains predefined data
6. Placing the center of a tagging image on each read point on a loaded background image
7. Saving a tagged image (background image combined with tagging images) by overriding the old one.

FIGURE 17 shows a diagram of the *ImageHandler* class which provides all functionality related to a data flow presented in the points above. The actions listed previously can be easily linked to the function enclosed in the *ImgeHandler* class.

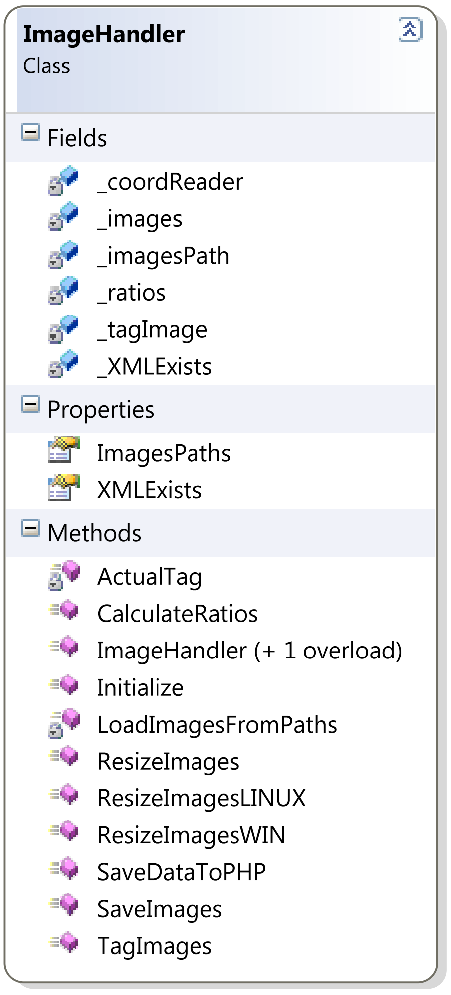


FIGURE 17. Image handler class

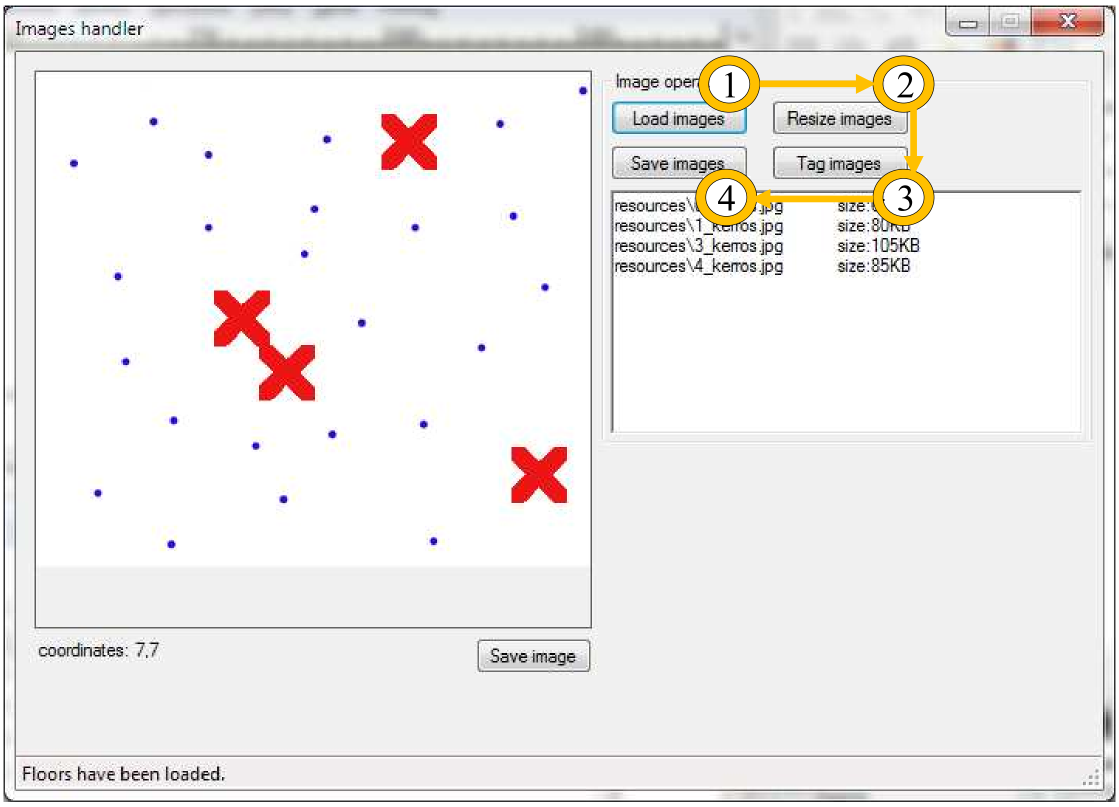
All fields of the above presented class are private ones. One of them – *ImagePaths* provides *get* and *set* property, the other – *XMLExists* provides only a *get* property.

This is done in this way due to the fact that object of *ImageHandler* class may receive and use information about the paths to the images, however, the existence of an XML file is being determined by this object itself.

Most of the methods of the *ImageHandler* class are public, only the loading and tagging images can be called internally by an object of this class.

FIGURE 18 shows the graphical user interface created for the testing purposes of the *ImageHandler* class functionality. The functionality of this testing application can be split into two parts.

The first one – mainly left part of the UI – tests tagging a predefined image, which is presented as a white rectangle containing blue dots and red crosses. Whenever the left mouse button is clicked within a rectangle the tagging operation is called and the coordinates of a clicked point are showed below the border line. In addition, it allows saving the showed figure to a file representation.



```
resources\        .jpg      size:
resources\1_kerros.jpg      size:80
resources\3_kerros.jpg      size:105KB
resources\4_kerros.jpg      size:85KB
```

coordinates: 7,7

Floors have been loaded.

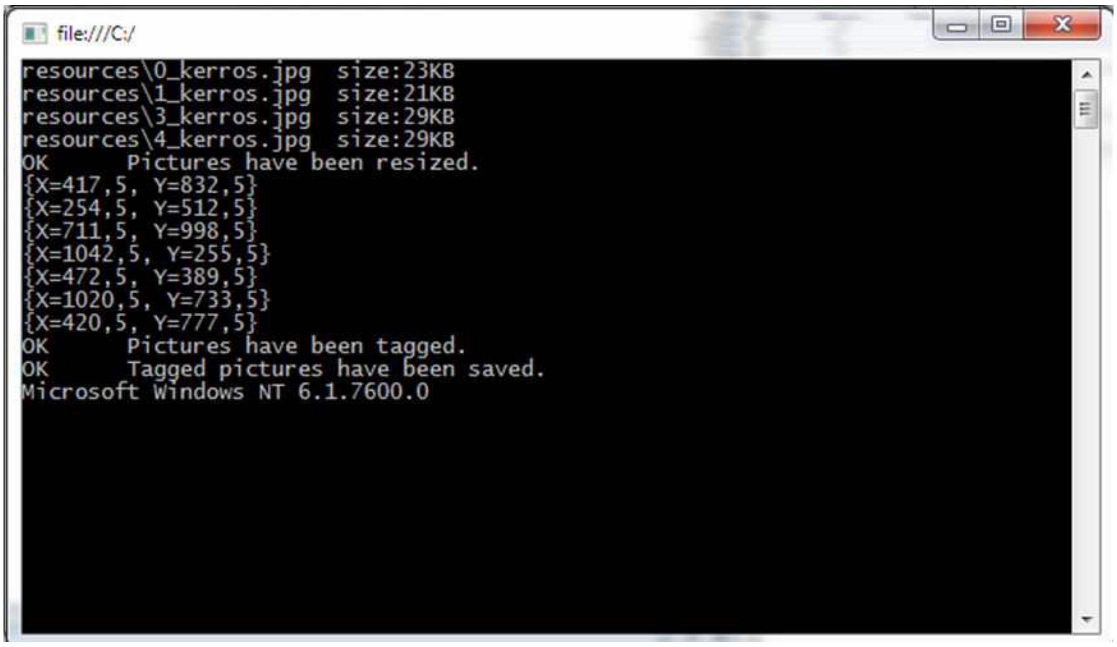
FIGURE 18. UI of the test application using the *ImageHandler* class

The second part of this testing application is a semi-automated process split into four following steps:

1. Loading the images from a hardcoded path
2. Resizing the previously loaded images to a proper size
3. Tagging the images
4. Saving the images by overriding the old ones

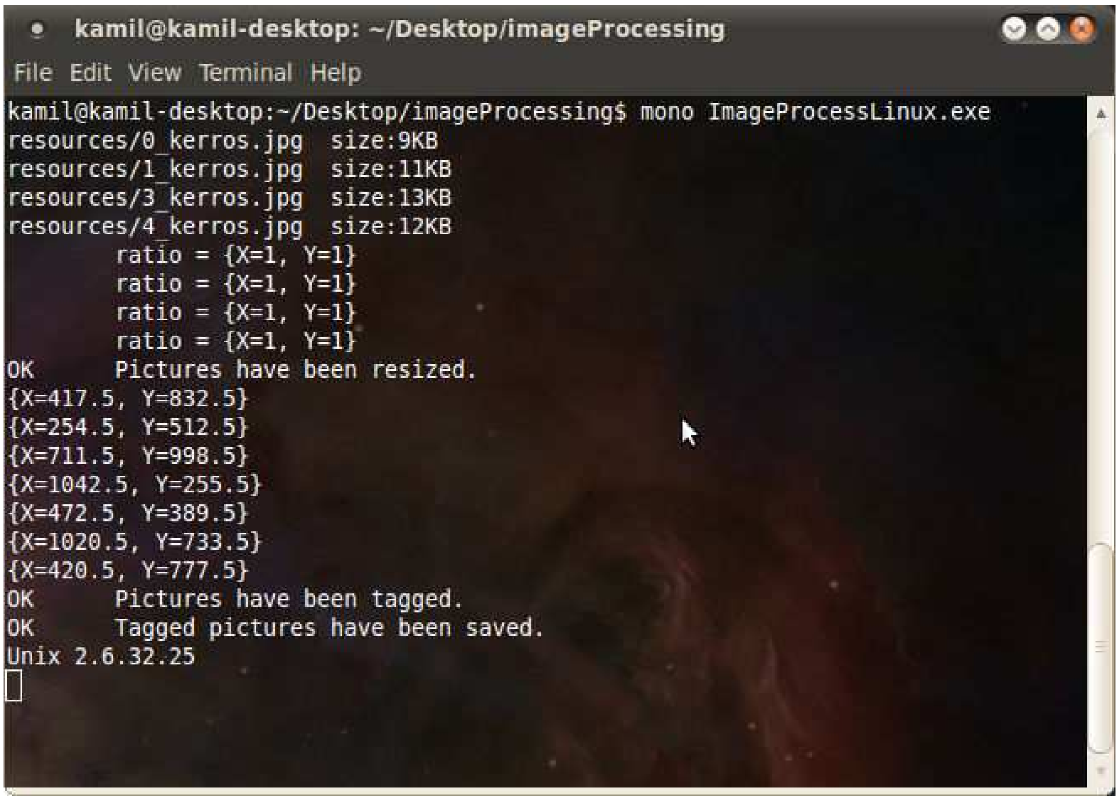
The listing below buttons in FIGURE 18 contains the paths to the loaded images and their sizes.

FIGURE 19 and FIGURE 20 show another testing application of the *ImageHandler* class. This time it does not contain any graphical interface, it is a pure console application. It was created in order to check the implemented different functionality of images resizing in the different operating systems. In the first FIGURE 19 the output comes from the Microsoft Windows operating system, the second printout comes from the Linux operating system. The testing application was compiled on the Windows that is why it can be run directly on this system, nevertheless to run it on the Linux OS, it requires to use Mono. (Mono, 2004)



```
file:///C:/
resources\0_kerros.jpg  size:23KB
resources\1_kerros.jpg  size:21KB
resources\3_kerros.jpg  size:29KB
resources\4_kerros.jpg  size:29KB
OK      Pictures have been resized.
{X=417,5, Y=832,5}
{X=254,5, Y=512,5}
{X=711,5, Y=998,5}
{X=1042,5, Y=255,5}
{X=472,5, Y=389,5}
{X=1020,5, Y=733,5}
{X=420,5, Y=777,5}
OK      Pictures have been tagged.
OK      Tagged pictures have been saved.
Microsoft Windows NT 6.1.7600.0
```

FIGURE 19. Console testing application run under Windows OS


kamil@kamil-desktop: ~/Desktop/imageProcessing
File Edit View Terminal Help

```
kamil@kamil-desktop:~/Desktop/imageProcessing$ mono ImageProcessLinux.exe
resources/0_kerros.jpg  size:9KB
resources/1_kerros.jpg  size:11KB
resources/3_kerros.jpg  size:13KB
resources/4_kerros.jpg  size:12KB
        ratio = {X=1, Y=1}
        ratio = {X=1, Y=1}
        ratio = {X=1, Y=1}
        ratio = {X=1, Y=1}
OK      Pictures have been resized.
{X=417.5, Y=832.5}
{X=254.5, Y=512.5}
{X=711.5, Y=998.5}
{X=1042.5, Y=255.5}
{X=472.5, Y=389.5}
{X=1020.5, Y=733.5}
{X=420.5, Y=777.5}
OK      Pictures have been tagged.
OK      Tagged pictures have been saved.
Unix 2.6.32.25
```

FIGURE 20. Console testing application run under Linux OS

There is also one more very visible difference between running the application on Windows and Linux OS. Due to the problem with interpolation while image resizing, which occurred on the Linux OS, iSync has to use external application which in this case is ImageMagic. (ImageMagick: Convert, Edit and Compose Images, 2009)

3.8 Mail client

As presented in chapter 2.4 iSync is intended for logging its activities to the log file. This is enough, however only as long as everything goes without any unexpected issues. When any exception occurs, the administrator of the application should be notified about the issue, the legacy PHP script used to send a mail notification, which was further connected with a text message notification. When an email server receives a message from a particular sender – in that case an email sent by the script – it forwards an SMS to the administrator via messaging service. This is a very convenient and fast way to inform the responsible person to look into a script or / and a database in case of any serious problem.

It was decided that a new C# iSync will also use an email notification. For that purpose the *Mailer* class was created. It provides simple functionality and acts as an email client only with sending messages option.

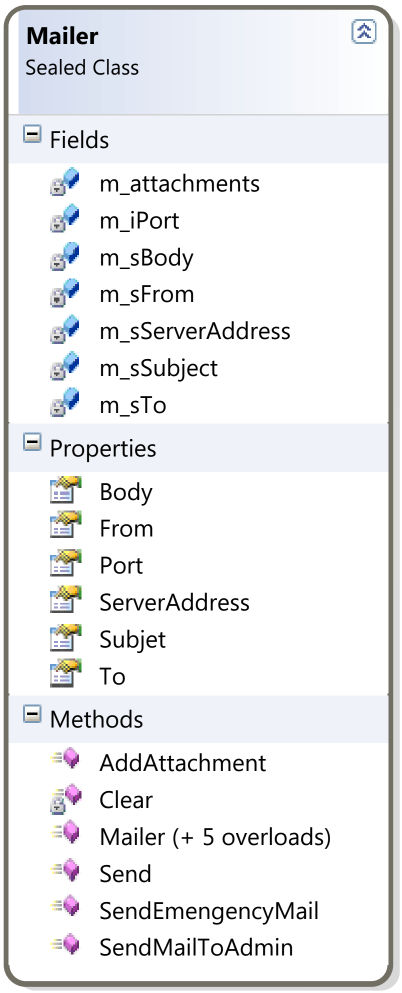
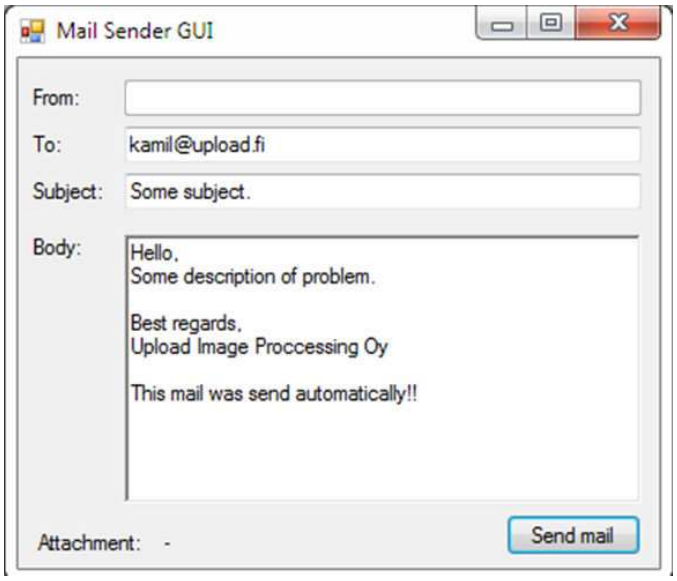


FIGURE 21. Mailer class

The object of *Mailer*, however, is used not only to notify an administrator that some problem occurred, it is also used to send an email notification to the clients whose email address was delivered from a database. The information about an email address is kept in the *Item* object in one of its fields called *_emailAddress*. These kinds of emails are sent when an item has been already copied to the FTP server and it is now available for the client. This routine is showed in FIGURE 7.

FIGURE 22 shows a simple testing application which was used to check whether *Mailer* class works as expected.



To: kamil@upload.fi
Subject: Some subject.
Body:
Hello,
Some description of problem.

Best regards,
Upload Image Proccessing Oy

This mail was send automatically!!

Attachment: -

FIGURE 22. Simple UI of testing application which sends an email

3.9 Logger

The *Logger* class was created as a sealed class implementing Singleton design pattern. This means that no other classes can derive from it and it can have only one instance accessible from every part of the code. This is very a convenient way of designing *Logger* considering its purpose. *Logger* has few characteristic attributes:

- single private constructors which in consequence do not allow other classes to create an instance of *Logger*
- sealed syntax word used in order to help JIT optimizes things more
- it contains private *Nested* class which has access to the whole *Logger* class
- *Nested* class contains static variable which holds a reference to a single created instance of *Logger*

- Instantiation is triggered by the first reference to the static member of the nested class, which only occurs when the *Instance* property is called and this means that implementation is fully lazy.

(Implementing the Singleton Pattern in C#, 2006)

FIGURE 23 shows the diagram of the *Logger* class. It presents that the main *Logger* class has two constructors, although both are used only internally and are private.

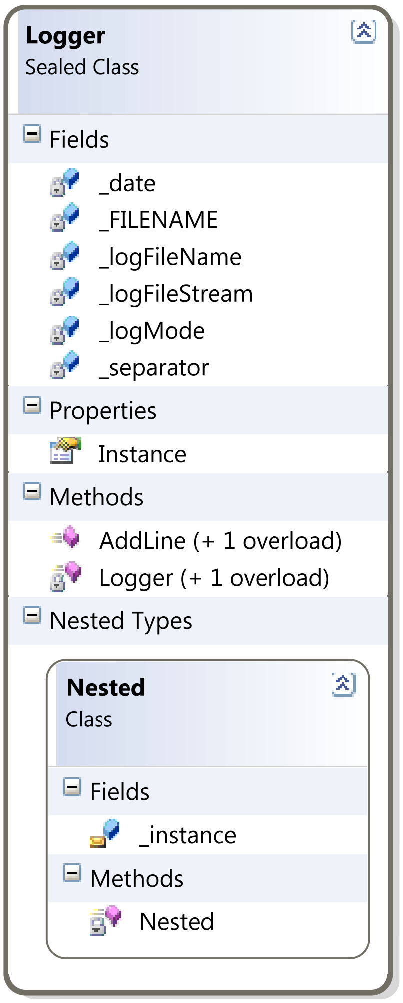
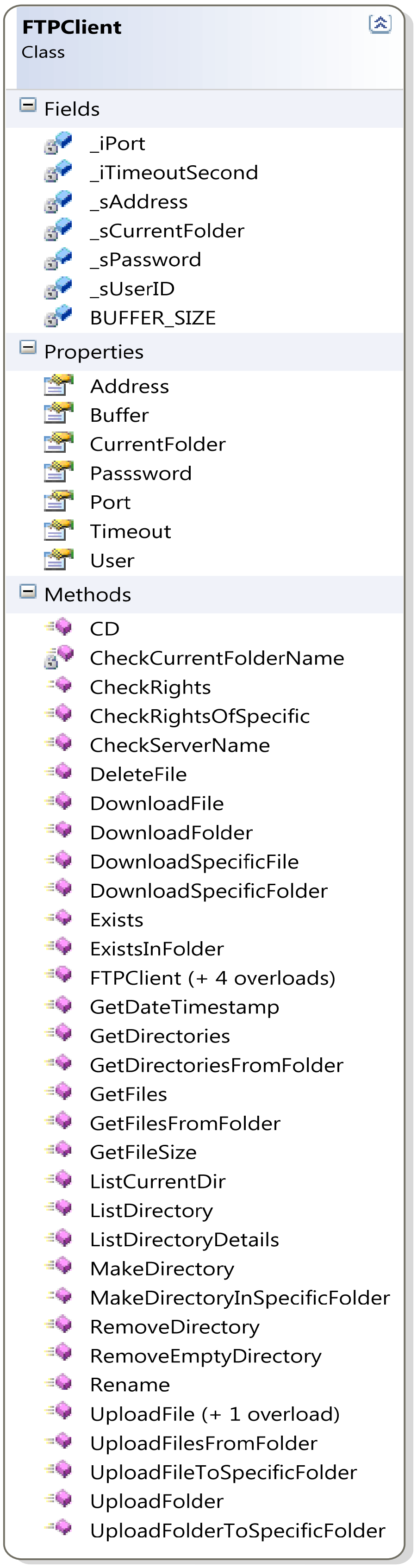


FIGURE 23. Logger class

3.10 FTP client

The class presented in FIGURE 24 is the richest one. It contains the whole needed functionality to become a base for a FTP client application. It was decided to do it in that way as it was mostly beneficial for the company. In the future that class may be easily contributed to other projects with the functionality it contains. The class has a very similar purpose as the *FileManager* class presented in the chapter 3.5 with the difference that the *FTPClient* class instance takes care of managing files and / or folders in the network. Hence, it contains the necessary fields and properties which allow software developers to configure it in the way they want.



FTPClient
Class

Fields
_iPort
_iTimeoutSecond
_sAddress
_sCurrentFolder
_sPassword
_sUserID
BUFFER_SIZE

Properties
Address
Buffer
CurrentFolder
Passsword
Port
Timeout
User

Methods
CD
CheckCurrentFolderName
CheckRights
CheckRightsOfSpecific
CheckServerName
DeleteFile
DownloadFile
DownloadFolder
DownloadSpecificFile
DownloadSpecificFolder
Exists
ExistsInFolder
FTPClient (+ 4 overloads)
GetDateTimestamp
GetDirectories
GetDirectoriesFromFolder
GetFiles
GetFilesFromFolder
GetFileSize
ListCurrentDir
ListDirectory
ListDirectoryDetails
MakeDirectory
MakeDirectoryInSpecificFolder
RemoveDirectory
RemoveEmptyDirectory
Rename
UploadFile (+ 1 overload)
UploadFilesFromFolder
UploadFileToSpecificFolder
UploadFolder
UploadFolderToSpecificFolder

FIGURE 24. FTP client class

Despite that it was created in the way to allow customization, some of the settings are hardcoded. The main mechanism used in this class - `FtpWebRequest` - has two fields set to predefined values. These fields are `FtpWebRequest.sePassive` which is set to false and `FtpWebRequest.KeepAlive` which is also set to false. The first option says that an application uses active data transfer process and the second option specifies that after the request is completed the control connection to the FTP server is closed.

The figure below shows the graphical user interface for the test purposes of the *FTPClient* class. It uses all functionality provided by a presented class and it is a very simple FTP client application.

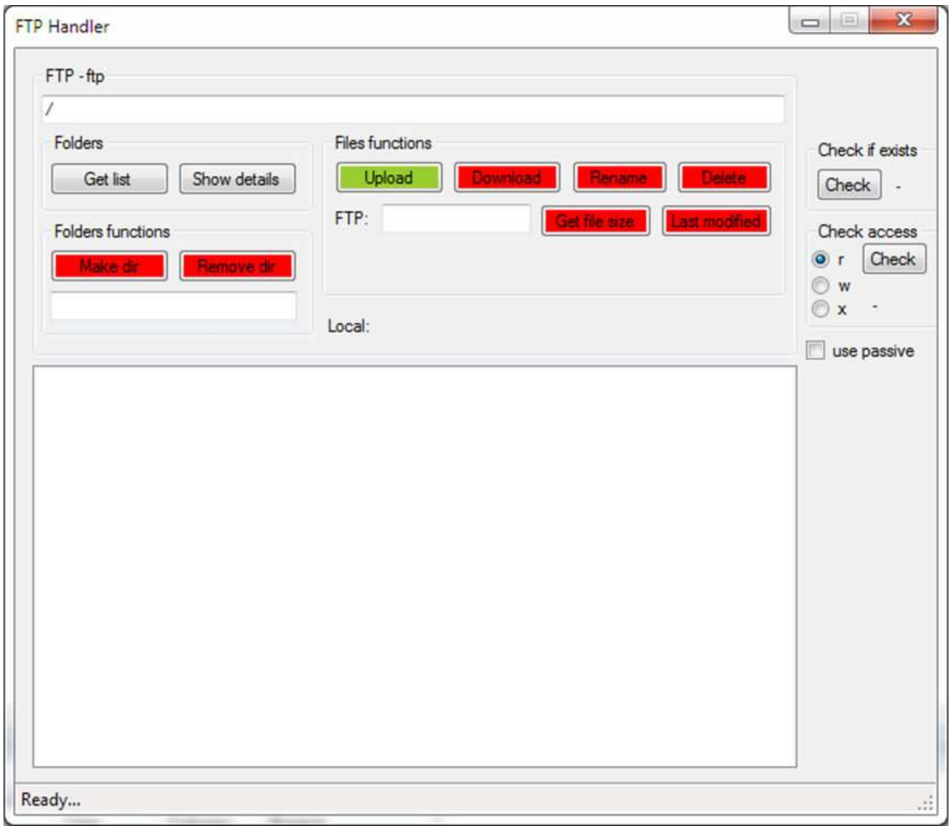


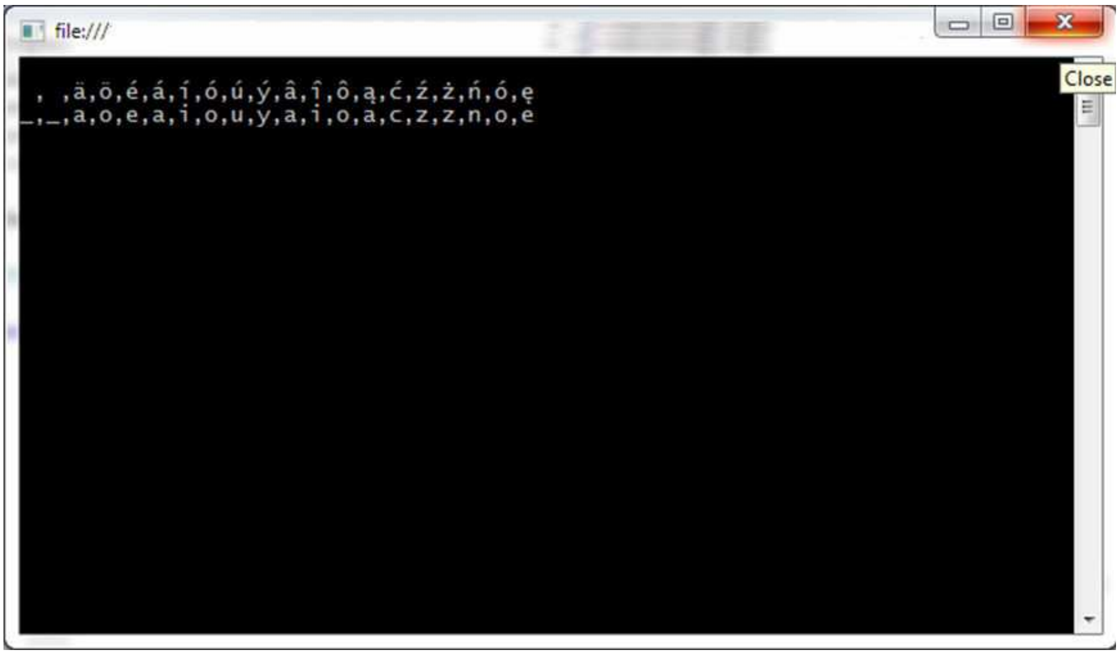
FIGURE 25. UI of test application using FTPClient class

3.11 Diacritic characters

In the Finnish and Scandinavian languages there are some special letters which are diacritic characters. The data which is stored in a database is used for generating

URLs, therefore such characters cannot be used in any part of the URL. To avoid keeping such letters in any string a special function was introduced to convert diacritics characters to US-ASCII encoding. This function also takes care of two other characters which are dangerous to put to an URL, namely: ‘ ‘(the space) and ‘/’ (the slash). The pace separates words in one a sentence but in the URL it is replaced by ‘%20’. On the other hand the slash separates different parts of the URL. If an application adds an entry from a database which contains the slash to an address it separates the URL in the place where it stands, which is not desired. It was decided that these two characters are replaced by underscore which is represented in the URL by itself - ‘_’ meaning underscore.

FIGURE 26 illustrates an example of how the function described in this chapter works. (Uniform Resource Locators (URL), 1994; Universal Resource Identifiers in WWW, 1994; Uniform Resource Identifiers (URI): Generic Syntax, 1998)



```
 , ,ä,ö,é,á,í,ó,ú,ý,â,î,ô,ą,ć,ź,ż,ń,ó,ę
_,_,a,o,e,a,i,o,u,y,a,i,o,a,c,z,z,n,o,e
```

FIGURE 26. Example of changing diacritics characters

3.12 iSync Runner

All the legacy code was changed and tested in the new .NET environment. It provides the same functionality as the previous PHP script but in a new systematized form.

The old script was run by another shell script from a command line. It caused that iSync was run every two minutes regardless if there were or were not any ready to proceed items in a database. That behavior was changed in a new application. For the purpose of managing iSync and running it, the application *iSync Runner* was introduced. It contains several advantages comparing to the previous solution:

- Checking database before running iSync – if there are no valid entries in a database iSync is not run
- Opening iSync log files
- Allowing to run iSync manually at any time
- Showing balloon tips when any actions is taken
- Allowing to create schedules – a frequency of a database checking varies and hence running iSync may differ on each day.

iSync Runner also contains the previously utilized functionality, namely automated periodic execution of iSync. This option, however, may be disabled. FIGURE 27 shows the main graphical user interface available to manage the script and monitor database.

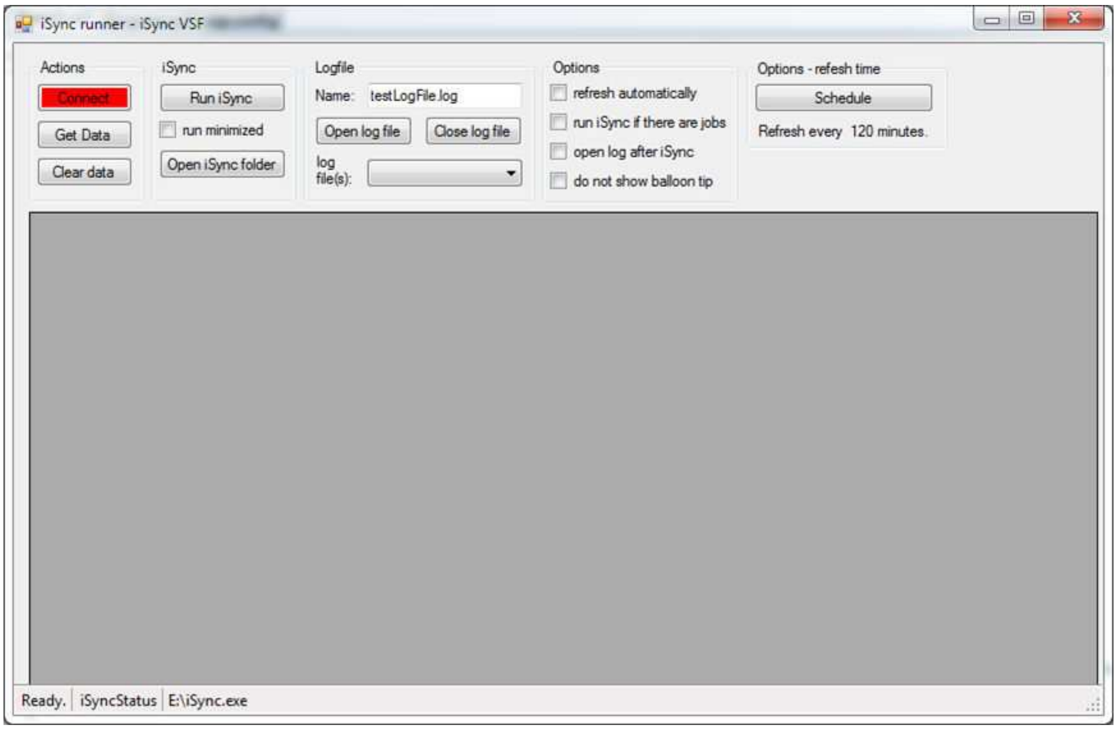


FIGURE 27. Application to manage iSync app

iSync Runner comprises an additional feature which is icons in the notification area. This allows minimizing it to a notification area and even from there controls iSync. FIGURE 28 shows the menus of *iSync Runner* available to the user from the notification area.

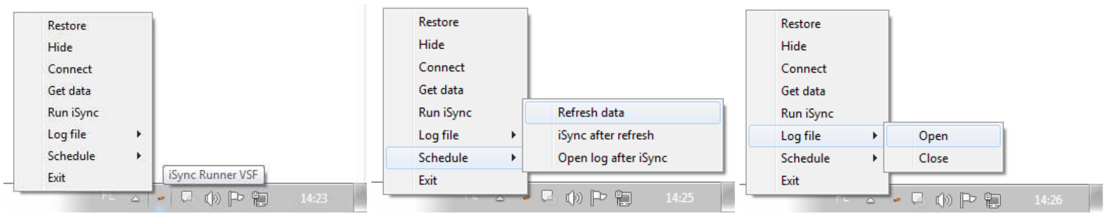


FIGURE 28. iSync Runner options in the notification area

4 CONCLUSIONS

4.1 Challenges

After familiarizing with the legacy PHP script and planning all stages of the code reimplementation, the process of transforming old code to the new environment has looked really promising. The work was planned for the whole practical training period, that is three months, and it included heavy application testing.

Code division

The old code was divided into three files, however, they did not group any certain functionalities. Because of that the code was not easily readable, flexible and easy to be modified. To avoid similar problems in the new code it was decided to group specific functionalities to certain classes and the *Main* function included only the following lines of code:

```
Manager _iSyncManager = new Manager();
_iSyncManager.MakeTemporatyDirectory();
_iSyncManager.GetDataFromMySqlDB();
_iSyncManager.CheckIfExit();
_iSyncManager.Process();
_iSyncManager.ClearTemporaryFolder();
_iSyncManager.RemoveTemporaryDirectory();
```

The other advantage of a new code is that it is self-explanatory as can be noticed from the fragment above.

Database connection

Building the new application has started from creating the necessary classes, which are used as tools while the application is run. These classes include: *DBHandler, FileManager* and *FTPClient*. The first of them is a data entry point of the application and it manages a connection with a MySQL database. This fetched data from database determines what type of activities will be executed by iSync. In the .NET documentation there is no information about its cooperation with MySQL databases. Lack of compatibility with MySQL databases caused that additional research in finding a proper solution was needed. It turned out that MySQL's owner provides the necessary library connecting .NET framework with MySQL database. After applying

MySQLConnector library no other major problems were encountered during further database handing implementation process. (MySQL::Download Connector/Net, 2010)

File management

Developing of the classes managing files and folders both locally by *FileManager* class and remotely in the network by *FTPClient* class was not problematic; it was rather time consuming. These both classes are versatile because they provide a full functionality and thus they can be used by a company also in the other projects. It required, however, creating full documentation containing all information about *FileManager* and *FTPClient* classes.

Images processing

One of the next stages was image processing, which also caused some problems. Image processing in iSync starts from image's resizing, more concretely downscaling. The application uses the following interpolation method: `System.Drawing.Drawing2D.InterpolationMode.HighQualityBilinear`, which provides one of the most accurate functionalities thanks to which the images lose neither on their quality nor their granularity. Everything worked correctly on Windows operating system, however, when the testing application presented in FIGURE 20 was run using Mono on Linux OS some problems with the quality of the images were noticed. Further investigation showed that using such a good interpolation method did not bring the expected results. What is more the images' quality from Linux OS did not change when changing interpolation mode and it was comparable with the images' quality from Windows OS only when using `System.Drawing.Drawing2D.InterpolationMode.Low` interpolation mode on Windows side. It was a major issue considering the images' role which was using them on the web pages as a part of the presentation. To solve this problem an external console tool – ImageMagic - was used. This is one of the dependencies and a requirement that has to be fulfilled if the application is to be used on Linux OS.

4.2 Improvements

In the practical part of the thesis the testing applications were presented. This approach of testing allowed an easy and convenient but a manual way of checking the functionality of the classes. Nevertheless, it would be much better to create separate solution in Visual Studio containing unit tests. It has a lot of benefits, most of all the whole functionality would be tested automatically at once. It would require much more effort; however, the final balance would have much more profits than losses, for example, any changes in the function's body would be immediately verified.

When telling about the other things that could be modified or improved the following issues cannot be omitted:

- *FileManager* class
- *Manager* class
- the other classes using configuration file.

The *FileManager* class does not contain any fields but only methods. It operates only on files and folders without loading or modifying their content. Hence this class could be a static one since then no objects of this class would have to be created in order to operate on filed or folders.

The *Manager* class is created only once in the *Main* function and it manages all operations done by iSync. Due to the fact that only one instance of this class has to be created it could also implement singleton design pattern as *Logger* class presented in the chapters 2.4 and 3.9 does.

Loading setting from the configuration file is another thing that could be modified. The classes like *DBHandler* or *Logger* should be the ones that could be enclosed by the other projects using their functionalities. The way of initializing some of their fields, however, prevents from using them by the other projects without former modifications. For example a field *_SelectCommand* in *DBHandler* class contains a hardcoded query to the MySQL database, which has read-only and private attributes, hence they are not supposed to be changed. It could be better, however, if MySQL query was customized by constructor's parameter or property.

The *Logger* class depends even more on iSync application itself. It contains static fields, two of which, namely *_FILENAME* and *_logMode*, are filled based on data from the configuration file. It should be definitely changed to allow integration into the other projects.

4.3 Summary

The refactorization of the legacy PHP script did not rely only on rewriting it into the new environment - .NET, but also introducing new order into the previously very chaotic code. Currently the functionalities related to the common areas are divided into separate classes, they were all tested using testing applications and the whole documentation is also created. The code of the present iSync is much more readable and it can be easily modified in the future if needed. Retrospectively it can be clearly visible that there could be some changes done already at the last stage of the application just to improve its functionality in the current project. Unfortunately the short period of the practical training and the complexity of the iSync did not allow the author to introduce any additional improvements. The main goal of the development, however, was achieved. The application uses many technologies and hence the author was able to broaden his knowledge from different areas. What is more he has created the new application, which has the same functionality as a legacy code and can be easily extended or enriched by the other developers.

REFERENCES

.NET Design Pattern and Architectural in C# and VB. (2010). Retrieved from http://www.dofactory.com/Default.aspx

Albabari, J., & Albabari, B. (2010). *C# 4.0 in a Nutshell.* O`Reilly Media.

Bellovin, S., & Laboratories, A. B. (1994, February). *RFC 1579 - Firewall - Friendly FTP.* Retrieved January 19, 2010, from IETF Documents: http://tools.ietf.org/html/rfc1579

Exploring the Singleton Design Pattern. (2010). Retrieved Septemer 16, 2009, from MSDN Library: http://msdn.microsoft.com/en-us/library/ee817670.aspx

Extensive Markup Language (XML) 1.0 (Fifth Edition). (2008, November 26). Retrieved January 23, 2010, from All Standarfs and Drafts: http://www.w3.org/TR/REC-xml/

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1994). *Design Patterns: Elements of Reusable Object-Oriented Software.* Boston: Addison-Wesley.

Gilmore, J. W. (2008). *Begininning PHP and MySQL.* New York: Springer-Verlag New York.

ImageMagick: Convert, Edit and Compose Images. (2009). Retrieved from ImageMagick: http://www.imagemagick.org/script/index.php

Implementing the Singleton Pattern in C#. (2006, January 7). Retrieved from http://www.yoda.arachsys.com/csharp/singleton.html

Krishna Rao, R. B. (2007). *Programming with C#: Concepts and Practice.* Delhi: Asoke K. Ghost.

Liberty, J., & Xie, D. (2008). *Programming C# 3.0* (Fifth Edition ed.). O`Reily Media.

Mamone, M. (2006). *Practical Mono.* New York: Springer-Verlag.

Mono. (2004, June). Retrieved February 18, 2011, from http://www.mono-project.com/Main_Page

MySQL :: MySQL 5.1 Reference Manula :: 20.2 MySQL Connector/NET. (2010). Retrieved December 09, 2010, from MySQL :: Developer Zone: http://dev.mysql.com/doc/refman/5.1/en/connector-net.html

MySQL::Download Connector/Net. (2010). Retrieved from http://www.mysql.com/downloads/connector/net/

Nash, T. (2007). *Accelerated C# 2008.* New York: Springer-Verlag.

Nash, T. (2010). *Accelerated C# 2010.* New York: Springer-Verlag.

Postel, J., & Reynolds, J. (1985, October). *RFC 959 File Transfer Protocol.* Retrieved January 21, 2010, from IETF Documents: http://tools.ietf.org/html/rfc959#appendix-I

Schonig, H. J., & Geschwinde, E. (2004). *Mono Kick Start.* Sams Publishing.

Singleton Design Pattern in C# and VB.NET. (2010). Retrieved September 16, 2009, from .NET Design Patterns and Architectures in C# and VB, .NET training, DoFactory: http://www.dofactory.com/Patterns/PatternSingleton.aspx

Suehring, S. (2002). *MySQL Bible.* New York: Wiley Publishing, Inc.

Swaminathan, R. K. (2007). *Cross-Platform Application Development using .NET and Mono.* Waterloo: University of Waterloo.

Thai, T., & Lam, H. (2002). *.NET Framework Essentials* (2nd ed.). Gravenstein Highway North: O'Reilly.

tortoisesvn.tigris.org. (2010). Retrieved February 20, 2010, from http://tortoisesvn.tigris.org/

Uniform Resource Identifiers (URI): Generic Syntax. (1998, August). Retrieved 11 20, 2010, from Web Naming and Addressing Overview: http://www.ietf.org/rfc/rfc2396.txt

Uniform Resource Locators (URL). (1994, December). Retrieved September 20, 2010, from RFC-Editor Webpage: http://www.rfc-editor.org/rfc/rfc1738.txt

Universal Resource Identifiers in WWW. (1994, June). Retrieved 11 20, 2010, from Web Naming and Addressing Overview: http://www.w3.org/Addressing/rfc1630.txt