Behavior recording with the scoring program MouseClick

A study in cross platform and precise timing developing

Erik Karlsson
Abstract

This thesis will deal with problems and solutions of cross-platform developing using MoNo framework as a replacement of Microsoft .NET framework on Linux and Mac OS-X platforms. It will take into account matters such as limitations in the filesystem to problems with deploying released programs. It will also deal with demands of precise timing and the need of efficient code on precise tasks to construct a program used for creating data from recordings of animals. These animals is set to perform a task, for example exploring a labyrinth or running on a rod, and it is all recorded on video. These videos are later reviewed by an observer which transcripts the recordings into data based on predefined behaviors and the time and frequency with which the animal is expressing them.
Keywords

• C#
• MoNo
• .Net-Framework
• Cross-platform
• Precision timing
• Threading
• Spreadsheet automation
• Behavior studies
Table of Contents

1 Introduction ......................................................................................................................... 1
  1.1 Background .................................................................................................................. 1
  1.2 Purpose ....................................................................................................................... 1
    1.2.1 Thesis .......................................................................................................................... 1
  1.2.2 Work ......................................................................................................................... 2

2 Methods ............................................................................................................................. 2

3 Limitations .......................................................................................................................... 3

4 Concepts .............................................................................................................................. 3

5 Dissertation .......................................................................................................................... 4
  2.1 Developing Environments .............................................................................................. 4
    2.1.1 Visual Studio .............................................................................................................. 4
    2.1.2 MonoDevelop ............................................................................................................ 5
    2.1.3 Comparison between the developing environments .................................................. 5

6 The Frameworks .................................................................................................................. 5
  2.2.1 .Net-Framework .......................................................................................................... 5
  2.2.2 Mono-Framework ....................................................................................................... 6

7 Level of Implementation ...................................................................................................... 6
  2.2.3 ..................................................................................................................................... 6

8 Components .......................................................................................................................... 9
  2.3.1 Timer .......................................................................................................................... 9
  2.3.2 Stopwatch .................................................................................................................. 10
  2.3.3 Backgroundworker .................................................................................................... 12

9 Development diary .............................................................................................................. 14
  2.4.1 December 2009 .......................................................................................................... 14
  2.4.2 January 2010 .............................................................................................................. 14
  2.4.3 February 2010 ........................................................................................................... 15
  2.4.4 March 2010 ............................................................................................................... 17
  2.4.5 April 2010 .................................................................................................................. 18
  2.4.6 May 2010 .................................................................................................................. 19

10 Closing ................................................................................................................................ 20
  3.1 Result ............................................................................................................................. 20
    3.1.1 Starting up .................................................................................................................. 20
    3.1.2 Recording ................................................................................................................... 24

11 Conclusion .............................................................................................................................. 24

12 Summary ................................................................................................................................ 30

13 Bibliography .......................................................................................................................... 31

14 Theresis .................................................................................................................................... 33
Illustration Index

Illustration 1: Legend used to describe level of implementation..................................................6
Illustration 2: Features in .Net-framework 1.1 present in Mono..................................................6
Illustration 3: Features in .Net-framework 2.0 present in Mono..................................................7
Illustration 4: Features in .Net-framework 3.0 present in Mono..................................................7
Illustration 5: Features in .Net-framework 3.5 present in Mono..................................................8
Illustration 6: Features in .Net-framework 3.5 present in next version of Mono..........................8
Illustration 7: Start view of the program....................................................................................20
Illustration 8: File menu............................................................................................................20
Illustration 9: Create a study window..........................................................................................21
Illustration 10: Main window as shown when a study have been defined.................................22
Illustration 11: Studymenu..........................................................................................................22
Illustration 12: Empty behavior info window.............................................................................22
Illustration 13: Completed behavior info window.......................................................................23
Illustration 14: Main window with one behavior.......................................................................23
Illustration 15: Main window with three behaviors, how to start...............................................24
Illustration 16: Start of recording information.............................................................................24
Illustration 17: Main window, first behavior recording...............................................................25
Illustration 18: Main window, recording paused.........................................................................26
Illustration 19: Main window, third behavior recording...............................................................26
Illustration 20: Main window, recording finished.........................................................................27
Illustration 21: Main window, Study menu................................................................................27
Illustration 22: Review events window.......................................................................................28
Illustration 23: Review events window, intersection....................................................................28
Illustration 24: Main window, File menu.....................................................................................29
Illustration 25: Main window, New Animal dialog box..............................................................29
Illustration 26: Main window, New Animal replace warning......................................................30
Appendix Index

Appendix 1: Example of settings file
Appendix 2: Example of Excel-file
Appendix 3: Example of CSV-file
Appendix 4: Example of timeline
Appendix 5: Letter of reference
Acknowledgments

I especially want to thank Åsa McKenzie and Klas Kullander, at Neurolab, BMC, Uppsala University in Uppsala, Sweden, for their support and willingness to try out my work.

And also to Karin Nordenankar at the same lab for her help with bugtesting and evaluating the program. Special thanks to Mats Nordström, Department of Informatics and Media, Ekonomikum, Uppsala, Sweden for his help in the beginning of the project.

I want to thank my supervisor, Torsten Palm, for approving this work to be done and supporting me in doing it.

And thanks to Nadine Schweizer for being my sounding board in the beginning and giving me many good suggestions for improving the work.
1 Introduction

This work is about how to create a program, based on requests and demands of the future users of the program, specifically: How to deal with cross-platform developing with the language C# developed by Microsoft and the Mono-Framework developed by Mono-Project. To understand it, a program will be created in collaboration with Nuerolab at BMC, Uppsala. This program will be a so called Scoring program. Scoring is used when animals with and without certain changes in the their genome have been set to be performing tasks in a controlled environment and then compared with a animal that is so to say not altered. This is done to find out what different genes manipulation of those do to find out best way of treatments and new medicines.

1.1 Background

Studyrecording, scoring, is a way to measure differences in behaviors between different animals, one breeding (knockout) with a different genotype than the original animal (control). These animals are then set to perform different tasks which are recorded on video, and later, scored from the recording by an observer. This observer uses a program in which keys are assigned to behaviors which are defined when the study is projected. The observer presses the key which corresponds to whatever specific behavior the animal currently expresses in the video, and when the animal changes behavior, the observer simultaneously changes the key. These recordings, eventdata, are then put together and statistical operations are performed on these datasets, to look for significant changes. The current programs used all work well and are well tested. Even so, some have different flaws, for example bad performance, inaccurate output because of interval switching or prohibitively high costs to be viable to use.

1.2 Purpose

The purpose of this project is to learn more about programming and set up the development with all the stages of planning and production. How to plan and implement an application that is targeting several different platforms, and take it out to release to the designated users. The users are working with scoring on a daily basis, and with their help the program will serve as a resource in their work. The thesis will aid in understanding the program and the process of development, to them and to others.

1.2.1 Thesis

The purpose of this thesis is to go through which components to use in creating a time precise program for scoring behaviors. These components are included in the programming language C# used in the program. The components used in the program will be described, how and when to use them and their limitations. The results will be discussed, if the development went as planned, and show off the program as it looked when this thesis was finished. The
development will be presented as a journal with sections for each month with major events and edits to the code and the program.

1.2.2 Work

The purpose of this work is to create a reliable and accurate program used to sample behavior studies. Some of the currently used products express various drawbacks and bugs, which this program aims to eliminate. The goal with the program is to have it running in Microsoft Windows as well as in Macintosh OS X. This will be possible thanks to an open source implementation of the Microsoft .Net-framework called the Mono-project. The finished program is aimed to be lightweight and easily deployed and used by the designated users.

1.3 Methods

I will mainly write about key parts of my program, the use of these and why I choose them. I will also write about the .Net-framework from Microsoft and the Mono-framework from the open source project Mono-Project and compare those. I will present the two mainly used platforms for developing, Visual Studio from Microsoft and Mono-Develop.

One main and big problem with this approach is that there is not a lot of available documentation in books or papers in this area. There exist a few books about Mono, but no one have been published recently which makes them to outdated to be any use as the area of computers change rapidly. Therefore the main sources of information is various online resources, with the Microsoft programming website and community MSDN (Microsoft Developer Network) as the main source.

I chose to do the thesis based on the Design and Creation method, where I had the help from customers for the program I developed. Interviews were performed with several users of other scoring programs. They provided feedback and suggested changes, but they also found most of my first solutions usable out of the box. When a beta version was finished in late April, it got sent out on evaluation to the people that I had previously interviewed.

The Design and Creation methods defines a problem solving approach to the problem, Learning via making. The method can be divided in five subcategories, Awareness, Suggestion, Development, Evaluation and Conclusion (RISC).

Awareness of the problem can be obtained in various ways. With this program, interviews were the most effective way to get background and an understanding of the problem. To some extent, research on old programs, as well as reading up on technology were used to gather information.

Suggestion, the second step, discussions with teachers and reading articles found ways of dealing with the problem, making it solvable. A big part of the knowledge needed to complete this program was gathered from the internet.
Development is by far the biggest part of this thesis, and the biggest part of the practical work. Most work done here was made directly for the user. This way, different methods of implementations could be easily and quickly tested for practical usability. A lot of work was put into error handling, and a big part of the code is to prevent accidents and bad output with poor or non-existing input. The program was tested continuously throughout the development phase by running it on fictional data as well as real recordings with the help of users from the Neurolab at BMC.

Evaluation was performed throughout the development as part of the development to ensure that the final product is what the users wants or altered to fit the demands from the users.

Conclusion in this project was reached when the program was ready for deployment. Deployment consists of introduction to the new program, both personal and as a presentation of this work.

1.4 Limitations
The paper will be limited to involve techniques that were used for the program and the decisions that were concluded in the process of making the program, and why they were made. In the sense of cross-platform, it will only deal with the use of Mono for use on other systems, Java will not be dealt with as this work is to learn the C# language on the Mono-framework. It will not contain any real code, just suggestions and explanations on some of the used components. It will not deal with how to program in C# or any of the discussed developing environments.

1.5 Concepts
Framework – A framework is a library of existing code for easy reuse when programming. It can as well often be modified with added functionality that is needed in the context.

Event – Sometimes when something happens, a event is fired. For example when a list is changed, when the user moves the window or when a user presses a key. This is useful to alert a user to what happens in the program, or trigger another thing to happen.

Groupbox – A control that is used to group controls to one unit, for easier modification and altering.

Textbox – A box used to present or input text to and from the user.

While-loop – Executes code until a specific condition is met. Is exclusive and blocks the thread it is running in, and can often give the user the illusion that the program has stopped responding.

Namespace – Groups of classes and functions that are similar to each other and therefore logically are grouped together to make them easier to locate.
2 Dissertation

This part consists of a comparison of the main developing environments used and a comparison of the the .Net-framework and the Mono-framework. There will also be a developing diary, a presentation of key components used in the program.

2.1 Developing Environments

A developing environment is also referenced to as a Integrated development environment, abbreviated IDE.

A developing environment is program used for developing program, comes with four main parts:

- Editor
- Compiler
- Debugger
- Build automation tools

The editor is used to create or edit source code files. The editor can contain help files for different languages which help with auto-completion of code or with information about methods and functions.

A compiler translates the human readable source code into code executable by the computer. A good compiler can help with bad programming were it is able to optimize the code when the programmer might have made it more complex then necessary.

A debugger is the best companion a programmer can have. If the code fails to run, the debugger points out where it went wrong and why. That is extremely helpful as it can save hours of manual troubleshooting.

The build automation tools is usually the last part used. With those tools the IDE can output the program as files or installs that the programmer can distribute to customers. The programs are then ready to run out of the box.

2.1.1 Visual Studio

Created by Microsoft, and for a long time one of few complete developing environments for C#. It uses the System.Windows.Forms namespace for creation of a user interface. The Mono framework have a wrapper for that namespace so applications written in Visual Studio can run on all platforms currently support by Mono.


2.1.2 MonoDevelop

Created mainly for the Linux platform, written completely in C#. Started as a migrating project from the program SharpDevelop, which is a open source developing environment written for windows in which you work against the Microsoft .Net-Framework. MonoDevelop, where you can develop for both the Mono framework as well as for the .Net-Framework, uses Mono to run in various Linux distributions and in the Macintosh operating system OS X. There is also a release for the Windows operating system. Latest release is MonoDevelop 2.2.2 from March 18, 2010 (MonoDevelop). That is the release used for the cross-platform testing of the program on Os X.

Official support for the Windows and Os X platforms came as late as in the 2.2 release at December 15, 2009.

2.1.3 Comparison between the developing environments

The main difference between Visual Studio and MonoDevelop is the way they are developed. Microsoft have as the inventor and maintainer of the .Net-framework and the languages built upon it, like C#, a clear advantage in the way of implementations of new additions to the language. The Mono community which uses the Mono framework usually has to wait until the specification are released by Microsoft, and then implemented in the Mono framework until MonoDevelop is on the same level as Visual Studio again. But what MonoDevelop loses in current-ness it makes up in platform support. With the help of the GTK# library it can be used to create applications which natively blend with the underlying operating system. For example in Os X an application written with GTK# as user interface will be able to dock in the program bar, as well utilize the common menu bar on the top of the screen in Os X (GTK#).

2.2 The Frameworks

Both the original .Net-framework and the open source Mono-framework take the same code and make a program out of it, then what is the difference?

2.2.1 .Net-Framework

The .Net-Framework is the standard framework used with for example the C# language. It is the de facto standard way of running C# applications (at least on the Windows platform) for one reason; Microsoft. They are responsible for developing both the different languages used by it as well as the biggest Developing Environment used to construct programs. That means that they decide what will be included in new releases of the .Net-framework, and that they because of that, they have a big advantages in time ahead of other frameworks (Mono).
2.2.2 Mono-Framework

Being based on what the .Net-framework has to offer, the differences cannot be too huge. The main selling point, and it is a big one, for Mono is the Cross-platform compatibility. As of 19th of May 2010, Mono promises support for not less than eight platforms and more than nine different CPU architectures! Main platforms are apart from the original support for Linux, support for Windows as well as for Mac OS X. More interesting platforms are the video game console Wii and the Apple’s smartphone iPhone.

2.2.3 Level of Implementation

This is the current implementation of features from the .Net-framework in the Mono-framework 2.6, released December 2009.

In short this quote from the Mono-project explains the current state of Mono:

”The easiest way to describe what Mono currently supports is:
Everything in .NET 3.5 except WPF and WF, limited WCF. “

The long version are a bit more interesting and more explaining.

The legend in Illustration 1 is used to define level of implementation in the following illustrations in this chapter. Implemented means that the features are fully useable in Mono. Partially Implemented means that some minor features are missing, or the feature can be used with a workaround. Not implemented means that it is not available at all so far, and in some cases there is no plan on implementing these.

.NET 1.1

Illustration 2: Features in .Net-framework 1.1 present in Mono
Mono implements almost 100% of the features present in .Net-framework 1.1 which were released in 2003, see Illustration 2. System.Management are not implemented as it depends on the Windows Management Instrumentation (WMI) infrastructure, which can't be used on the main platforms for Mono (Linux & Mac OS X) (MSDN Management).

**.NET 2.0**
- C# 2.0 (generics)
- Core Libraries 2.0: mscorlib, System, System.Xml
- ASP.Net 2.0 - except WebParts
- ADO.Net 2.0
- Winforms/System.Drawing 2.0 - does not support right-to-left

*Illustration 3: Features in .Net-framework 2.0 present in Mono*

As seen in Illustration 3, the 2.0 version of the .Net-framework is fully implemented in Mono, where only some minor details have been excluded. The Winforms lack of support for right-to-left text representation style comes from lacking support in the underlying mechanisms Mono uses to present text.

**.NET 3.0**
- WCF - silverlight 2.0 subset completed
- WPF - no plans to implement
- WF - Will implement WF 4 instead on future versions of Mono.

*Illustration 4: Features in .Net-framework 3.0 present in Mono*

The 3.0 .Net-framework has not been prioritized to implement into Mono. The reason for this is the lack of new features in this release, see Illustration 4. Focus on this release lays more on new ways to graphically present the program for the user (WPF, Windows Presentation Foundation). Not a bad thing in itself, as the standard Windows forms representation lacks support for more advance graphical ways of drawing things. WCF (Windows Communication Foundation) is a way to create services, seen as a gadget to the user, which is used to communicate between applications or between applications and users. The Silverlight 2.0 part of WCF have been remade to fit mono. That release is called Moonlight. Silverlight is a direct competitor to flash as it can be used on websites for games, video or other media as well as giving the developers the ability to create high-end programs using the .Net-framework (Silverlight 2).
As seen in Illustration 5, the version 3.5 of the .Net-framework was in most ways a larger release in terms of added features and functionality then version 3.0. Most noticeable additions are LINQ (Language Integrated Query) and Asp.Net MVC. LINQ is a very fast and effective way of reducing and picking sets of data from larger datasets and assigning it to structures or classes for easy representation. This is done in several different ways, for example with GROUP and WHERE clauses. This is similar to real SQL-queries, but with the advantage that it is now easy to assign the returned values as objects. ASP.Net MVC is a way of constructing websites complete with database access, data manipulation and finally representation to the user. MVC stands for Model–View–Controller, intended to separate the programming in these parts to make it easier to overview.

The newest .Net-framework, 4.0, was released as late as April 12th 2010 (.Net 4.0 Launch). Most important new features would include the improved multi-core support as well as the new addition PLINQ (Parallel Language Integrated Query). PLINQ is an improved version of LINQ introduced in the 3.5 version of the .Net-framework. PLINQ will automatically perform LINQ standard queries distributed among the cores on the host computer. This will in most cases lead to increased performance with better used calculation power and efficiency (PLINQ). Illustration 6 shows the status of the upcoming release of Mono. Most of .Net 4.0's features are already available in the upcoming release of Mono, and it can already be accessed from the Mono-projects website but without current official support (Mono Compatibility).

Version 2.0 is the version used in this program, as it is fully implemented with good support and more features then the original 1.1 .Net-framework. Newer version have been mostly implemented in Mono, but the older 2.0 were chosen to not jeopardize the reliability of the program when running it with Mono. The program will be extended into the .Net-framework
4.0 when Mono reaches that level, as many of the new features in both C# 3.5 and 4.0 are really interesting and can improve the functionality of this program.

(Mono Compatibility)

2.3 Components

Here I will present different built in components from the C# programming language and the .NET-Framework, which C# is built upon, which I use in this program. I will give my view of them and give quotes from the publicly available MSDN library found at http://msdn.microsoft.com/en-us/library/default.aspx

2.3.1 Timer

The timer, a great tool for events that happen regularly and need to be taken care of at the same time as other events. Easy to use with small amount of methods and properties, but lacks accuracy and can be haunted by various bug regarding to queuing and stalling the events firing.

“A Timer is used to raise an event at user-defined intervals. This Windows timer is designed for a single-threaded environment where UI threads are used to perform processing. It requires that the user code have a UI message pump available and always operate from the same thread, or marshal the call onto another thread.

When you use this timer, use the Tick event to perform a polling operation or to display a splash screen for a specified period of time. Whenever the Enabled property is set to true and the Interval property is greater than zero, the Tick event is raised at intervals based on the Interval property setting.

This class provides methods to set the interval, and to start and stop the timer.”

As stated in the text above, this timer can only operate at the same thread as the user interface causing it to lack the benefits, and drawback, which multithreaded programming gives. It is also written that the events fire as soon as the interval is set to higher then zero (and the timer is enabled). This is a truth with modification as the timer is not that precise. The interval is measured in milliseconds which theoretically gives it the option to fire events every 1/1000ths of a second. Not bad. However, the timer comes with a small, but critical, note:

“Note
The Windows Forms Timer component is single-threaded, and is limited to an accuracy of 55 milliseconds. If you require a multithreaded timer with greater accuracy, use the Timer class in the System.Timers namespace.”

Having a timer with that small accuracy is not good, especially in this program where we want to be precise down to millisecond level. The timer went from a theoretically minimum
rate of fire at 1/1000th of a second to practical minimum accuracy of approximately 1/20th of a second. Not good.

(MSDN Timer)

2.3.2 Stopwatch

This class really lives up to its name. A really simple class with the sole purpose of counting the time, just as its old analog namesake. You start it, it reads time and then stops it. Then you can read the time again and finally reset it to be able to start another round of timekeeping.

“A Stopwatch instance can measure elapsed time for one interval, or the total of elapsed time across multiple intervals. In a typical Stopwatch scenario, you call the Start method, then eventually call the Stop method, and then you check elapsed time using the Elapsed property.”

This Elapsed property is of the type TimeSpan. From that it is possible to extract the passed time in various types. You can get a timestamp on how long the it has been running, the amount of hours, minutes, seconds and milliseconds. You can also get the elapsed time in various outputs, for example only hours or as a total of the amounts of milliseconds it has been running.

In this program I mainly call and ask for the lapsed seconds, for start and stop of behaviors. I also ask for the total time as hours, minutes and seconds to view the total time recorded so far to the user.

“A Stopwatch instance is either running or stopped; use IsRunning to determine the current state of a Stopwatch. Use Start to begin measuring elapsed time; use Stop to stop measuring elapsed time. Query the elapsed time value through the properties Elapsed, ElapsedMilliseconds, or ElapsedTicks. You can query the elapsed time properties while the instance is running or stopped. The elapsed time properties steadily increase while the Stopwatch is running; they remain constant when the instance is stopped.”

The property IsRunning is used to define when a user can record a behavior, as if the recording is paused it shouldn't be possible to modify the data that is being recording. Start is called when the user presses the first key after the recording is suppose to start. Stop is called either when the user aborting the recording or if the time runs out and the recording ends.

ElapsedMilliseconds were used early in the development, but later changed to the TotalSeconds part of the property Elapsed. This was done to eliminate conversion from milliseconds both when displaying the last recorded event as well as when saving and exporting every event. It was also done to ensure that the code was threading safe, as a Int64 (which the ElapsedMilliseconds is) was saved and processed in two parts in the memory rendering potential bugs and errors. A Double which the TotalSeconds property is doesn't have this drawback, and it is not needed to convert it as it is already expressing the time as seconds.
“By default, the elapsed time value of a Stopwatch instance equals the total of all measured time intervals. Each call to Start begins counting at the cumulative elapsed time; each call to Stop ends the current interval measurement and freezes the cumulative elapsed time value. Use the Reset method to clear the cumulative elapsed time in an existing Stopwatch instance.”

As the time is recorded as a long interval, Reset is only called when the recording have ended. Start and Stop are called when the recording is paused and resumed, as well as when starting and stopping the recording.

“The Stopwatch measures elapsed time by counting timer ticks in the underlying timer mechanism. If the installed hardware and operating system support a high-resolution performance counter, then the Stopwatch class uses that counter to measure elapsed time. Otherwise, the Stopwatch class uses the system timer to measure elapsed time. Use the Frequency and IsHighResolution fields to determine the precision and resolution of the Stopwatch timing implementation.”

The Stopwatch class assists the manipulation of timing-related performance counters within managed code. Specifically, the Frequency field and GetTimestamp method can be used in place of the unmanaged Win32 APIs QueryPerformanceFrequency and QueryPerformanceCounter.”

The program checks at startup if the Stopwatch-class is capable of high resolution recording by querying the IsHighResolution property. Most computers have the possibility to use this, even before the Stopwatch-class was released with the 2.0 version of the .Net-framework. If a computer doesn't have it, the stopwatch-class automatically falls back on the less accurate but always used system clock. Even though the accuracy might be less then with a high resolution timer, it is still good enough for this kind of recordings.

“Note
On a multiprocessor computer, it does not matter which processor the thread runs on. However, because of bugs in the BIOS or the Hardware Abstraction Layer (HAL), you can get different timing results on different processors. To specify processor affinity for a thread, use the ProcessThread.ProcessorAffinity method.”

As more and more of new computers have a processor with more than one core giving them the functionality to process several threads at the same time. But that gives rise to problems as processes can be shifted between cores as the operating system deals with uneven loads on the cores. To prevent this, processes can be assigned to a specific core by the program.

In this program the function that keeps track of time and intervals is given highest priority and the rest second highest priority by the system.

(MSDN Stopwatch)
2.3.3 Backgroundworker

The official workhouse in the shadows, the Backgroundworker is provided in the framework as a helper to make the program execute several demanding tasks at the same time.

“The BackgroundWorker class allows you to run an operation on a separate, dedicated thread. Time-consuming operations like downloads and database transactions can cause your user interface (UI) to seem as though it has stopped responding while they are running. When you want a responsive UI and you are faced with long delays associated with such operations, the BackgroundWorker class provides a convenient solution.”

The background worker works separated from the main thread which handles the user interface but are at the same time depending on the main thread, the foreground thread. When all the foreground threads stopped running the execution of the program stops, even if any active background threads are still running.

As the main functionality in this program is achieved by user interaction through the user interface with the keyboard, a background worker is needed to deal with code that might interfere with the user input.

“To execute a time-consuming operation in the background, create a BackgroundWorker and listen for events that report the progress of your operation and signal when your operation is finished. You can create the BackgroundWorker programmatically or you can drag it onto your form from the Components tab of the Toolbox. If you create the BackgroundWorker in the Windows Forms Designer, it will appear in the Component Tray, and its properties will be displayed in the Properties window.”

The background worker is used mainly as it is but with the addition of supporting cancellation by setting the property WorkerSupportsCancellation to true. The default value is false and if it isn't changed the worker won't be able to stop work being executed.

To be able to restart and cancel a recording, the worker needs to support cancellation. Otherwise the user has to wait for the previous recording to run out, even if it is not visible to the user that it is happening.

“To set up for a background operation, add an event handler for the DoWork event. Call your time-consuming operation in this event handler. To start the operation, call RunWorkerAsync. To receive notifications of progress updates, handle the ProgressChanged event. To receive a notification when the operation is completed, handle the RunWorkerCompleted event.”

The DoWork event in the background worker is the main part of the class, and stands for the main processing of work. In there code which take computation power is placed to not interfere with the user interface.

This program utilizes the background worker to run a while loop as long as the stopwatch is running and constantly checks the time passed to trigger new intervals or end the recording.
“Note:
You must be careful not to manipulate any user-interface objects in your DoWork event handler. Instead, communicate to the user interface through the ProgressChanged and RunWorkerCompleted events.

BackgroundWorker events are not marshaled across AppDomain boundaries. Do not use a BackgroundWorker component to perform multithreaded operations in more than one AppDomain.”

The changing user interface object DoWork event are allowed, so it is recommended to call the user interface thread with the events ProgressChanged and RunWorkerCompleted. The first one can be changed as the works progresses, if it is possible to get the amount of percentage of the work that has been done. The second event is triggered to run when the backgroundworker is done with the assigned task.

The program only triggers the RunWorkerCompleted event. It is used to give a control of that the previous task is completed before restarting the worker. Because trying to start the background worker again, without it being finished, throws an error.

“If your background operation requires a parameter, call RunWorkerAsync with your parameter. Inside the DoWork event handler, you can extract the parameter from the DoWorkEventArgs.Argument property.”

Some implementations of a background worker might need arguments for it to start working with. An example would be having a functions which reads large files or lots of files from a folder. Then an argument of the file or folder path might be necessary.

This is not implemented in the program, as it only works on already defined static variables that are set when a study are defined.

“For more information about BackgroundWorker, see How to: Run an Operation in the Background.

Note:
The HostProtectionAttribute attribute applied to this class has the following Resources property value: SharedState. The HostProtectionAttribute does not affect desktop applications (which are typically started by double-clicking an icon, typing a command, or entering a URL in a browser). For more information, see the HostProtectionAttribute class or SQL Server Programming and Host Protection Attributes.”

The attribute HostProtectionAttribute are only needed when setting up programs other then a desktop application. It is therefore not necessary to be taken into account in the program.

(MSDN BackgroundWorker)
2.4 Development diary.

I will present this work as diary, where I present the work I did over time divided into how the developing progressed. I will discuss what I did, why I did it, and how the outcomes became.

2.4.1 December 2009

The development started in late 2009 as a hobby project for learning and understanding how to implement logging key captures. At first the project was simple, having a single form with nothing in it, capturing keys and outputting it as a CSV (Character Separated Values) file.

A timer is a built-in class in the framework, which contains a way of performing a task on a regular interval. This task is performed in a Tick event, which in this case is just assigned to increase a integer, time, every tenth millisecond. The idea was to give a accuracy on 1/100 of a second level.

The integer time will be read once when a key is pressed and saved as start time for that instance. When the key is released, the time is read again and saved as stop time and from that a duration and latency can be calculated. This is defined as an event. At this point an event contains definitions of how to export itself as a line of text to a CSV file and a definition of how to represent itself as a filled rectangle on the time graph, as well as the integers start, stop, duration and latency.

2.4.2 January 2010

In the later part of January 2010, development was coming closer to a usable product which could be evaluated in collaboration with the users from Neurolab. A custom control have been implemented into the program, an extended groupbox, which contains controls from various other types of controls. It had been custom made and contained textfields for showing assigned key, name, duration, latency and the new fields frequency and total time. It also contains instances of the global functions of capturing keys, so it is now possible to capture several keys at the same time. This custom groupbox is called a Behavior, symbolizing the meaning of everything it contains. When the chosen key is pressed down, the custom groupbox, the Behavior, saves the time when it was pressed into the integer 'start'. At the same time the background color changes into yellow to show that this specific behavior is recording. When the corresponding key is released, that time value is saved into the integer 'stop'. From the difference of 'stop' and 'start', a 'duration' is calculated. A function is also added into the Tick event that can split up and end the capturing in intervals and stop it when the maximum time has been reached. It works so that when a key assigned to a behavior is pressed it checks if the behavior has been used earlier in the current interval and if not, assigns the start time as latency for this interval.
2.4.3 February 2010

Developing and testing went on and in the middle of February, I found a severe bug regarding the timer, that its accuracy is only good for values down to approximately 55 milliseconds.

The main reason for this is that each tick takes time, and if a tick was not finished when the next one fires off, then stalling of the ticks occurred and they formed a queue, which means that the integer responsible for keeping track of the increasing time was not correct.

To solve this problem a more precise and well timed class was needed. Something that could give accurate time readings down to the millisecond, or even better, but still give the option to perform tasks on a regular interval with no queuing or stalling.

After reading on the MSDN website about timers and timing, a class were found. A stopwatch-class. Ideal and with very high precision since it counts the time passed based on a high performance timer in the computer. It returns time in various formats, or the amount of ticks the underlying mechanism has been ticking since it started counting time. The most basic time returned is the property ElapsedMilliseconds. It is a long integer, a 64 bit integer, with the amount of milliseconds the stopwatch has been running. Its precision is down to the milliseconds, and the value is rounded down to the nearest whole millisecond value (MSDN Stopwatch Elapsed Milliseconds).

More complex time can be fetched with the help of the TimeSpan class. A instance of that class is returned when you call the Stopwatch property Elapsed. This class can return total milliseconds, seconds, hours days. It does not return months and years, since the amount of days varies in the months. For example, if we assign a TimeSpan the values two hours, nine minutes, twelve seconds and four milliseconds. Calling the TotalSeconds property of the TimeSpan would give 7752,0004 seconds as the representation is done using the decimal based structure Double. Calling the property Seconds on the same TimeSpan would only give a value of twelve, as it just returns the seconds part as a integer. The chosen property for time representation was the ElapsedMilliseconds. Simple implementation and stable and good performance were the idea behind that decision. This way the old method of a increasing integer in the timers tick function could easily be replaced with long integers instead, incread by the Stopwatch itself. Calculations and ToString() methods would be the same. This introduced the program to its first real severe bug hazard, as the implementation of drawing the rectangles to a timeline is based on regular integers of 32 bits length. Therefore code had to be written which converted the longer integers to regular integers, when it was possible, so the timeline could still be created.

But one thing was still missing, the tick ability, to fire events regularly. It is true that it was not needed now for increasing the time, but we still needed something to take care of the creation of the new intervals as well as when the time is up for the recording.

One implementation would be to check that when a key is pressed, and then decide if it is time for a new interval or not, or if the time is up. Problem here is that it is hard to alert the users to changes before the user interacts for it, for example if the time is up the user would not be alerted unless a key is pressed. Problem can also arise if two keys are pressed simultaneous, both key presses tells that its time for a new interval. This gives that both of them will each create a new interval, and not just one new. So this solution was not usable.
Another way to solve the problem would be to use a timer just for that purpose, to raise events on regular intervals. Or check for a condition and if true, fire the event. The problem with this is the same problem the timer previous had, lack of accurate timing and a high chance of errors by stalling.

The ideal implementation would be to continuously check the time elapsed on the stopwatch and then trigger events when enough time passed. The way to do this would be to encapsulate these readings in a loop, preferably a while-loop that runs as long as the stopwatch is measuring time, and while it is running always controls if the time has reached its max value, or if it is time for a new interval. But this solution rendered a big BUT. While-loops are exclusive for the thread. While it is running nothing else will get access to the machines resources, like memory or the CPU, in the execution thread. Not even the graphical interface will get computation time, which would give the user the illusion that the program stopped responding. So there were big problems with that implementation.

Or can the problems be avoided? If it were possible to split the main part of the program from the time and resource consuming while loop, then functionality requested would be achieved. The keyword for the while-loops problem is Thread, or rather threading. Is it possible to execute more then one thread on a computer? Of course, that is one of the basic parts of modern computers. Is it possible to make one program access more threads? Simple answer, yes. The namespace System.Threading states that the reason for its existence is to give programmers the ability to execute several tasks simultaneous (MSDN Threading).

The namespace System.Threading is a really competent solution, but a bit too much to solve this problem. With another separate thread comes big demands on the code as well as bugs when dealing with asynchronous operations. For example, it is required to let the GUI (Graphical User Interface) thread do updates to just the GUI. Other threads have to send requests to the main GUI thread through the forms Invoke method. In Mono, when a program is based on the System.Windows.Forms to create an interface to the user, all controls added need to be done so by the same, main, GUI thread (Mono Winforms).

Even some structures like the Int64 are not regarded safe for multithreading. On 32 bit processors the 64 bit long integer is not an atomic operation, meaning that it is not stored as one. Therefore one part can accidentally be edited by one thread while another thread reads the value, which gives an unpredictable result (MSDN Int64).

Since System.Threading is too complex for this task, a search for a better suited component began. Googling the subject and reading on the MSDN website came up with the solution. A class called Backgroundworker, with a very fitting name. The task to perform is assigned to the event DoWork. From there, once it is started it runs until finished, cleaning up the garbage on its way out, and all that without interfering on the GUI thread (MSDN Invoke). It is also possible to stop it before the computation is complete, but that requires adding code in the DoWork event itself for it to break. A backgroundworker cannot be canceled if it is busy calculating, so one must either wait for it to finish or add a controller in the function that checks if the worker is asked to abort. Waiting for the while-loop to finish is not the best way to do it in this program, mainly because it can run for several minutes before it is finished. Therefore a controller was added, which checks several times per minute if the worker is suppose to cancel. This makes it possible to end the recording before the time runs out if something happens, like a glitch in the movie or if the observer noticed that a behavior was
recorded by mistake. Extra threads can often keep a program alive, even after the user presses the quit button. This can occur when there is a high priority thread with an ongoing task. These threads are referred to as Foreground threads. The user interface is an example of that kind of thread. If more than one of those exists, precautions have to be made to make sure those are dealt with during application shutdown. The backgroundworker on the other hand uses a Background thread, which doesn't render the same errors. When all foreground threads are shutdown, the background threads will shutdown as well (MSDN F&B Threads). This is a intended behavior and will be used in this program.

The DoWork event for the backgroundworker is set up so that it starts with turning on the stopwatch, then it runs the checks of the time passed to create new intervals, and lastly, end the recording when the time is up. Both actions use the invoke method as changing intervals and ending the recording will trigger changes in the user interface. The while-loops condition to continue running is that the stopwatch is running, so stopping the stopwatch will effectively stop the loop as well.

2.4.4 March 2010

Development in March was mainly about fixing bugs and polishing the user interface. Some changes are worth noting down though. The biggest was be the ability to export a excel spreadsheet. The base of the spreadsheet is from the planning phase, when a template was constructed. This template is later modified and data from the event recordings are added to it. As there is no tool built into the c# library to export data sets to excel, an external part was used. This part is the GemBox Spreadsheet library. The main reason for this choice was that they offered a free version of their software, that could be used with full functionality, except for two limitations:

- Maximum number of rows per sheet is 150.
- Maximum number of sheets per workbook is 5.

This was within the limits we planned to output data in. The library is well written with full support for both old and new excel file formats, as well as support for the OpenOffice spreadsheet file format. A great thing with it was that it had built in support for the mono-framework, and therefore its full functionality could be used (GemBox).

Another function added was the ability to remove a specific behavior, without removing or rearranging the other existing behaviors. This was done using the built in function in the parent container of a control, with which a control can be removed based on itself. Apart from that other things had to be removed, like the key capturing events and the existence of that specific behavior from the list of behaviors.

On a user interview about added functionality to the program, a request was made for a way to pause the recording. There are many reasons for this useful function, such as if the video was in two parts, or that the observer needs to switch attention to something else for a while. Theoretically it is quite easy to implement. It is just to stop the stopwatch and then start it again when the recording is resumed. Practically the implementation required some change to
existing code as the background worker would stop when the stopwatch was stopped, thinking that either the time was up or that observer canceled the recording. So the worker was changed, a global Boolean was added to keep track of an ongoing recording, and the background worker kept track on the Boolean instead of the stopwatch.

2.4.5 April 2010

Still a key functionality was missing from the program, one that definitely was needed for the future users of the program. That was a way to save and open old project files, so a study could be continued. The name of the study, the observer, the amount of intervals and the maximum time were all needed to be stored. That is the settings of the study. Also, the behaviors added and which animal identities that had been used had to be saved. A behavior consists basically of just the name and which key that it is assigned to. The animal identity needed to be saved because if the observer tries to change the identity to a already existing one, a warning should appear telling that an existing animal might be overwritten if the observer chooses to continue. The behaviors and animal identities were saved in separate lists as they there is no real limit of the amount of behaviors or animals used in a study.

The best way to store data, to make it manageable and readable to both computers and machines is to save it as XML tagged data. It is defined so that you can customize the tags used to make it match your needs. Only a few limitations exist, and the main one is that you have to close every tag you open (XML).

Instead of creating the XML file by myself, the .Net-framework provides a nice built in class which serves as a transformer of user specified classes to well formed XML data. This class is called XmlSerializer and provides methods to both save and open instances of classes to and from XML (MSDN XML-serializer). To utilize this a new class was constructed which contains an instance of the whole configuration from creating a study, as well as the lists of behaviors and used animal id's. The outcome is a neat, usable and editable XML file which the user can edit if something needs to be changed on the study. It is good for changing the observer or which files to autosave. An example of a configuration file can be seen in Appendix 1.

During extensive testing in the end of April, a bug was found regarding the stopping and restarting a recording. In the first implementations of the background worker, an implementation was made which made the thread that the background worker was using to sleep in between the intervals. This did not matter for shorter studies as the intervals were short enough to snap out of the sleep before the recording restarted. But on longer studies, these sleeping periods made it impossible to restart recording as the background worker was still busy (sleeping) with the old recording. The best way to solve this would be to have the ability to awake the thread if it were sleeping when a recording ends. No solution that is simple or easy to implement exists, so a change in the original code was needed. So instead of having the thread sleep for almost the entire interval, it was changed into half sleeping a lot for almost the entire interval. These half sleeping sessions are only 0.5-1 seconds long, and before each sleep it checks if the recording is still active or if the user canceled the recording. If that is the case the background worker cancels itself and sends a message to the main thread that it is ready to start again.
2.4.6 May 2010

In the beginning of May, the first real prototype was sent out for field testing. Before that, some minor tweaks were done and the user interface code cleared up to make it smoother in rendering.

After the first feedback from the prototype testing, lots of work was done in regard of error control of user input and the processing of saved studies. It hadn't been made clear enough for the user which input was expected where and what was allowed as input.

But one problem was still present in the program. The use of the third party library GemBox to create the excel files still limited the possibilities of the program. But finding another good library was hard, and most of the available ones cost a great deal of money. After searching I found a library written in Java called POI, released under the Apache license which makes it viable to use even in larger and commercial applications. It provides the same functionality as GemBox with compatibility with both the old excel file type XLS and the new open source standard file type XLSX. It is possible to use a java library and call it with the .Net-framework but while reading about POI, I found that an port to C# had been made, called NPOI. This port was exactly what was needed, all functionality with creating spreadsheets as well as open source and adjustable if any customization was needed (NPOI). The Mono-framework provides a tool called MoMa, Mono Migration Analyzer. With its help, is it possible to analyze existing code, libraries or programs to see if it is possible to run it natively with Mono. If not, MoMa will give a detailed view over which features that are used which the Mono-framework currently lacks, and how to change it. Running it on the 1.2.1 release of NPOI points out certain errors and parts that need to be adjusted to be able to be used on the Mono-framework.

After the NPOI libraries had been adjusted it was time to implement them in the program. Thanks to the big similarities between GemBox and NPOI, the implementation was really easy and most of the code could be reused.

The program has now no external limitations regarding the recording, it is handling errors in the input and exporting data as expected. Therefore the program is now tagged with the release tag and received the version numbering 1.0 and is ready to be delivered to the customers.
3 Closing

This part will present the results of the work and to do so the finished program will be presented. There will be a discussion on how the development went and if it progressed as planned.

3.1 Result

Presentation of the program: how it would be used by the users with informative example screenshots from an example of a recording session with the program. This example session is recorded in Windows 7 with the newest .Net-framework, version 4.0. The overall look is similar on Os X using the Mono-framework, but there are differences in the graphical output to the user. This does not interfere with the outcome or general usability and therefore only screenshots from Windows 7 are shown here.

3.1.1 Starting up

The startup look of the program (Illustration 7) is simple, just the menu with an empty body. The reason I made it this way was because I wanted a slim program that didn't take up unnecessary space on the desktop because the observer should focus on the recording instead of this program.

Illustration 7: Start view of the program

Clicking on the File menu gives three options for the user. The Study menu contains actions for the study itself, and the commands there are only active after a study and recording have been made. Help contains an about dialog box and quick help files.

Illustration 8: File menu
The user is here (Illustration 8) given three available choices, New Study, Open or Close. If the user wants to continue on an old study, clicking on Open brings up an open file dialog with which the user can navigate to an old study and load it into the program. Close shuts the program down and New Study opens up a dialog where it is possible to set up a new study.

![Create Study window](image)

Illustration 9: Create a study window

This window (Illustration 9) provides input for a new study. Projectname is the name of this new study, in this case “Example”. The Observer (“Me”) is the person observing the recordings. Animal ID is the first animal to record, here specified as “test_1”. Length is the amount of minutes the recording will go on, ten in this example. Only whole minutes are allowed as input. Intervals is the amount of intervals in the recording. The smallest value is one, as the recording will be over at least one interval. Here four intervals are chosen. Projectpath is where the study will be saved. Defaults are in a folder called Recorded_data in the same folder as the application is located. To change this, simply press the button Browse and chose a new folder in the dialog window that pops up. It is also possible to check any of the checkboxes in the area Files to autosave to choose which files to autosave when a recording is complete, here an Excel file will be saved. The user can choose not to check any of these as it is possible to save this via the menu Study after a recording is complete. Pressing the Create button will create a study with the values chosen and save it in a folder named as the name of the study in the folder specified as Projectpath. An example of this settings file can be seen in appendix 1. Then, it will close the Create Study window and return the user to the main window.
As seen in Illustration 10 above some additions have been made to the user interface when a study is specified. The name of the study is shown in the status bar of the program and in the menu bar the current animal is shown. Labels are shown specifying the content of the different textboxes for the upcoming behaviors. The button with the red circle is the button to start the recording and the black square is the stop button for the recording.

From the study menu is it possible to add behaviors to the study, by the menu item *Add behavior* (see Illustration 11). Pressing it will show a new windows window where the information for a new behavior can be inputted.
In the textbox after *Behavior*: will the name of this new behavior be inputted. Under it is an area intended to define which button to use. The user activates it by clicking or tabbing to it and then press a key to assign it to this behavior (Illustration 12).

![Illustration 13: Completed behavior info window](image)

This behavior (Illustration 13) has the name *behavior_1* and the assigned key *A*. Pressing redo will clear the name as well as the bonded key, allowing the user to redo the Behavior info if something is wrong. Pressing OK will close the window and return the user to the main screen, with the added behavior.

![Illustration 14: Main window with one behavior](image)

Here (Illustration 14) the new behavior appears and the size of the main window is increased to make room for it. The information shown to the user is from left to right: A box showing the assigned key, the name of the behavior, the last duration of the behavior in this recording. Frequency is the amount of times the behavior has been expressed, Total Time is the total of all durations and Latency is how long time it took for this behavior to be expressed in the current interval. Pressing the white X in the red square will remove that specific behavior. Every added behavior will trigger a save to the settings file while removing a behavior will not. So if a behavior is accidentally removed, either load the settings file or add the behavior again.
3.1.2 Recording

Continuing from the starting up part, but now with three specified behaviors, to how to start and manage a recording session.

Illustration 15: Main window with three behaviors, how to start

There are two ways to start a recording, either with the record button on the main window or with the *Start recording* option in the Study menu (see Illustration 15). Both will show information about how the recording works, and how it is started.

Illustration 16: Start of recording information
This popup (Illustration 16) states to the user that the recording won't be started until a key on the keyboard is pressed. Pressing a key assigned to a behavior will trigger the recording to start with the behavior activated, giving a latency of near zero. Pressing a key which is not assigned will just start the recording and the behaviors might be activated later on. No menu items related with the study will be accessible after the popup has been shown. Pressing the stop button or the ESC key from here will stop the recording before its defined stoptime.

![Illustration 17: Main window, first behavior recording](image)

This recording (Illustration 17) has been started by pressing the A key given to the first behavior, behavior_1, a latency of zero seconds. The background of the current active behavior(s) will shift into yellow, to point out that it is recording. The label with the green background color on the left side of the window is stating that the recording has started. And on the bottom of the screen a clock is shown with the time passed in the recording. The recording button has now switched to a pause button.
When pausing a recording, the time will stop and any current recording behaviors will stand by, see Illustration 18. The pause button will change to a record button again, and pressing it will resume the recording. The label on the left has now shifted its background color to red and the text Timer Paused are shown.

While recording, press any assign key to start recording that specific behavior. More than one behavior can be recorded at the same time. In the above picture (Illustration 19) behavior_1 stopped recording at 14,239 seconds and behavior_2 started at 14,217 seconds into the interval. The time missing or overlapping is usually from the delay of switching keys on the keyboard. Behavior_3 was started 51,829 seconds into the current interval.
As the recording has reached the defined maximum time of the current study (ten minutes), the timer will stop and a window (see Illustration 20) will alert the user that the recording is finished. If any file was chosen to be autosaved during the definition of the study, the status of that will be shown to the user here. If the recording was interrupted, for example if the user pressed the ESC key, it will be shown to the user and any recorded data will not be automatically saved. The user can choose to save the data from the Study menu, even though the recording did not reach its end time.

When a recording has been stopped or completed, several ways of exporting the data have been activated in the Study menu (see Illustration 21). Save Excel-file will save the whole
recorded study as a formatted Microsoft Excel file, each behavior summarized over each interval with latency, total duration and frequency. See appendix 2. for an example file. Save CSV-file is a way to export the recording as a simple formatted, all data included, text file. It saves every recorded behavior as a line of text, with each event defined as a start time, a stop time, the latency and which interval it were recorded in. See appendix 3. for a example file. The highlighted Review events will show a window where the recorded events will be represented on a timeline. See appendix 4 for a example file.

Illustration 22: Review events window

Illustration 22 shows here a wide picture which displays graphically how the recording went. On the left side of the y-axis the names of the behaviors are shown. The scale is 1/100th's of a second per pixel in the picture and the number shown below the x-axis are the second marks.

Illustration 23: Review events window, intersection
The reviewing of events is good to check for intersections of behaviors, where a behavior is stopped and the next one starts recording (See Illustration 23). It is good to see if an overlap or gap occurred and if so for how long time. Then it is possible to go back to the recording and check if it was something strange there or if the user itself made an keyboard error. The Save button saves a copy of the picture as a TIFF file and the Done button closes the window.

Back on the main window (Illustration 24) again it is possible to change the id of the animal by pressing New Animal in the File menu. This will retain the original settings for the study with the new id, allowing the user to continue recording the next animal.
Inputting the next animal's id is simple, write it in the textbox and press enter or OK (see Illustration 25). The new animal id is shown in the Current Animal box in the menu bar. The id is also saved in the list of used animal id's in the settings file and the settings file are automatically saved.

If the user types an animal id already used in the study a warning will appear (Illustration 26) giving the choice to use the already used id or type in a new one. Using an already used animal id might result in a loss of already recorded data.

### 3.2 Conclusion

Development of the program went well, and all user suggestions regarding what functions were most needed were implemented. Following the plan was not easy, as this was new ground. It was easy to find solutions and ways around problems, and Agile Developing was a good method to choose. Design and creation as a whole is hard to follow, it is easy to listen and dream but hard to follow it up and goals and steps had to be constantly adjusted. Having a target and testers for the program is a great thing, and being able to constantly query them was of great help as developing such a specific program is very demanding and a great insight in the area is needed, which is hard to get as a developer.

At the 20th of May, a presentation was held at the Department of Neuroscience at Uppsala University. The program was demonstrated and shown for the lab groups which are lead by Klas Kullander and Åsa Mackenzie. The program lived up to their expectations and got good critiques from the other attending team members.

The move to cross-platform support went surprisingly smoothly. The choice to keep the developing targeting the version 2.0 of the .Net-framework played a large part in that, as no major workarounds were needed.
3.3 Summary

Creating a efficient program in C# and making it run in Mac OS X is indeed doable. The support is there, it is good and provides a good insight as to how Mono works and what parts that can be used to the fullest. Developing a program is tricky, but with good planning and good support it reduces the stress to a minimum.

Writing about it on the other hand was more strenuous then first imagined. Better planning on that side could have helped a lot. Even still, this was a fun project and really challenging. Picking up the flow became easier and easier and the desired outcome was eventually reached.

Finding the information on how to do a program like this was one part challenging and one part piece of cake. Google is your friend and from there help is not far away. Some things cannot be found so easily, then a person to talk with or a book to go through is better.

So in the end, this was a great learning experience, the program fulfilled its intentions and desired functionality, and it became a rewarding project which probably will be easy to continue to build on in the future.
3.4 Bibliography


MonoDevelop; *MonoDevelop Main Site*, http://monodevelop.com/, 2010-05-18


Silverlight 2; *Silverlight 2 released*, http://weblogs.asp.net/scottgu/archive/2008/10/14/silverlight-2-released.aspx, 2010-05-21


Mono Compatibility; *Current Mono Compatibility*, http://mono-project.com/Compatibility, 2010-05-19


Mono Winforms; *FAQ: Winforms*, http://mono-project.com/FAQ:_Winforms, 2010-05-18


GemBox; *GemBox Spreadsheet*, http://www.gemboxsoftware.com/GBSpreadsheet.htm, 2010-05-17

XML; *Extensible Markup Language (XML)*, http://www.w3.org/XML/, 2010-05-28

<?xml version="1.0" encoding="utf-8"?>
<StudyConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema" Version="1.0.0.0">
  <Observer>Me</Observer>
  <AnimalID>test_1</AnimalID>
  <StudyName>example</StudyName>
  <ProjectPath>C:\Users\Erik\Documents\Visual Studio 2010\Projects\keytestproject - Kopia\keytestproject\bin\Release Recorded_data</ProjectPath>
  <Intervals>4</Intervals>
  <Length>10</Length>
  <asExcel>true</asExcel>
  <asCSV>false</asCSV>
  <asTimeline>false</asTimeline>
  <AllMice>test_2</AllMice>
  <AllMice>test_1</AllMice>
</Settings>
<AllBehaviours>
  <Name>behavior_1</Name>
  <Key>A</Key>
</AllBehaviours>
<AllBehaviours>
  <Name>behavior_2</Name>
  <Key>S</Key>
</AllBehaviours>
<AllBehaviours>
  <Name>behavior_3</Name>
  <Key>D</Key>
</AllBehaviours>
</StudyConfig>
example
Me
17-05-2010

Appendix 2 Example of Excel-file

<table>
<thead>
<tr>
<th>Interval</th>
<th>behavior_1</th>
<th>behavior_2</th>
<th>behavior_3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Duration (s)</td>
<td>Frequency</td>
<td>Latency (s)</td>
</tr>
<tr>
<td>Interval 1</td>
<td>14,24</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Interval 2</td>
<td>0</td>
<td>0</td>
<td>300001</td>
</tr>
<tr>
<td>Interval 3</td>
<td>0</td>
<td>0</td>
<td>450001</td>
</tr>
<tr>
<td>Interval 4</td>
<td>0</td>
<td>0</td>
<td>600001</td>
</tr>
<tr>
<td>Behavior</td>
<td>Start (s)</td>
<td>Stop (s)</td>
<td>Interval</td>
</tr>
<tr>
<td>--------------</td>
<td>-----------</td>
<td>----------</td>
<td>----------</td>
</tr>
<tr>
<td>behavior_1</td>
<td>0</td>
<td>14,24</td>
<td>1</td>
</tr>
<tr>
<td>behavior_2</td>
<td>14,22</td>
<td>51,81</td>
<td>1</td>
</tr>
<tr>
<td>behavior_3</td>
<td>51,83</td>
<td>150</td>
<td>1</td>
</tr>
<tr>
<td>behavior_3</td>
<td>150</td>
<td>300,01</td>
<td>2</td>
</tr>
<tr>
<td>behavior_3</td>
<td>300,01</td>
<td>450</td>
<td>3</td>
</tr>
<tr>
<td>behavior_3</td>
<td>450</td>
<td>600</td>
<td>4</td>
</tr>
</tbody>
</table>
Appendix 4 Example of timeline
To whom it may concern,

**Re: Letter of reference for Erik Karlsson**

Erik Karlsson has developed software to score mouse behavior and has presented the features of this software in an oral presentation to us (Unit of Developmental Biology, Department of Neuroscience, Uppsala University). We are pleased and impressed that Erik has undertaken this project since we find that his software will be very useful in several of our projects. Further, Eriks presentation was clear and concise, and he demonstrated the possibilities of the software in a readily understandable way. From my point of view, **Erik Karlsson** definitely deserves recognition for his efforts.

Please do not hesitate to contact me if you should have any further questions.

With best regards

Principal Investigator,
Unit head,
Developmental Genetics