Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelor Thesis

Truong Vinh Phan

Development of a custom application for the Tobii Eye Tracker

Fakultät Technik und Informatik
Studiendepartment Informatik

Truong Vinh Phan

Development of a custom application for the Tobii eye trackers.

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung
im Studiengang Angewandte Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer : Prof. Dr. Olaf Zukunft
Zweitgutachter : Prof. Dr. Stefan Sarstedt

Abgegeben am 24. August 2011

Truong Vinh Phan

Thema der Bachelor Thesis
Entwicklung einer Softwarekomponente für den Tobii Eye-Tracker
Stichworte
Software, Komponente, Customapplikation, Tobii, Eye Tracker, Usability, Tobii Studio™, Tobii SDK, Tobii Technology GmbH, Tobii API, Microsoft Visual Basic
Kurzzusammenfassung
Der Eyetracker ist einer der wichtigsten Komponenten in jedem Usability-Labore (der Eyetracker, der in dieser Bachelorarbeit referenziert wird, greift auf den Tobii X-Series Eyetrackers zurück). Die Eye-gaze Daten werden mit Hilfe der Tobii Studio™ Suite aus der Firma Tobii GmbH. verarbeitet und visualisiert. Das Tobii Studio™ spielt auch die Rolle eines Clients zur Kommunikation mit dem Tobii Eye Tracker Server. Diese Software kostet viel jedoch, hat auch manchmal überflüssige Funktionalitäten und erfüllt nicht immer die Anforderungen des Benutzers. Deshalb werden die so genannten „Customapplikationen" benötigt. Für diesen Zweck bietet Tobii GmbH. den Entwicklern und Benutzern einen SDK, der so-genannte Tobii SDK. Der Ziel dieser Bachelorarbeit ist es, eine Customapplikation für den Tobii X120 Eyetracker, die ein einfaches Interface hat und allgemeinen Tasks in jeder Usability-Studie erleichtern soll, zu entwerfen und mit Hilfe des Tobii SDK zu implementieren.

Truong Vinh Phan

Title of the paper
Development of a custom application for the Tobii Eye Tracker
Keywords
software, component, custom application, Tobii, Eye Tracker, usability, Tobii Studio™, Tobii Technology Inc., Tobii SDK, Tobii API, Microsoft Visual Basic
Abstract
The eye tracker is one of the core components in every usability labs (the eye tracker mentioned in this report is of the Tobii X-Series Eyetrackers). Until now the software that handles the processing and visualization of eye-gaze data is the Tobii Studio™ suite from Tobii Inc.. This software plays also the roll of a client to communicate with the Tobii Eye Tracker Server, which is the firmware that resides inside the hardware. This Tobii Studio™ suite, however, costs pretty much, has sometimes excessive functionalities and not always can fulfill all the different demands of end users. Therefore highly customized applications are necessary. For this purpose does Tobii Inc. provide application developers and end users with an SDK, which is called Tobii SDK. The aim of this thesis is to design a customized application that has a simple interface and simplifies common tasks that a basic user would normally do when participating in an usability study.

Contents

Chapter 1

Introduction

1.1 Motivation

Software developed in recent years has been devoting a large portion of the code to the user interface. It would thus seem reasonable to allocate a good amount of effort in software development projects to ensuring the usability of those user interfaces. Any object, product or system that will be used by humans is subjected to usability problems and should undergo some form of usability engineering. There are several different methods of usability testing that can be used for the development of user interfaces to any kind of interactive system, including most consumer electronics products. Today, almost every usability labs are equipped with at least a device called the eye tracker, since eye tracking is one of the most popular and essential ways to achieve various usability studies and is widely used in research on visual system, computer software, psychology, product design, etc . . .

Eye tracking is the process of measuring or determining either the point of gaze, i.e where the user is looking, or the motion of the eyes relative to the head. An eye-tracker is thus a device for measuring eye positions and eye movement. Eye-trackers are divided into several categories, the broadest of which uses some non-contact, optical approaches to measure eye motion. Light, typically infrared, is reflected from the eyes and sensed by a video camera or a special optical sensor. Eye-gaze data is then extracted and delivered to the eye tracking application.

Based on these eye-gaze data, the application then analyzes and forms visual representations, such as a map of eye movements or a heat map, of the data. The developer can gain valuable information and knowledges about the behaviors and the habits of the user and/or the problems/difficulties that a user faced when using a specific interface from these visual representations. The developer can then improve the usability of the user interface based on the gained knowledge and thus the value of the software to serve a wider range of users.

This process of usability studying and improving is rather effective since it is mostly based on the end-user's point of view and feedbacks.

The eye tracker that is equipped in the usability Lab of the Department of Computer Science at the Hamburg University of Applied Sciences is of the model X120, manufactured by Tobii Technology, Inc.[1] . The software that is required to interact with the eye tracker is also from the hardware manufacturer, the Tobii Studio™. It is the cost for the software, cost of updates and the lack/superfluousness of certain functionalities that lead to the need of custom applications.

The author's work on a custom application which is aimed mainly at the Tobii X120 Eye Tracker serves as the main focus of this report.

1.2 The Tobii Eye Tracker custom application overview

This custom application for the Tobii X120 Eye Tracker aims at providing software developers and usability testers with a simple-to-use interface and basic functionalities for tracking, analysing and visualizing eye-gaze data.

The application is a Microsoft Windows application and interacts with users through a GUI [2]. It is conceptually divided into these functional units:

- Frontend: is the GUI that exposes the application's functionalities and provides visualisation of eye-gaze data and display of results to the users. It is user-friendly in that the user is guided through a three-steps workflow. The user - i.e the test leader - first creates a test project, to which tests and test participants are added. The user then designs the tests in the project by adding stimuli to them. After the tests are all set up and ready, the recording phase can follow. Before the actual recording can take place, the eye gaze of the test participant is calibrated. The test participant is then shown another GUI for the calibration process. Calibration is performed by having the test participant simply look at a dot that moves across the screen for a a few seconds a few times. Afterwards the eye tracker handles all recordings automatically, which include keystrokes, mouse clicks, screen contents, etc ... After each recording, the user can have the result, i.e the eye movements of the test participant visualised for better analysis. Data can be visualised as, for example, a heat map.

[1] Tobii Technology AB. - http://www.tobii.com

[2] Graphical User Interface

- Backend: Behind the scene resides the Tobii™ API that acts as a client (TETClient) to communicate, i.e exchange messages, with the eye tracker server (TETServer). The server delivers gaze data periodically to the client. The client handles them including calibration process automatically and sends to the GUI for visualisation.

1.3 Objectives and scope of this report

The main objective of this work is to design and implement a custom application (the front-end and the back-end) for the Microsof Windows platform using Microsoft Visual Basic technology. The custom application's main purpose is to provide a mean to collect eye-gaze data and visualise them. This is the main task of the author of this report. This application supports firstly only the Webpage stimulus, but it can be expanded with more stimuli and functionalities later as well. The TETServer technology and implementation is beyond the scope of this report, but general information concerning it will be also provided for better understanding of the whole matter.

1.4 Structure of this report

This report is organized in six chapters. The first chapter is an introduction, which gives an overview on this report's purpose, motivation and scope.

Chapter 2 is about the technologies that act as references or are actually required to implement the custom application. We first take a look on the commercial, all-in-one software solution from the Tobii company itself, the Tobii Studio™, and after that the hardware, which is manufactured also by Tobii, the Tobii X120 Eye Tracker. We will then get to know about the core logic that powers most of the eye tracker's functionalities, the Tobii™ API, which is part of the Tobii™ SDK. Finally we shall be introduced to one of Microsoft's technologies for building Windows applications, also one of the platforms that Tobii™ SDK supports, the Microsoft Visual Basic platform.

In chapter 3 we talk about the requirements for the custom application being built. These include functional requirements and technical requirements. A prospect of the custom application's future expansion possibilities will also be given.

Within chapter 4 is the in-detail discussion on the application's design. A brief explanation on the author's decision for the concret implementation of the custom application will also be given. The second part of chapter 4 is the discussion on the process of building up the custom application based on the design in the earlier part of the chapter.

In Chapter 5 we discuss about application testing. This include functionality testings and usability testings on the application to ensure the custom application fulfils the requirements.

Chapter 6 is the conclusion of this report. A summary on the custom application (purpose, functionalities) as well as possible future work will be given.

At the end of the report is an appendix, a glossary, a list of all figures and abbreviations used in this report, as well as an index and CD with source codes and documentations of this work.

Chapter 2

Used Technologies

2.1 Tobii Studio™ at a glance

> „Tobii Studio™ now brings new unique benefits to your research and business by providing such features as support for Retrospective Think Aloud (RTA), integrated questionaries, automatic E-Prime scene generation and much more." (Technology, 2010)

Tobii Studio™, a commercial software product of the company Tobii Technology, Inc., is a comprehensive platform for recording and analysis of eye gaze data, facilitating interpretation of human behavior, consumer responses and physiology. It combines preparation of test procedures and advanced tools for visualization and analysis into a powerful tool, which offers easy eye tracking data processing for useful comparison, interpretation and presentation.

Tobii Studio™ features a large variety of stimuli and allows both multiple and different types of stimuli to be combined in one single recording. A deep and thorough view of behavior is achieved by intergrating eye gaze data with user videos, sound, keystrokes, mouse clicks, etc ... and other data sources in a single solution.

A broad rage of studies can be achieved using the eye tracker hardwares from Tobii Technology, Inc. in combination with the Tobii Studio™ software, ranging from usability testing and market research to psychological experiments and vision research, thus ideal for evaluating interactive media such as websites, software, e-mail campaigns, computer games as well as print and online advertising, TV commercials, etc .... With its comprehensive workflow and multiple tools, Tobii Studio™ enables test leaders to gather data from a large number of test participants cost-efficiently. Recorded data is then easily taken to a high level for in-depth

analysis to extract invaluable knowledge of individual user behavior and statistical results from a group of users.

Its large variety and complexity of features and tools, however, might lead to confusion for most novice users. Basic users will find at the first use of Tobii Studio™ that its user interface is not so intuitive and self-explained, therefore it takes time for the users to get acquainted with the software, how long depends on the users' needs, experience and knowledge in the field of usability.

The wide range of features available in the software makes them sometimes to be excessive (for most basic users). The Studio comes therefore in three editions: Basic, Professional and Enterprise. Most of the times all that a basic user needs is just about 1/3 of the functionalities that the Tobii Studio™ offers.

The Tobii Studio™, just like most other commercial softwares, costs (lots of money), not to mention the eye tracker hardware that is required to operate with the Studio software (the hardware costs even more). The users will receive after purchase an account with login information in order to download the software direct from Tobii website.

The basic workflow of the Tobii Studio™ consists of four parts, which are briefly described below:

- Test design
- Recording (including Calibration)
- Replay
- Statistical analysis and visualisation

2.1.1 Test design

In this first step, a new project is created to which tests and test participants are assigned. A single project may include multiple tests and test participants.

One or several stimuli are then added to each test. A stimulus can be for example an image, a website, or a movie, etc . . . . Different types of stimuli can be combined in one single test.

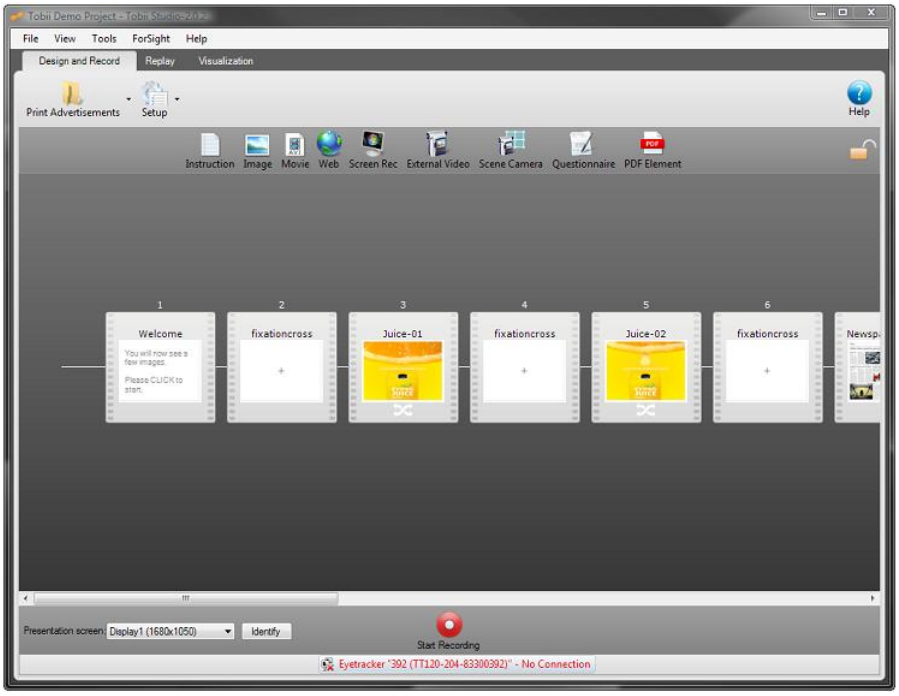


Figure 2.1: Tobii Studio™'s test design interface

For each test participant, a user profile which contains name and personal eye tracking calibration is saved. Independent variables such as age and gender can also be associated with the participant, providing more filtering possibilities for later analysis.

The Tobii™ Studio's user interface for test design is outlined in Figure 2.1.

2.1.2 Recording

Recording procedure begins when the user clicks on „Start Recording" button, then create or select one of the participants in the projects.

Before the recording is actually started, the eye gaze of the selected participant is calibrated. The stimuli is then shown on the monitor and the test participant will view it while the test leader can control the experiment and watch the eye gaze and user camera in real-time on a different monitor. Typical events (keystrokes, mouseclicks, screen contents, etc . . .) are automatically registered for later analysis. Events can also be explicitly logged by the test leader.

2.1.3 Replay

Right after the test, the recordings can be reviewed using the Replay tool. Logged events can be searched and new events can be logged. Segments of the recordings can be defined and video highlights can be exported.

2.1.4 Analysis and visualisation

With the analysis tool the test leader can visualise and analyse the eye movements of an individual recording or a group of recordings. Gaze data can be visualised as gaze plots, heat maps, clusters, animated visualisation or bee swarms. Specific area of interest (A.O.I) can be defined and data can be presented in a statistical tool. Results can then be exported into different kind of format.

2.2 Tobii™ X120 Eye Tracker

This is the target hardware that the custom application aims to interact with. The Tobii X120 Eye Tracker is a stand-alone eye tracking unit, which is designed for eye tracking studies relative to any surfaces. It is rather flexible, therefore ideal for visual testing of physical, real-world objects among many other things, which include TVs, monitors, projection screens...

The eye tracker's basic operating principle is to use infrared diodes to generate reflection patterns on the corneas of the user's eyes. Built-in image sensors will collect these patterns along with visual information about the person. It then utilizes sophisticated image processing algorithms to identify relevant features, such as eyes and corneal reflection patterns. Three-dimensional position of each eyeball and finally the gaze point (where the user is looking) are calculated using complex mathematics and delivered to client applications.

Tobii X120 Eye Tracker is a single, plug-and-play unit with no moving tracking component, thus allows large freedom of head movement for long studies without fatigue. It can track people regardless of age, ethnic, glasses or contact lenses. The hardware configuration can be done with little efforts and tracking is fully automatic. The eye tracking server (TETServer), which is the eye tracker's firmware, is already embedded in the eye tracker and requires TCP/IP connectivity in order to communicate with client applications over LAN.

2.2.1 Overall dimensions

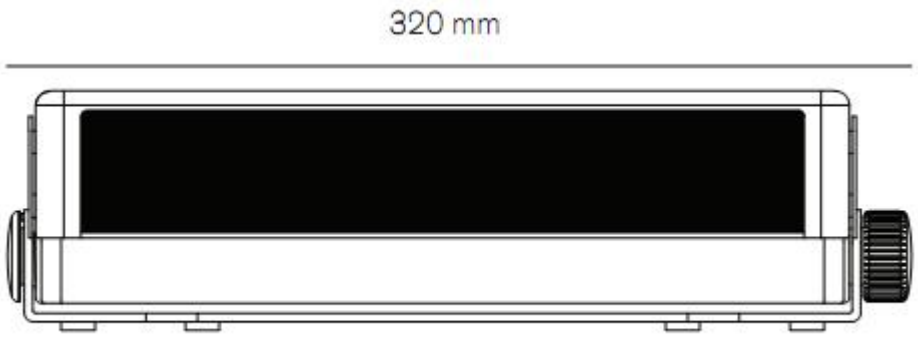


Figure 2.2: The length of the Tobii X120 Eye Tracker

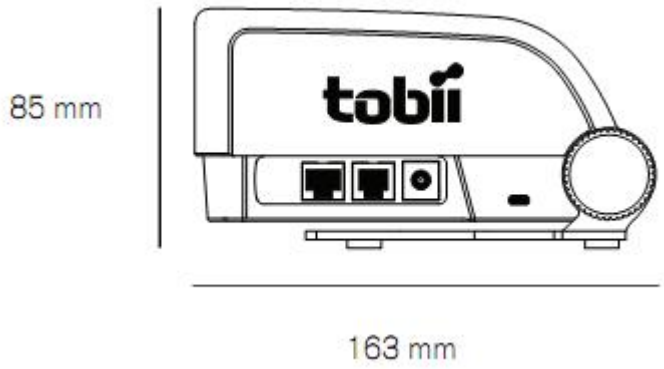


Figure 2.3: The width and height of the Tobii X120 Eye Tracker

Figure 2.4: Tobii X120 Eyetracker side-view

Figure 2.5: Tobii X120 Eyetracker front-view

2.2.2 Technical specification

Models	X120
Data rate	60 or 120Hz
Accuracy	typical 0.5 degrees
Drift	typical 0.1 degrees
Spatial resolution	typical 0.3 degrees
Head movement error	typical 0.2 degrees
Head movement box (width x height)	30 x 22 cm at 70 cm
Tracking distance	50 - 80 cm
Max. gaze angle	35 degrees
Top head-motion speed	25 cm/second
Latency	max. 33 ms
Blink tracking recovery	max. 8 ms
Time to tracking recovery	typical 300 ms
Weight (excluding case)	~ 3 kg (7 lbs)
Eye tracking technique	both bright and dark pupil tracking
Eye tracking server	embedded

Table 2.1: Tobii X120 Eye Tracker technical specification

2.3 Tobii™ SDK

The Tobii™ SDK, which contains the Tobii API, enables the development of application software for controlling and retrieving data from most Tobii eye trackers. These softwares are mostly highly customized applications as well as various interaction applications based on eye tracking. The Tobii API consists of the following sub APIs :

- TET API : Tobii EyeTracker API
- TETComp API : Tobii EyeTracker Components API
- TTime API : Tobii Time API
- TCVTrigger API : Tobii ClearView Trigger API

A brief overview for each of the APIs will be given below.

2.3.1 TETServer and Eyetracking firmware

The Tobii EyeTracker server (TETServer) is the component that handles all eye tracking and calibration calculations. For Tobii T and X series eye trackers (which include the X120), these calculations are done inside the embeded firmware, which acts as a kind of server. This means with the T/X-series eye trackers, no software installation is needed on the computer. For Tobii 50-series eye trackers, the TETServer must be installed as a software on the computer that is attached to the eye tracker hardware (it then runs as a service on that computer)

2.3.2 TET API

The TET API provides full access to the TETServer. Its interface provides a variety of function calls, including those to connect to the TETServer, start, stop, as well as calibrating it. The TET API does not enforce any GUI in order to allow full freedom of implementation. The user of the API therefore has to implement his own GUI if necessary.

The communication internally between the API and the TETServer is built on a proprietary protocol that works on TCP/IP communication layer. Thus any host computer running on Windows platforms (Windows 2000 or later) with a TCP/IP connection, i.e a network connection, to the TETServer can access it. To achieve programming language freedom, i.e the user can access the TET API using almost any programming language, it is provided as a standard Windows Dynamic Linked Library (DLL), the TET.dll.

2.3.3 TETComp API

The TETComp API provides COM and ActiveX objects that help to simplify and fasten application implementation process. Programming languages that consume COM objects include VB 6, VB.NET, C#, C++, and more. This API contains the following objects:

- TetClient : COM object used to simplify communication with the TETServer.
- TetCalibProc : ActiveX object which simplify calibration procedure.
- TetCalibProcWin : COM object providing a full-screen calibration window.
- TetCalibPlot : ActiveX object showing a graphical plot of the quality of the calibration.
- TetTrackStatus : ActiveX object showing the current tracking ability of the user.
- TetTimeManager : COM object used to communicate with the Tobii Time API (TTime API).

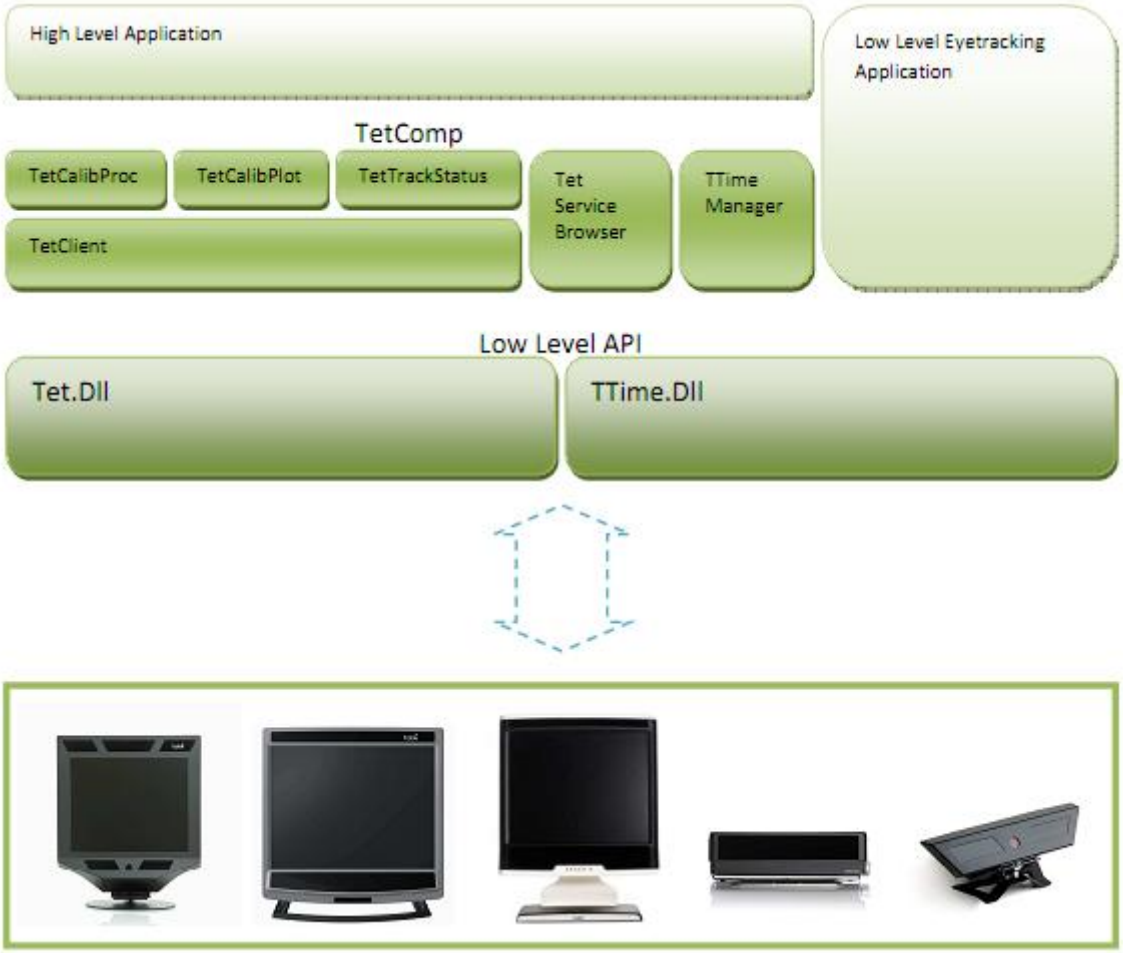


Figure 2.6: Tobii Eyetracking APIs

- TetServiceBrowser : COM object that automatically detects Tobii hardware on the network (currently only Tobii T/X series).

The component hierarchy is outlined in Figure 2.5.

2.3.4 TTime API

The TTime API is provided as a DLL, the TTime.dll. Its job is to provide high resolution synchronised common time for all applications interacting with Tobii Eyetrackers. If the TET-Server and the TTime.dll reside on different computers, each computer must have an instance of TTime.dll, which will automatically synchronize time with each other.

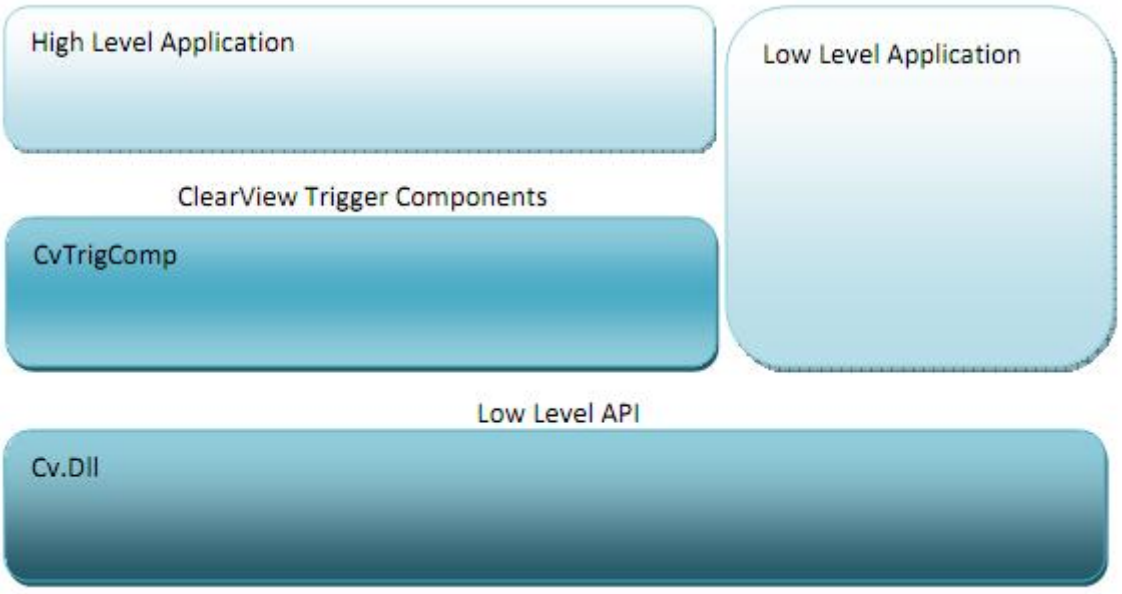


Figure 2.7: Tobii ClearView Trigger APIs

2.3.5 TCVTrigger API

This API is used to send triggering signals over TCP/IP to ClearView[3]. The triggering signals can be used to start and stop the recording or send time stamped data. This API is also provided as a DLL, the CV.dll, which can reside on the same host as ClearView or on another host with TCP/IP access.

The component hierarchy is outlined in Figure 2.6

2.4 Microsoft Visual Studio and Microsoft Visual Basic.NET

To have full freedom in developing algorithms and their fast implementation, and more important because this application is a GUI-oriented one: to have fast and powerful tools for the graphical output, especially the user-modules, Microsoft Visual Studio® 2010 development kit together with the programming language Visual Basic.NET was taken into consideration. It enables real-time support and fast data access due to its performance. It is also

[3]An older software solution from Tobii, replaced by the newer Tobii Studio™

the programming language that was used as an example in Tobii® SDK. Below is a quick introduction to the language.

Visual Basic (VB) is the third-generation event driven programming language and integrated development environment (IDE) from Microsoft for the COM programming model.

Visual Basic is derived from BASIC and enables things such as rapid application development of GUI applications, access to databases by using Data Access Objects, Remote Data Objects or ActiveX Data Objects and creation of ActiveX controls and objects.

Programs written in Visual Basic can also use the Windows API by declaring external functions.

The final release was version 6 in 1998 and the designated successor is called Visual Basic.NET, which is now known simply as Visual Basic.

Chapter 3

Software Requirements

This custom application aims at providing an alternative, simpler software solution that supports Tobii's hardware eyetrackers in usability studies. This part provides a look into the software's requirements specification. It includes use cases, functional and non-functional requirements.

3.1 Use cases

A use case in software engineering and systems engineering, is a description of steps or actions between a user (or "actor") and a software system which leads the user towards something useful.Bittner und Spence (2003)

Use cases are a software modeling technique that helps developers determine which features to implement and how to gracefully resolve errors.Adolph (2002)

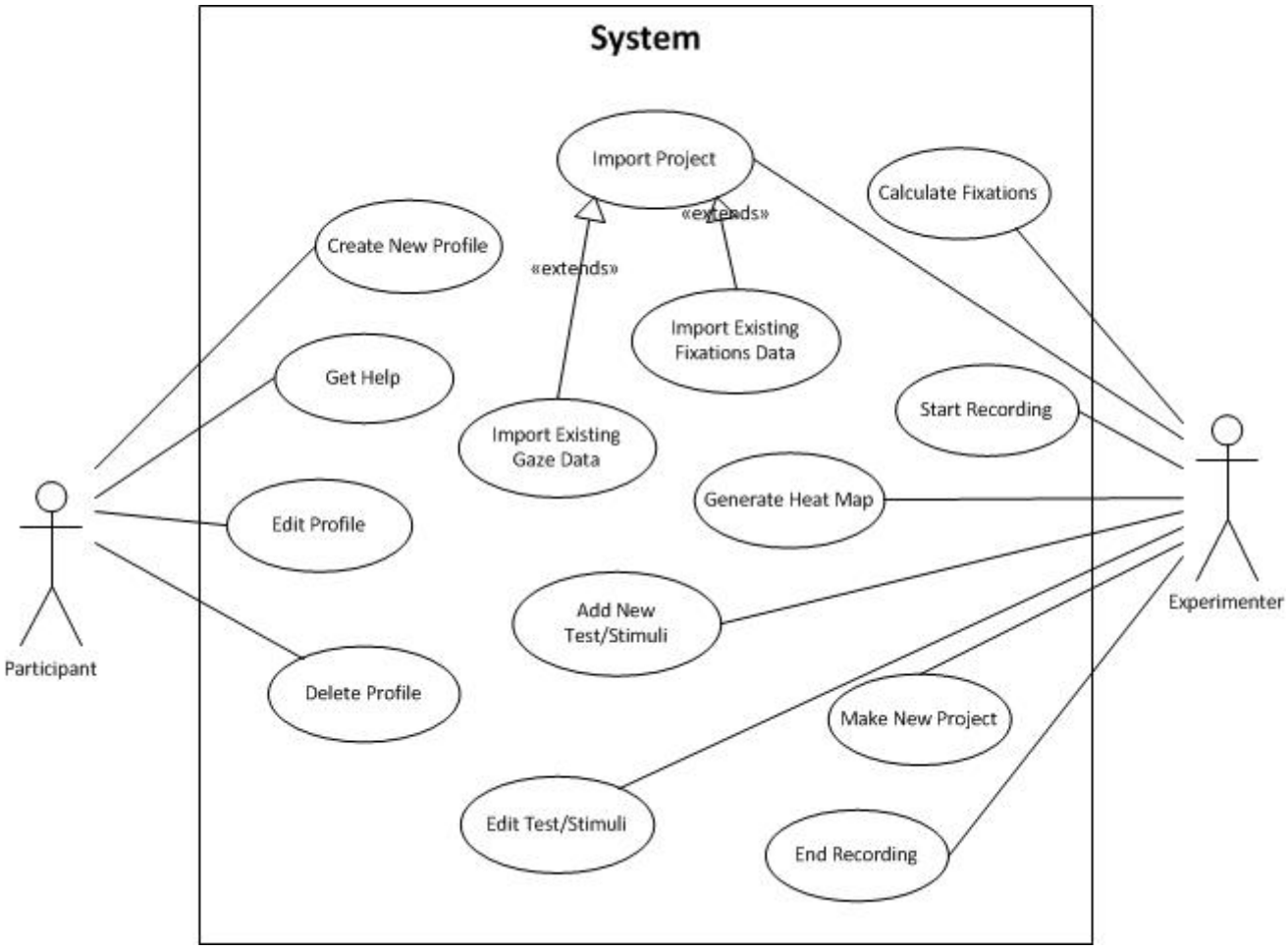


Figure 3.1: Use case diagram of the application

3.1.1 Use case „Design a test”

Actors	application user (experimenter)
Goal	a new project is created, a test with at least a web stimulus is added to it.
Summary	the experimenter can use the application to create a new project and use the TestDesign module to design and add tests to this new project.
Triggers	the experimenter decided to make a new project and start designing stimuli.
Pre-conditions	the types of stimulus that the experimenter wants to use and design are supported by the application.
Post-conditions	a new project is created, a test with the desired stimuli are added to it.
Basic course of events	1. the experimenter launches the application.
	2. the experimenter chooses to make a new project from the initial dialog.
	2. the experimenter configures the initial project settings following the project settings dialog.
	3. the experimenter clicks on the „Design a test” button from the start task dialog.
	4. the application launches the TestDesign module.
	5. the experimenter selects a desired stimulus type and configures it following the according stimulus setup dialog.
	6. the experimenter clicks on the „Save” button.
	7. the application adds the designed test with the stimulus to the new project and saves all the changes.
Alternative paths/Exceptions	n/a
Requirement	3.2.1

Table 3.1: Use case „Design a test”

3.1.2 Use case „Record and visualize data”

Actors	application user (experimenter), test subject
Goal	record a test and visualize the result (raw gaze data) as heat maps.
Summary	the user can use the application to record the designed test, which will deliver eye gaze data and save those data into a database , then visualize them as heat maps.
Triggers	the test subject is ready to record the designed test.
Pre-conditions	the test is properly designed and ready for recording, i.e. all stimuli are set up and there is at least a subject available to record the test.
Post-conditions	the test session is recorded, a calibration profile for the test subject is produced, a heat map bitmap of the recorded data is produced and shown to the user.
Basic course of events	1. the experimenter clicks on the „RecordModule” button to launch the module.
	2. the experimenter chooses a tracking device and configures it following the instructions on the setting dialog.
	3. the experimenter clicks on the „Connect” button.
	4. the experimenter chooses the test subject to be recorded by clicking on the „Subject” button.
	5. the experimenter clicks on the „Calibrate” button to start the calibration process for the chosen test subject.
	6. the experimenter clicks on the „Record” button to start the record session.
	7. after recording finished, the experimenter runs the FixationModule tool against the newly recorded data to detect and calculate fixations.
	8. the experimenter runs the HeatMapModule tool against the newly produced fixation data to generate heat maps.
Alternative paths/Exceptions	
Requirements	3.1.2.3

Table 3.2: Use case „Record and visualize data”

3.2 Functional requirements

3.2.1 Test design

The user can opt for already existing data, i.e. import data into into the application to be analyzed and visualized, or create a new project and then add tests to it. A test can have at least one or as many stimuli as the user wishes added to it. The application provides the user with a wide range of stimuli to choose from. The user will then have to configure each of the added stimuli in order for the test to be carried out properly as a whole.

3.2.1.1 Test subject

Test subject is the participant of the test, i.e. the person with whom the test should be carried out and therefore one of the key elements of the application. Each participant will have his/her own unique profile with personal data like name, age and gender. The application's workflow should have the test subject as one of the key elements and thus should associate the test subject's data with each of the functionality and actions it performs.

3.2.1.2 Stimulus

Stimuli are different types of media that can be visually delivered to testers. They are the objects to be tested and one of the key elements in every tracking and analyzing software. Commercial eye-tracking and analysis applications normally provide users with a wide variety of stimuli to choose from . For the scope of this report, the application should aim mainly at the „Web" stimulus that enables usability testing for web pages.

3.2.2 Calibration and recording

The application is able to interact with hardware trackers in order to receive tracking data. For these data to be accurate the application should firstly guide the user to go thourgh a calibration process.

3.2.2.1 Calibration

Every measuring devices need to be calibrated before they can properly function, so are eye trackers. Each participant will go through the calibration process before their gaze movement can be actually and properly tracked. The calibration data should be then saved and associated with the participant's profile for future reuse. The calibration tool is normally provided by the eye tracker hardware manufacturer. The application will prompt the user for calibration before each recording should no calibration profile found for the user.

3.2.2.2 Recording

The application will signal the eye tracker to start tracking the participant and start recording the screen contents itself. For the scope of this report, the application will only record web page contents, i.e screenshots and collect eye gaze data. The application should be able to receive data that the tracking hardware returns and stores them into a database for later analysis.

3.2.3 Visualization and analysis

The application can visualize the gaze data collected during the test by various means. Existing data can also be imported into the project for visualize and analysis. For the scope of this report, data will be visualized by heat maps[4] and fixations maps.

3.2.4 Saving and restoring work data

The application is able to save data that the user is currently working on into the project file, should there be any problem that may cause the application to crash or when the user finishes some editing tasks. The user is then able to restore the file that he/she was working on at a later time.

3.2.5 Import and export of data

The application is capable of importing existing data for analysis and visualization and exporting them once the user finished with his/her tasks.

3.3 Non-functional requirements

This is the non-functional requirements specification for the custom eye-tracking and visualizing application. It consists of software usability, efficiency, reliability and maintainability. It includes a set of use cases that should describe all the interactions the user will have with the software. These requirements are sometimes called supplementary[5] requirements and are after ISO-9126Wikipedia (2011a) international standard for software quality. Those are requirements which impose constraints on the design or implementation, for example engineering requirements, quality standards or design constraints.

[4] Also called attention maps.

[5] Non-functional requirements

3.3.1 Application usability

The user interface is key to application usability. The application should include content presentation, application navigation and user assistance. While a comprehensive discussion of effective user interface design is beyond the scope of this report, this section provides some guidelines on how the application's user interface should be look like.

The first thing to take into consideration is the targeted user groups. Users often have different usability requirements. This application aims for every users that have a need to perform usability studying through eye-tracking. This targeted user group, however, might be divided into two groups : a Basic group and an Advanced group. Therefore the application should come with a step-by-step installation procedure and ready-to-use[6] settings to help the Basic group to start using it right out-of-the-box without putting much efforts on setting up and configuring it. To satisfy the Advanced group with the need of fine-tuning the user experience, the application should include a control panel-like settings interface which let the user adjust all the details of the parameters that the software depends on to function.

The second thing to come is the content presentation, and with it comes also a graphical user interface. The application should enable users to control content presentation.

The application also assists users in using it properly by providing help documents that include instructions on setting up the hardware, using the functionality of the application and software documentations.

Next thing to consider is the user navigation. In order to minimizing user's effort on learning the interface thus maximizing user's productivity, the functionality of the application should be clearly separated. Graphical computer applications with a MDI[7] are those whose windows reside under a single parent window (usually except for modal windows), as opposed to all windows being separate from each other (single document interface). Such systems often allow child windows to embed other windows inside them as well, creating complex nested hierarchies.Wikipedia (2011b).

So each module should be on a child window that resides separately in a main parent window.

[6] Default settings

[7] Multiple-Document-Interface

3.3.2 Application reliability

The application's desired and undesired behaviours should be known in advance. The application should provide error handling mechanisms to be able to resist crashing, i.e. continue to work properly even in the situation when a bug or error occurs at various degree of seriousness.

The application should provide the user with enough documentation, help information and instruction to assist the user in using the program properly. A fool-proof mechanism should be available to ensure data integrity and stability.

Last but not least, users performing complex analyses often want to save their work at a certain point. The application should therefore provide users with the ability to save and restore work.

3.3.3 Application efficiency

The application should strive at reducing time and resource consume as much as possible, so that it might be able to run on low-resource and older machine. The application should also aim at delivering fast and accurate calculations and results.

The application should provide the user with only required functionality, according to the purpose of the program. Extra features may become superfluous and lead to confusion or distract the user from main functionality.

3.3.4 Application maintainability

The functionality and behaviours of the application should be easy for every developers to understand and restructure. Software builds also play an important role here. Many programming languages like C and Java require the source code to be translated into machine code through a tool called a complier. Additional operations may be involved to associate, bind, link or package files together in order to create a usable runtime configuration of the software application.

The software build is critical because if any of the generated files are incorrect or corrupted, all the software is likely to fail. Also if the incorrect version of the program is accidentally used, it may lead to false results if the software is to be tested.

Software deployment is also important because it involves technical parameters, which, if set incorrectly, may also prevent software testing from the beginning.

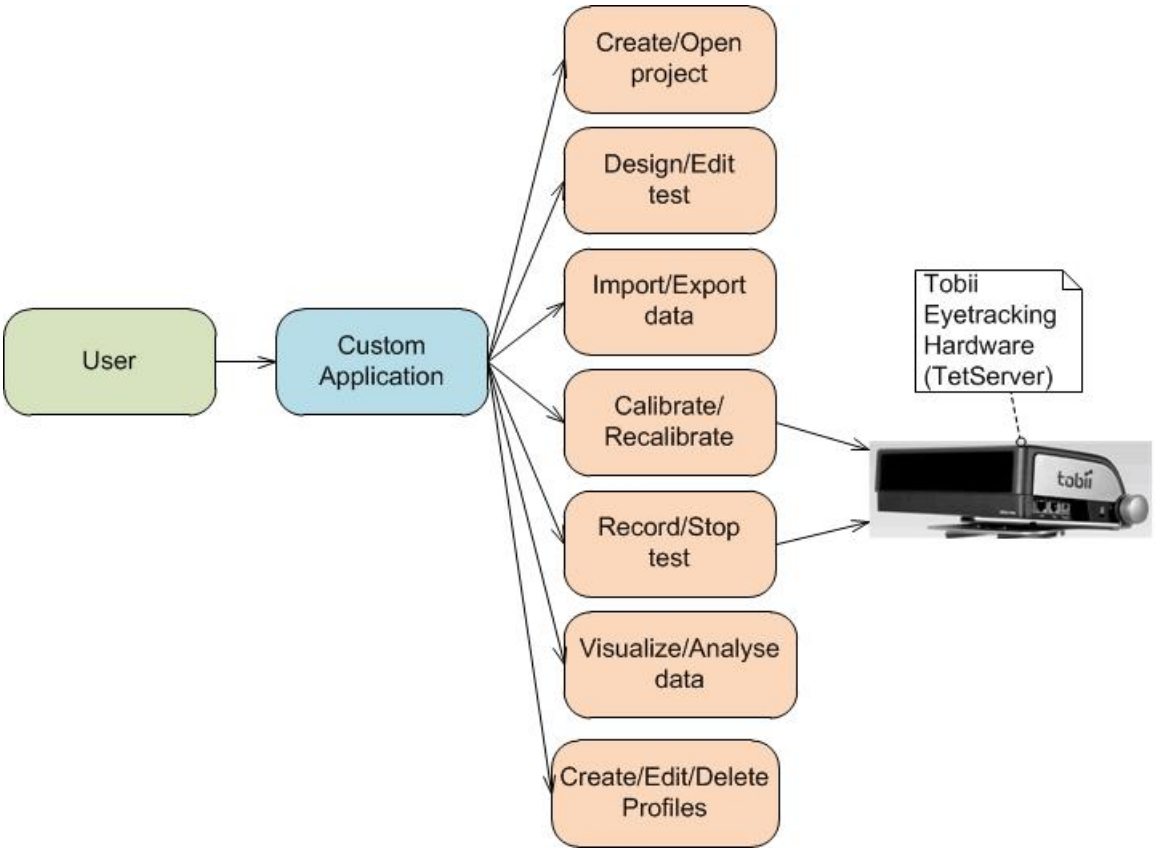


Figure 3.2: Base function diagram of the custom application

3.4 Base functions

The aforementioned requirements are mapped into the following base functions of the application, which are depicted in the diagram in Figure 3.2 below.

The user interacts directly with the application, which acts as a client to request and receive eye-gaze data[8] from the eye tracker, which is the TETServer, using proprietary protocol from Tobii. The communication is achieved by utilizing methods from the Eyetracker API.

The user can, via the application, create a new test, add media (stimulus) to it, then initiate the calibration process, or redo this process in case the last calibration was not good enough (recalibrate the bad points). The user can then start record the test, the application will fire up the browser with the pre-defined webpage so that the participant can start solving his tasks right away. The user can stop recording the test at any time. The eye-gaze data of

[8]More information about gaze data is shown in Appendix A at the end of the report.

the participant will then be visualized as heat maps/fixation maps, which serve the purpose of studying the usability of the webpage via user behaviors.

3.5 Prospects/Expansion possibilities

In this part we discuss on the matter how the application can be easily extended in the future. To answer this question, we should take several facts into consideration.

Firstly, the main target hardware for this application is the Tobii X120 Eyetracker. The back-end of the application, however, is built on Tobii API, which can be used to control a wide range of eye tracker hardware of different series from Tobii as well. The application is therefore extensible to other eye tracker hardware from Tobii as well.

Secondly, the applicatin currently supports „the Web" as its only media type (stimulus). It can be, however, expanded to support other types of stimulus as well, such as PDF, movie, text, image ...

Thirdly, the application can be extended with more options in visualization and analysis, which can range from a gaze plot, bee swarm map, to user video cam and flash contents. Also the software can be expanded to collect more kind of data, such as mouse movements, mouse clicks, sound, video, etc...

Fourthly, the application can be expanded with another module for the replay function. With this module, the entire recorded section will be replayed on a timeline and can also be exported as a movie (video format can be various, as long as the correspondent codec is available).

Fifthly, the application can be extended in such a way that additional participants can be added and also have their own profiles, also test project that can consist of several tests, and test that can consist of several stimuli are thinkable. We can achieve this by having some sort of database files coupled with the application.

Sixthly, one of the restrictions of the application is that it can target only the eye tracker hardwares from Tobii Technology, Inc., having the Tobii API as its back-end. Nevertheless, if we can somehow employ some sort of an standardized open-source eye gaze API instead of proprietary APIs like the Tobii API, then we can think of an application that targets all eye tracker hardwares from different manufacturers as long as they comply with the standard API. (Hennessey und Duchowski, 2010)

Chapter 4

Application Design and Implementation

4.1 Application design

This section specifies the design of the application, as well as the considerations and decisions made when designing the application in order to meet the requirements.

First of all, we need to consider about the objects and components which the application will require. There are a few criteria which we will take into consideration when designing the objects and components.

- Objects should be reusable.
- Clear distinction between different component's functionalities.
- Separation between static code and code that will be likely to change.
- The development process is normally carried out by a team of developers, but in this case it will be a one-man development process.

This section describes the design of this custom application for the Tobii eye trackers, which includes design decisions as well as the according architecture (functional architecture, technical architecture, component descriptions, database design).

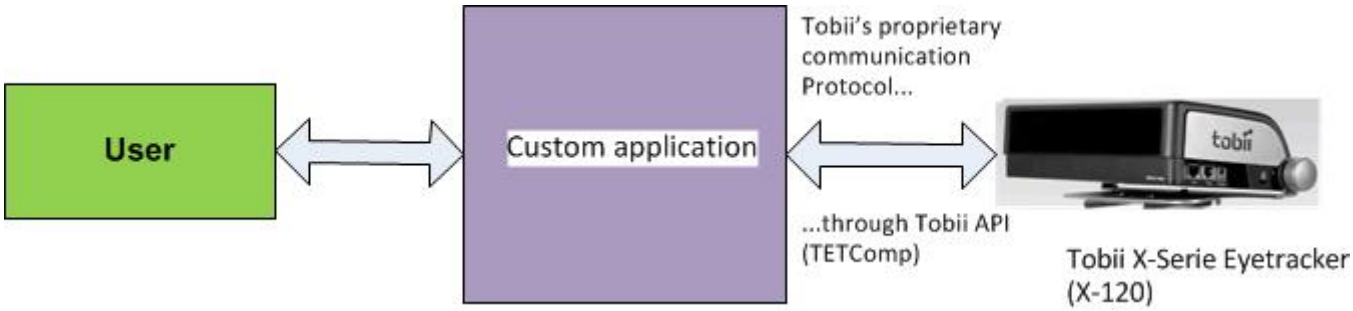


Figure 4.1: Abstract view of the application.

4.1.1 Some concepts

This part explains a few concepts that will be used throughout the application. An experiment can be thought of as a test project that can consist of one or multiple test(s). Each test represents a sequence of stimuli that are to be presented to the test participant. Those stimuli can be considered as the displayed slides in a slideshow. Each test also represents a recording unit. We define a test, which we shall call a „Trial", to be the default and smallest unit for display and analysis in the application. By default, each trial consists of only one single slide, but one can add multiple slides within one trial. Imagine a situation in which three web pages should be displayed one after another during analysis, but have moving areas of interest (AOIs). In that case one would create three web stimuli with the start URLs and concatenate them into one trial. During analysis, one can select a trial and then switch on the timeline between the different slides. The visualization output will be aggregated over the three stimuli contained in the trial.

4.1.2 Functional architecture

This core functionality of the application is separated into modules, according to the function of each module. There are six components: ImportExportModule, RecordModule, FixationsModule, HeatMapModule, DatabaseModule and TestDesignModule, as depicted in Figure 4.2. These components are categorized into two groups: Design and Record group, which contains the RecordModule and TestDesignModule, and Visualization and Analysis group, which contains the remaining components. They are members of the „Modules" namespace. The following is a brief description of what each component does:

The FixationsModule detects and calculates fixation and saccade patterns, based on the gaze data available in the database.

The HeatMapModule is designed to calculate and generate heat maps, which is a bitmap image computed from aggregated data and superimposed over the stimulus image, from the fixation patterns.

With the TestDesignModule, new trial and slides can be created, designed, existing slideshow can be modified and prepared for recording. Design activities include, amongst other things, adding and naming stimuli, naming the trial, configure and arrange the stimuli within a slide and slides within a trial, etc...

The RecordModule is used to interact with the hardware tracker and start a record session, which guides the user through a four-steps procedure from connecting to the hardware to actually collecting gaze data from the subject and writing data to the database once the session finished.

The DatabaseModule displays the content of the database in table format. With this component, the user can revise all data within the databse, ranging from test subjects to raw gaze data and modify them on demand.

The last component, ImportExportModule does nothing else than, as its name implies, to enable the user to import various kind of existing data into the application, ranging from existing project, slideshow with trials..., to raw gaze data for visualization and analysis, as well as to export the same kind of data back into files.

There is a package named „Common" which contains mostly custom types, forms, dialogs and controls, custom events, static classes and utility tools made ready to be used by other components.

The application is mostly GUI-based[9], and so are the components.

The component model of the functional architecture and the functional data model diagram is depicted in Figure 4.2 and Firgure 4.3, respectively.

[9]Graphical User Interface.

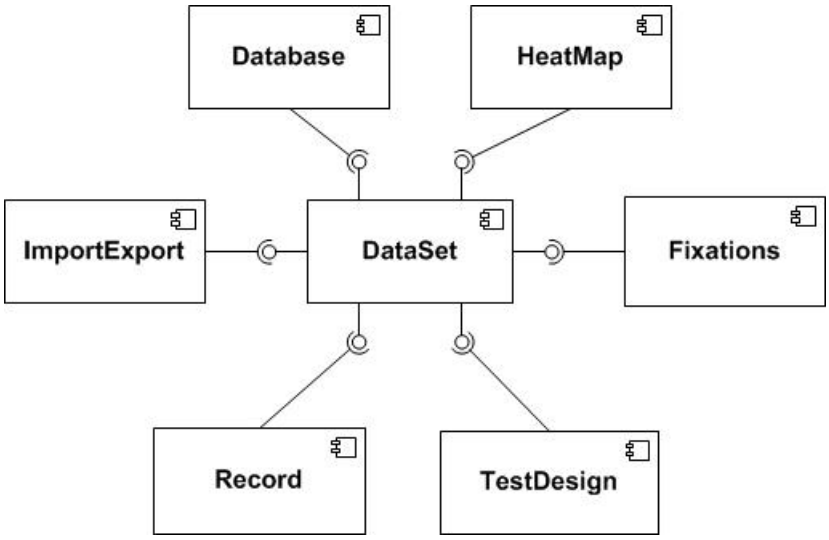


Figure 4.2: Component model of the functional architecture.

4.1.3 Technical architecture

The overall structure of the application consists of six namespaces, which are the followings: MainWindow, Properties, DataSet, ExceptionHandling, Help and Modules, as depicted in Figure 4.4 and Figure 4.5.

The MainWindow namespace contains the main GUI form and support classes. This main form acts as a MDI[10] parent, which hosts all the modules windows as children forms.

The Properties namespace contains data classes to store and retrieve all the setting parameters that are application-wide valid and a list of recent files.

The DataSet namespace contains the DataSet class and a DatabaseTemplate, whi to provide fast access and interact with the external SQL database through data adapters, because it represents an in-memory cache of data MSDN (2011).

The Help namespace contains help documents and explanations which assist the user in learning and using the application properly.

The ExceptionHandling namespace contains utility classes that handle all errors and exceptions that may possibly occur during runtime.

[10]Multiple Document Interface.

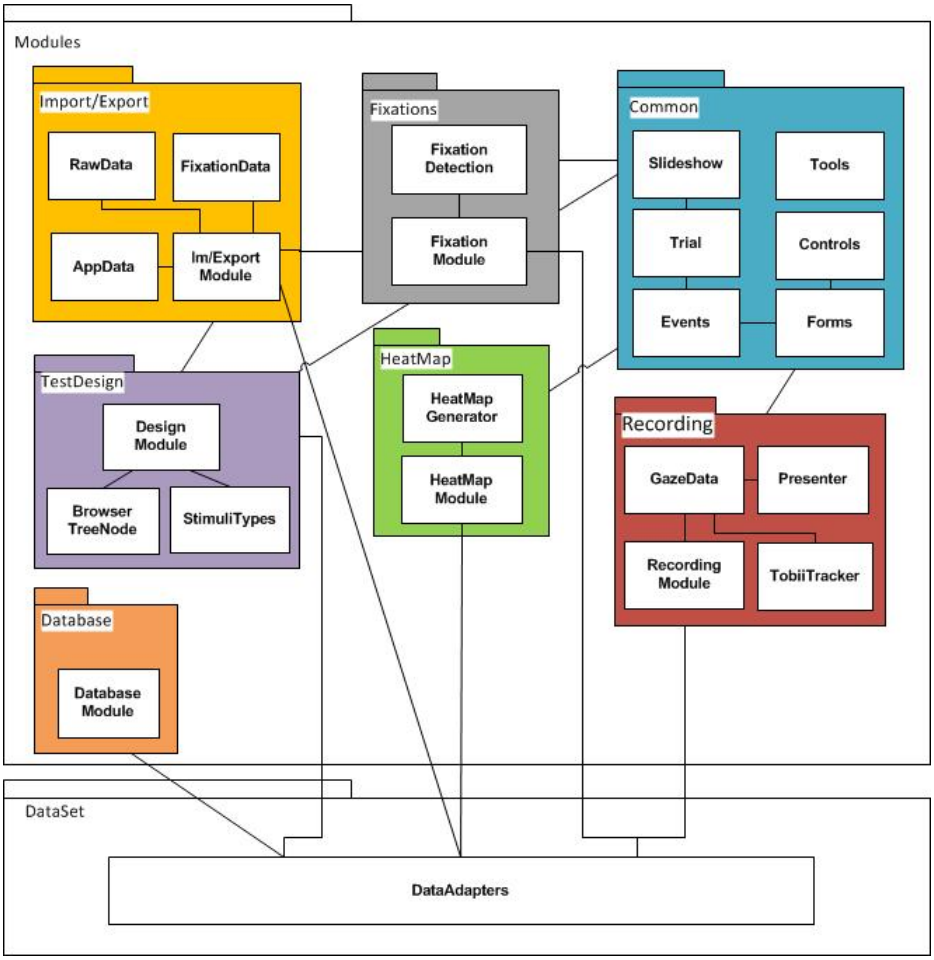


Figure 4.3: Functional architecture of the application.

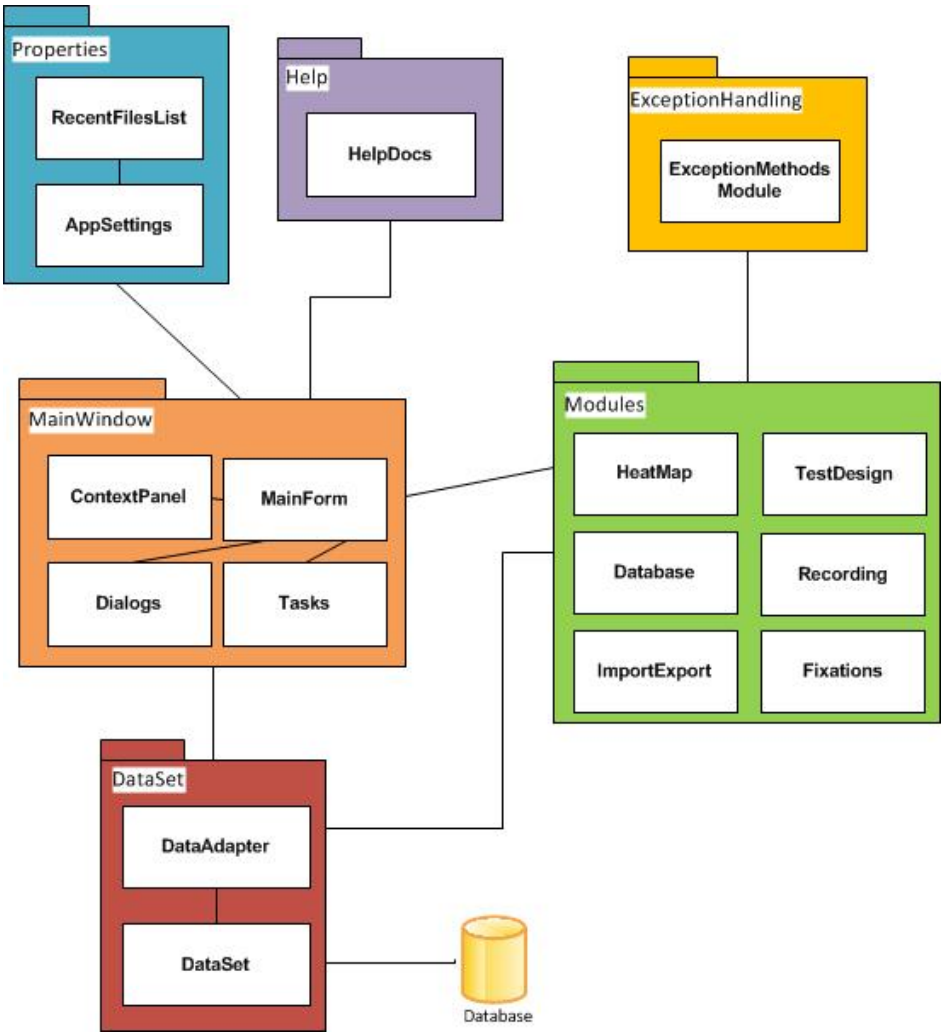


Figure 4.4: Technical architecture of the application.

Finally the Modules namespace contains the core functionalities of the application. Details on these modules will be discussed in the next section.

The following section describes each of the components in details,

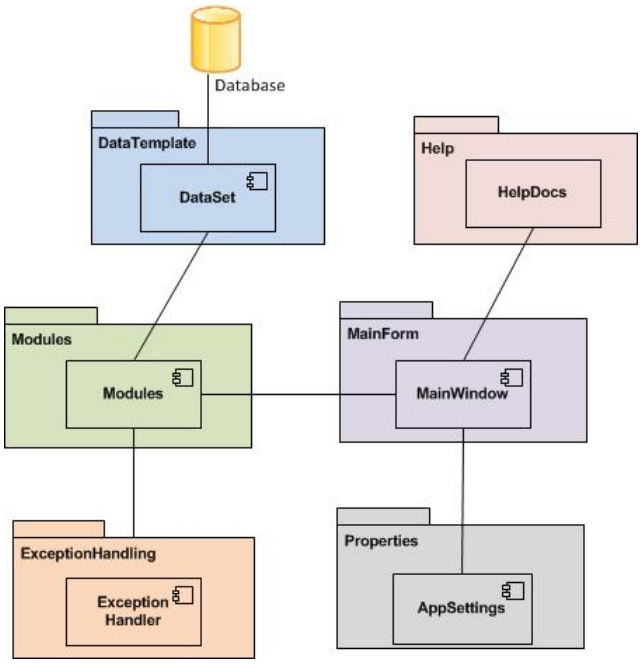


Figure 4.5: Component model of the technical architecture.

4.1.4 Component „TestDesignModule”

4.1.4.1 Task and responsibility of the component

The TestDesignModule enables the user to create and design new slideshow, adding stimuli, as well as importing existing slideshow and stimuli. This component has a GUI and aims to provide user with a WYSIWYG tool. This is one of the core components of the application and the first step in the workflow.

4.1.4.2 External view of the component

The TestDesignModule interfaces with the DataSet via DataAdapter, ImportExport component and enables the user to create and design (e.g add stimuli, name slides and trials) a new slideshow or import existing slideshows and trials to be revised and finally save the slideshow to the database.

4.1.4.3 Internal view of the component

The TestDesignComponentConfig class initializes the TestDesignManager and the use case class TestDesignMgmt. The TestDesignManager uses the entity classes StimuliTypes,

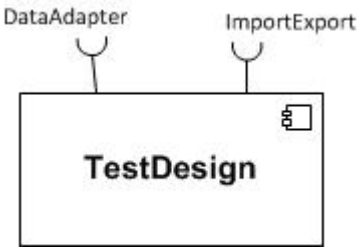


Figure 4.6: TestDesignModule, external view.

BrowserTreeNode to enable the Web stimulus, as well as Slideshow and Trial classes to represent a project with a sequence of tests internally.

4.1.5 Component „RecordModule”

4.1.5.1 Task and responsibility of the component

The RecordModule delivers mainly the ability of interacting with the hardware tracker, e.g. telling it to start or stop tracking, start the calibration procedure, as well as receiving gaze data returned by the eye tracker and storing them into database.

Another main task of this component is to present the ready-made slideshow with stimuli to the participant, only then the actual recording can begin.

4.1.5.2 External view of the component

The RecordModule uses the DataAdapter interface through the DataSet class to load the slideshow and trials from the database to be presented, and to write the recorded gaze data into the database. It is a GUI-based component.

4.1.5.3 Internal view of the component

The RecordingComponentConfig class initializes the recording module and the use-case class RecordingMgmt. The module uses the entity class TobiiSettings and TobiiTracker, which is backed by the library TetComp.dll by the Tobii API to be able to interact and communicate with the Tobii eye tracker. The GazeData class wraps the GazeData structure which is originally returned by the Tobii eye tracker and transform it into a format readable by the application.

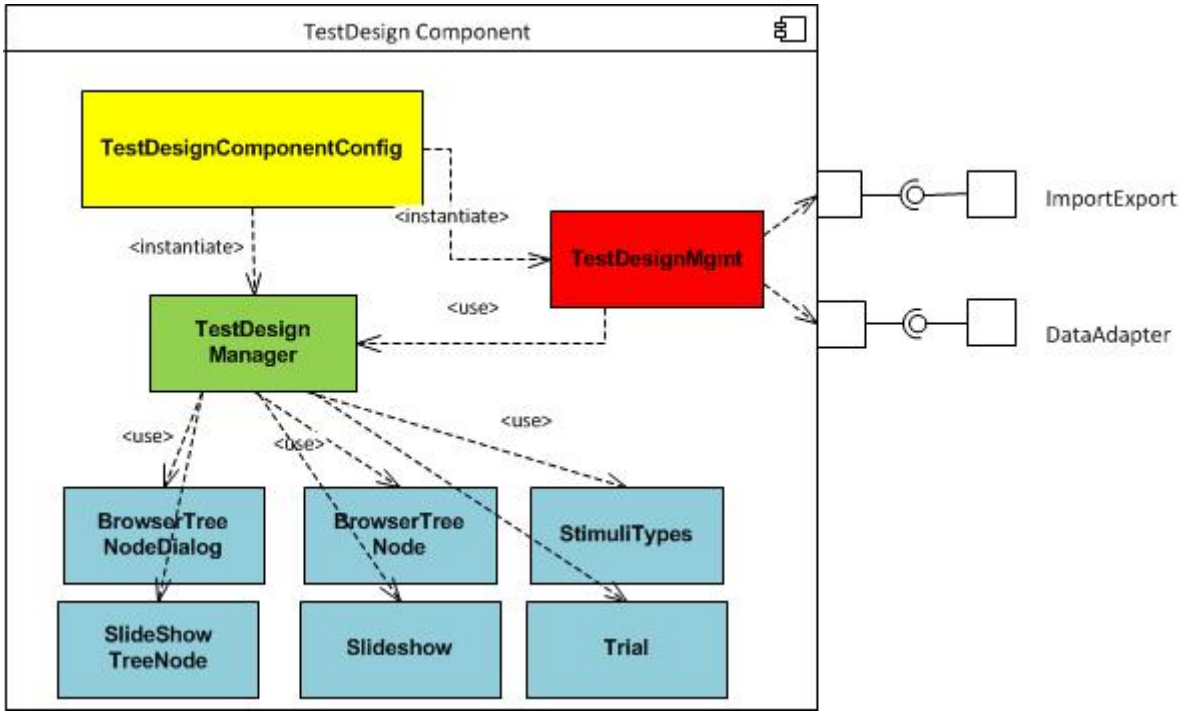


Figure 4.7: TestDesignModule, internal view.

Figure 4.8: RecordModule, external view.

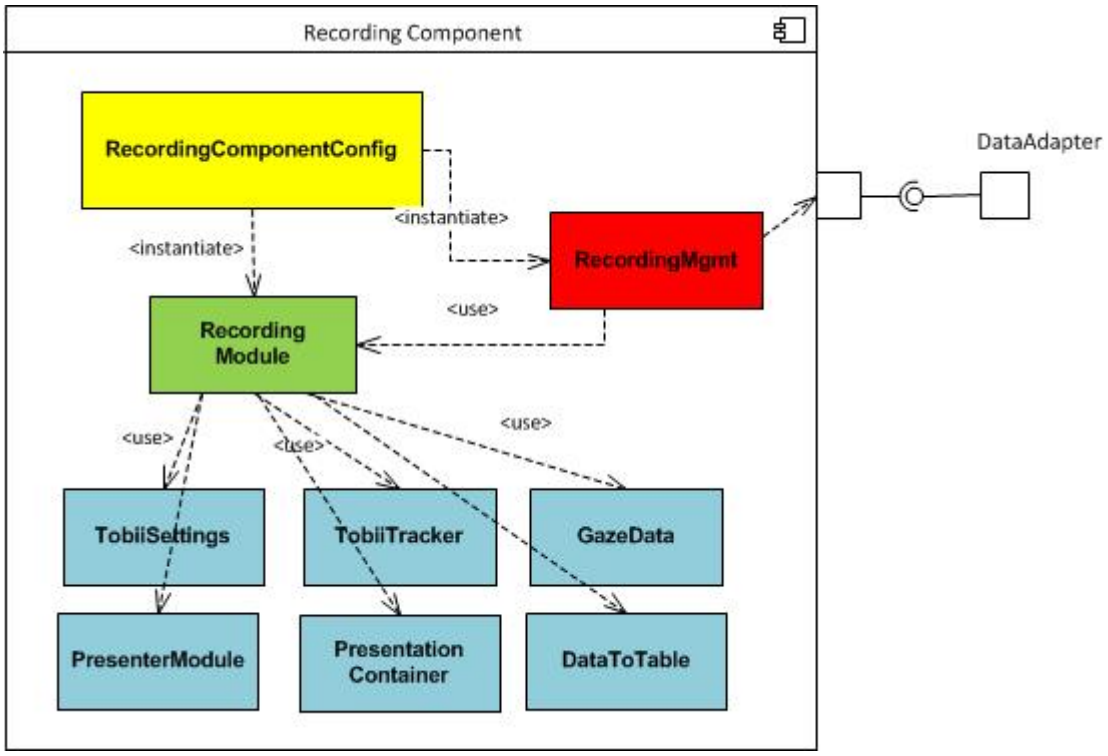


Figure 4.9: RecordModule, internal view.

The Presenter module and container classes wrap each slide of a slideshow and present them to the participant before triggering the record function.

The DataToTable class provides mapping information about which raw data will be written to which table of the database. The RecordModule also provides the user with a graphical interface to interact with.

4.1.6 Component „HeatMapModule”

4.1.6.1 Task and responsibility of the component

The HeatMapModule delivers one of the visualization capabilities of the application: the heat map. Its task is to generate a bitmap image visualizing aggregated fixation data overlaid one the stimulus image.

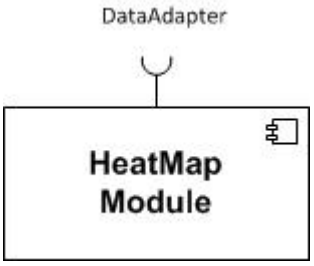


Figure 4.10: HeatMapModule, external view.

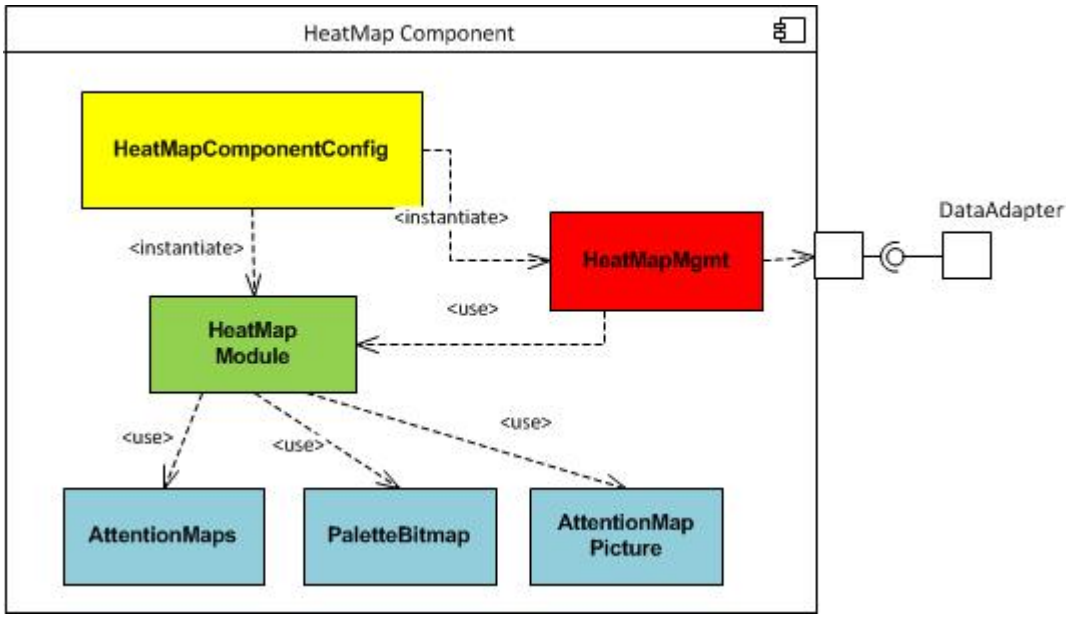


Figure 4.11: HeatMapModule, internal view.

4.1.6.2 External view of the component

The HeatMapModule uses the DataAdapter interface through the DataSet class to load fixation data from the database to calculate the heat map. It is a GUI-based component.

4.1.6.3 Internal view of the component

The configurator class HeatMapComponentConfig initializes the use-case class HeatMapMgmt and the HeatMapModule class, which uses the entity class AttentionMaps and PaletteBitmap to calculate heat maps from the fixation data and apply a colored gradient. The AttentionMapPicture class serves as a control to display the generated heat maps.

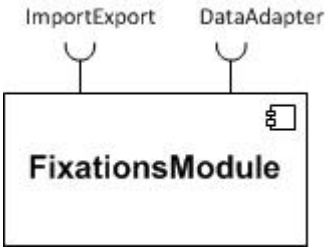


Figure 4.12: FixationsModule, external view.

4.1.7 Component „FixationsModule”

4.1.7.1 Task and responsibility of the component

The FixationsModule handles the detection and calculation of fixation patterns from raw gaze data. A fixation is defined when the user rests his eyes on a specific spot with a pre-defined radius over a pre-defined period of time. The fixation patterns are necessary to generate heat maps.

4.1.7.2 External view of the component

As depicted in Figure 4.12, the FixationsModule uses the DataAdapter interface through the DataSet class to fetch raw gaze data from the database / write calculated fixation data into the database and ImportExport module to load data from external files / export calculated fixation data to files.

4.1.7.3 Internal view of the component

The „FixationsComponentConfig” configurator class initiates the manager class FixationsModule and use-case class FixationsMgmt. The manager class uses the entity class FixationCalculation to initiate fixation calculation on the gaze data it loaded from the database or external files, as depicted in Figure 4.13. The ExportOptions and Dialogs classes provide the user with options that will affect the fixation calculation algorithm via dialogs.

4.1.8 Component „ImportExportModule”

4.1.8.1 Task and responsibility of the component

The ImportExportModule enables the application to read and write settings as well as various data from/to files. This functionality is an approach to stability, reliability and usability requirements. User can save his work for later use and export various tables of the experi-

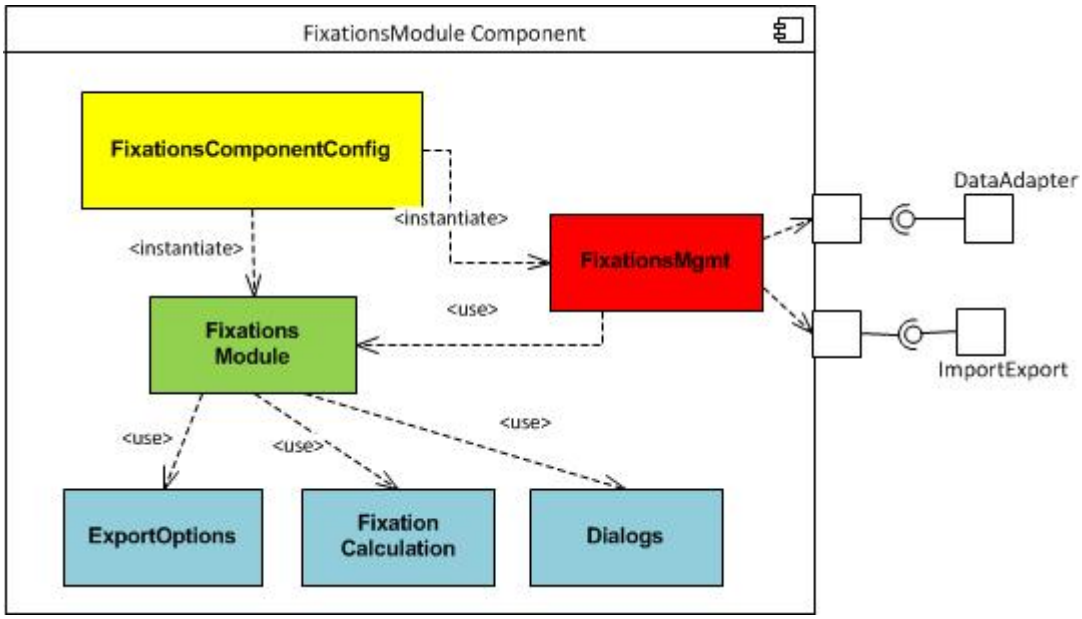


Figure 4.13: Fixations component, internal view.

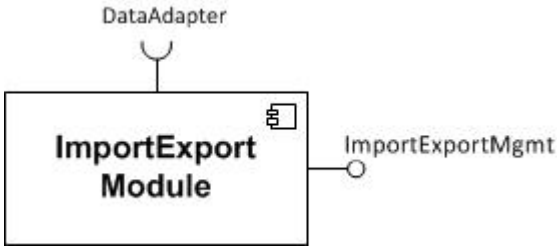


Figure 4.14: ImportExport component, external view.

ment database (which are the result of various calculation of the modules, e.g fixation data, raw data. . .) to external files.

4.1.8.2 External view of the component

The ImportExportModule interfaces with the DataAdapter interface through the DataSet class to be able to interact with the underlying database, and offers an interface for other components to be able to use the ImportExport functionality, as depicted in Figure 4.14.

4.1.8.3 Internal view of the component

The import mechanism is a multiple steps process, each of which is a procedure responsible for different types of data that the application can imported. Each step will require

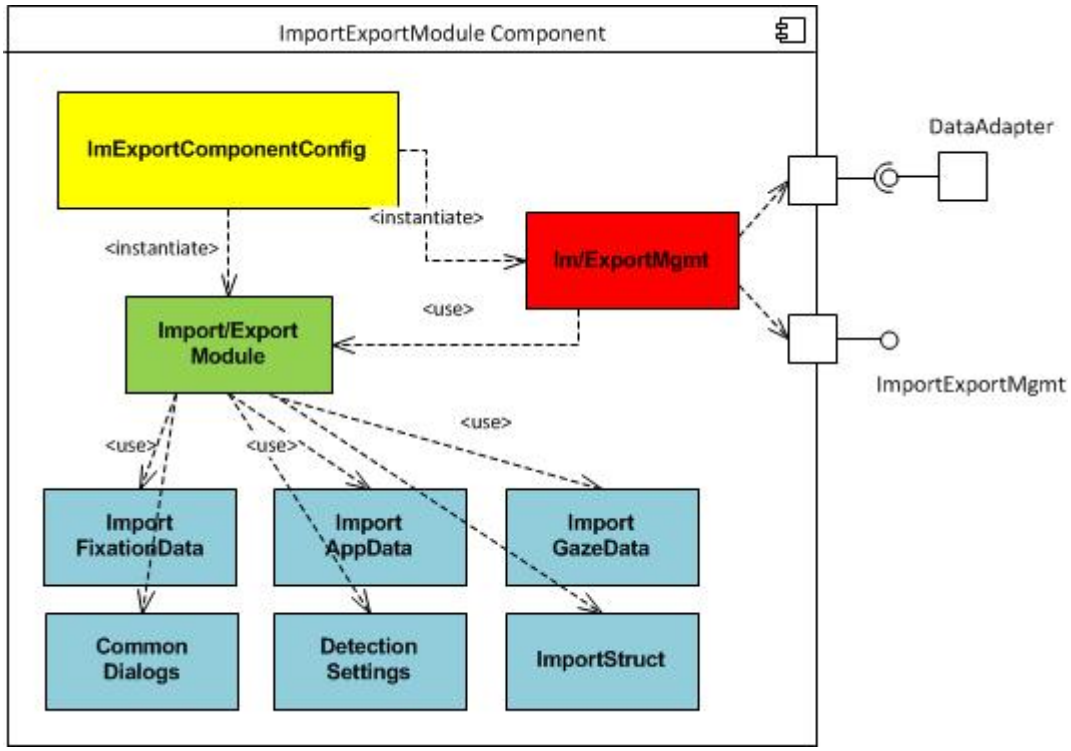


Figure 4.15: ImportExportModule, internal view.

a set of user defined rules in order to parse data correctly. The configurator class ImExportComponentConfig initiates the manager class ImportExportModule and use-case class ImExportMgmt. The ImExportModule class unifies these child procedures into one logical unit, and exposes it through ImportExportMgmt, as depicted in Figure 4.15.

4.1.9 Component „DatabaseModule”

4.1.9.1 Task and responsibility of the component

The DatabaseModule enables the user to revise the data that is in the database tables in the underlying database and also allows the user to modify this data and reflect the changes back to the database. This enables user to have fine granular control over the data that will be used throughout the application.

4.1.9.2 External view of the component

The DatabaseModule interfaces with the DataAdapter interface through the DataSet class to read and write data from/to the database and ImportExportMgmt to import/export data from/to external files.

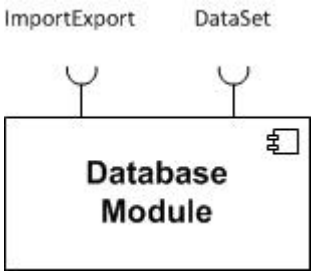


Figure 4.16: DatabaseModule, external view.

4.1.9.3 Internal view of the component

The DatabaseComponentConfig configurator class initializes the use case class DatabaseMgmt and the DatabaseManager class. The DatabaseManager class uses various entity classes to wrap different types of data internally, as depicted in Figure 4.17.

4.1.10 Database layout

The application stores the collected gaze tracking data in a .mdf file, which is a SQL-Server database file extension. These files are separately created for each test. The underlying database layout is shown in Figure 4.18. The whole database is sub-divided into seven tables. The following section describes each of the tables in details.

The „Subjects" table holds the list of participants with their associated descriptive attributes like category, sex, age, etc. . . . These pre-defined attributes can also be extended with custom variables. These custom variables will be stored in the „SubjectParameters" table, which contains the participant's name and a key-value pair for the parameter.

The „Trials" table holds the (meta) information of all trials. Information of each trial of the test, for each participant is stored with associated variables, such as Id, name, sequence, start time, duration, etc. . . along with a column indicating the validity of the recorded data.

For each participant a „RawGazeData" table is created which contains the imported or real-time eye-gaze tracking data and the corresponding timing information. For each time stamp it can include up to two columns for pupil diameter data (PupilDiaX, PupilDiaY), two columns for gaze position data (GazePosX, GazePosY) and a column associated with an occurred trial event.

All the events that occurred during recording are stored onto the „TrialEvents" table, which

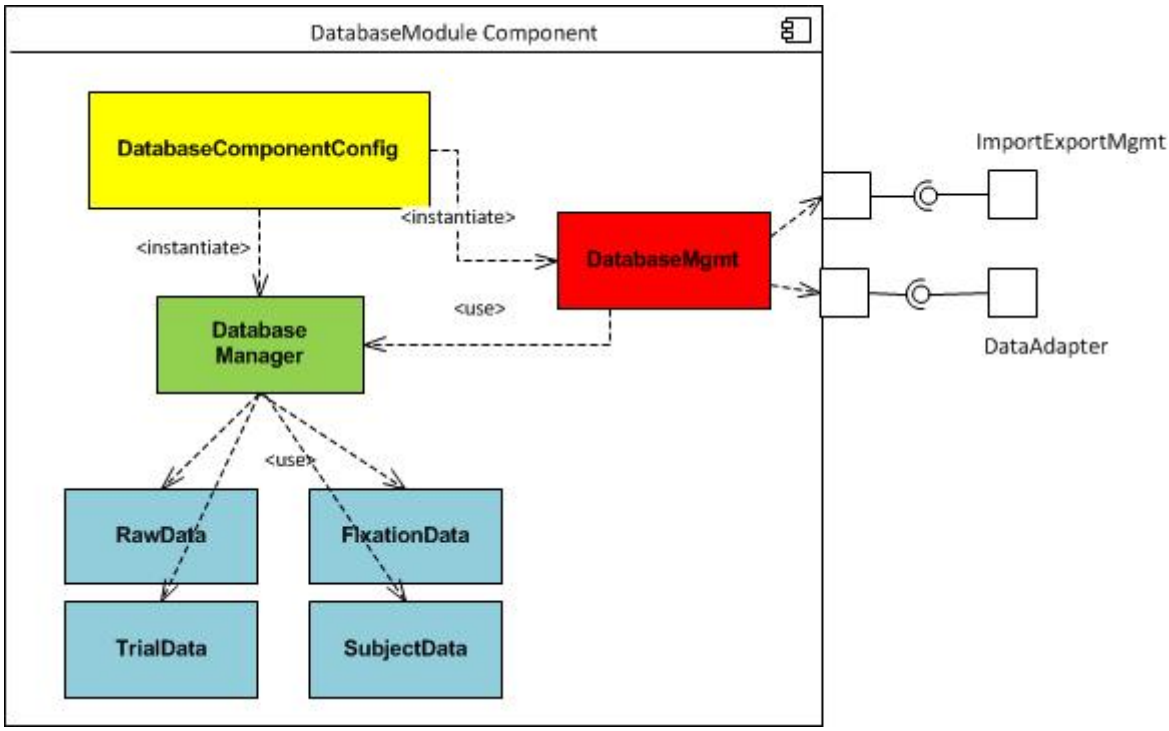


Figure 4.17: DatabaseModule, internal view.

holds the event's time, task, type and parameter. The application records for the scope of this report key, scroll, marker and user response events, but additional events, such as mouse, video, usercam, etc... can also be added. With them the trial timeline is separated and interesting points in time during the recording session can be marked.

Finally, a table lists the gaze fixation information for each participant and trial with its associated variables, such as the number of fixation in a trial (CountInTrial), the starting time (StartTime), duration of the fixation (Length) and the coordinate of the fixation (PosX, PosY).

All the above seven tables resides in a locally stored SQL-database file, which is created separately for each test. Data integrity is ensured over foreign key constraints between these tables as shown in Figure 4.18.

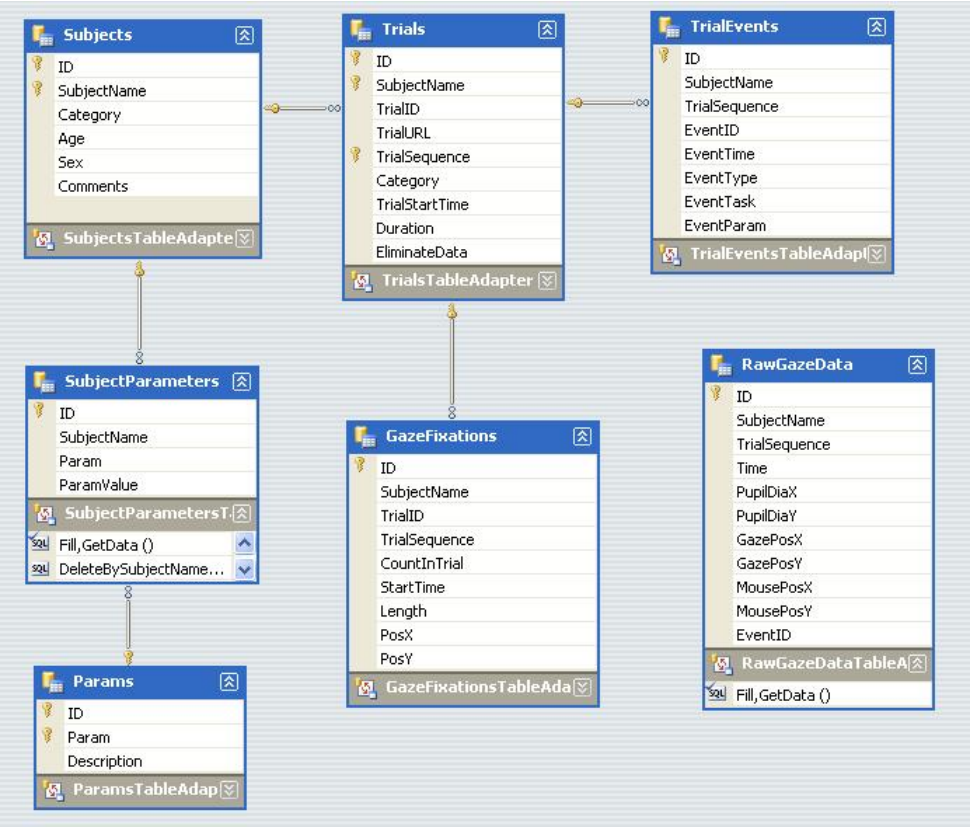


Figure 4.18: Database design of the application.

4.2 Class diagram and business processes

This section describes the class diagram as well as main business processes of the application. These processes included „Design a test", „Record a test" and „Visualize data" , depicted as EPC[11] diagrams.

[11] Event-driven process chain.

4.2.1 Class diagram

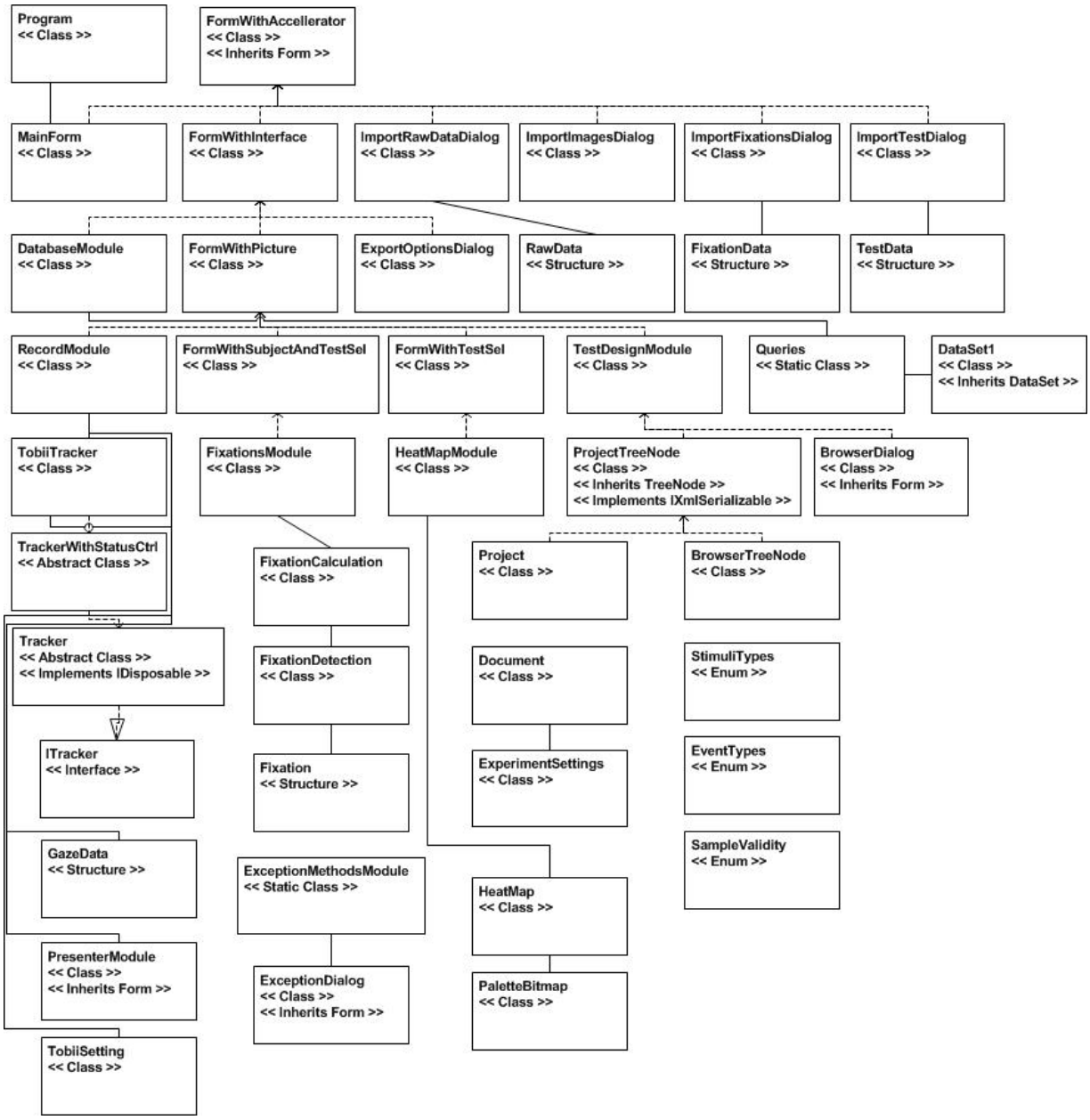


Figure 4.19: Class diagram of the application.

The MainForm class (acts as MDI parent) of the application derives from the base class FormWithAccellerators, which inherits from Window.Form and adds support to shortcut keys.

The FormWithInterface class derives from FormWithAccellerators and is the base class for forms with controls like DataGridView, ListView, TreeView, etc... with a data source. The FormWithPicture class inherits from FormWithInterface to add support for a Picture, which is a control to show dynamic content.

There are two more types of form that derive from FormWithPicture, which are FormWithSubjectAndTestSel and FormWithTestSel, which add support for drop-down boxes to select participant and test. There are a few static classes, e.g Queries, ExceptionHandling which provides utility methods. DatabaseModule makes use of the static methods in the Queries class to send SQL commands to the DataSet.

The RecordModule makes use of the TobiiTracker class, which represents the Tobii eye tracker and provides methods to interact with the Tobii API. The TobiiTracker class inherits from the TrackerWithStatusCtrl class, which in turns inherits from the abstract class Tracker, which implements the ITracker interface. The ITracker interface contains the basic methods to interact with an hardware tracker device. The TrackerWithStatusCtrl implements this interface and extends the tracker with a status control window, as in Tobii eye trackers.

The PresenterModule class is a component that presents the stimuli to test subjects on the presentation screen.

The Document class is a singleton class that represents an active project together with its settings and states.

The Program class contains the Main() method that starts the application.

4.2.2 Business Processes

4.2.2.1 Business process „Design a test”

Description: First, the experimenter has to create and name a new project, or open an existing project. For the case of new project, the experimenter then has to specify project settings. Next, the experimenter starts the TestDesign module and begins adding stimuli to the test. After that, the experimenter can choose to save the whole project.

4.2.2.2 Business process „Record a test”

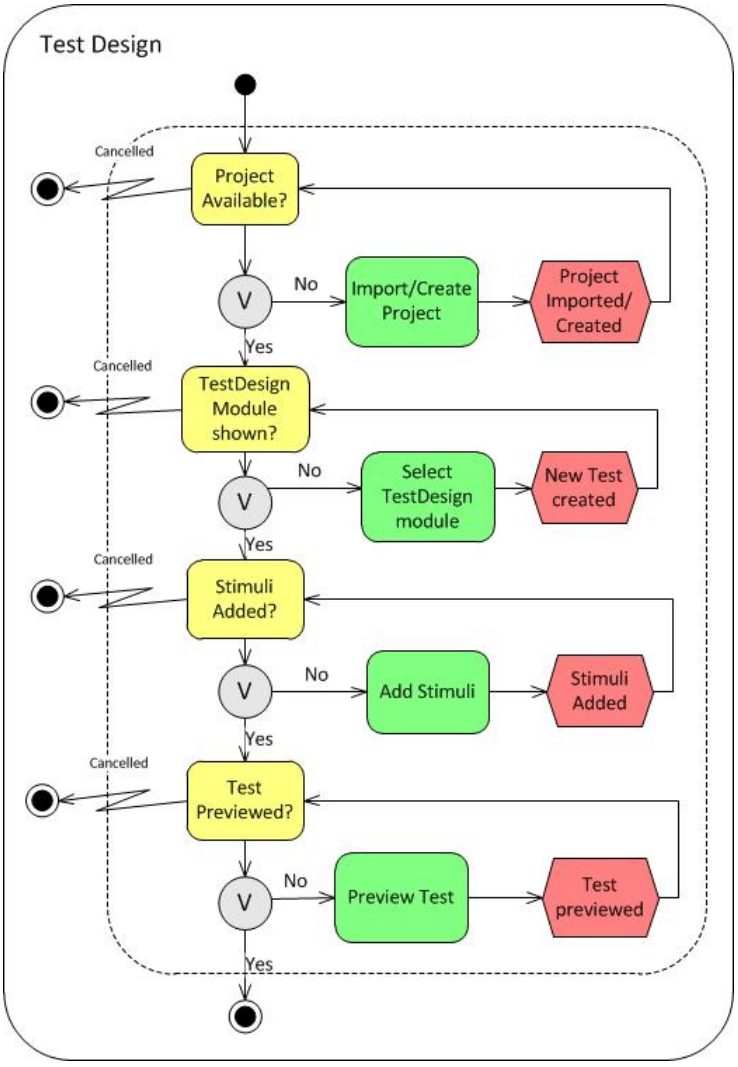


Figure 4.20: „Design a test” business process.

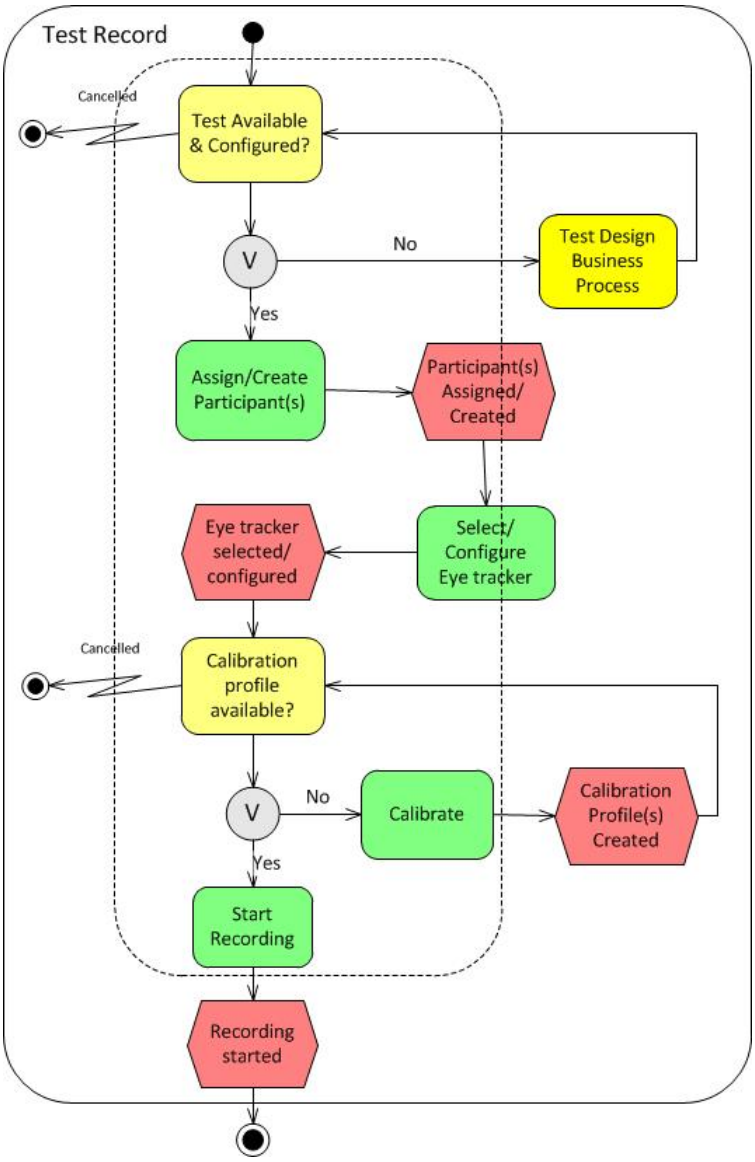


Figure 4.21: „Record a test” business process

Description: The experimenter has to set up and configure a project, design at least a test which contains at least one stimulus, starts the record module and assigns at least one test subject to the test. If this is a new subject, a profile will be created, otherwise the experimenter can choose from a list of existing profiles. Then the experimenter chooses the eye tracker that is to be used and clicks the record button. In the case of new subject, a calibration procedure will be fired up immediately to gather eye-gaze data and produce a calibration profile for that particular test subject. Existing test subject can decide for a recalibration process. Afterwards, the record module will present the stimuli to the test subject and record process will begin.

4.2.2.3 Business process Visualize data"

Description: When a record session is finish, raw gaze data will be available in the database. To visualize those data, the experimenter first decides a visualization method, then starts the according module. The experimenter then chooses a test subject and/or a recording, whose data is to be visualized. After the visualization process has finished, the result is shown to the experimenter.

4.3 Implementation

4.3.1 Setting up the working environment

- Installing the .NET Framework 3.5 (with Service Pack 1) and 4.0.
- Installing the Microsoft Visual Studio 2010 with the IDE settings configured to Visual Basic Development Settings.
- Installing the Tobii SDK and all the components required for the development, TetComp.dll, tet.dll and ttime.dll, using the standard installer of the Tobii SDK.
- Installing the Tobii Studio 2.0.
- Getting access to the Tobii Eyetracker X120 for testing purpose (in the usability lab.).

When the above steps are done and the development environment is ready on the machine, then the next thing to do is to start the Visual Studio and create a new VB project (Windows application).

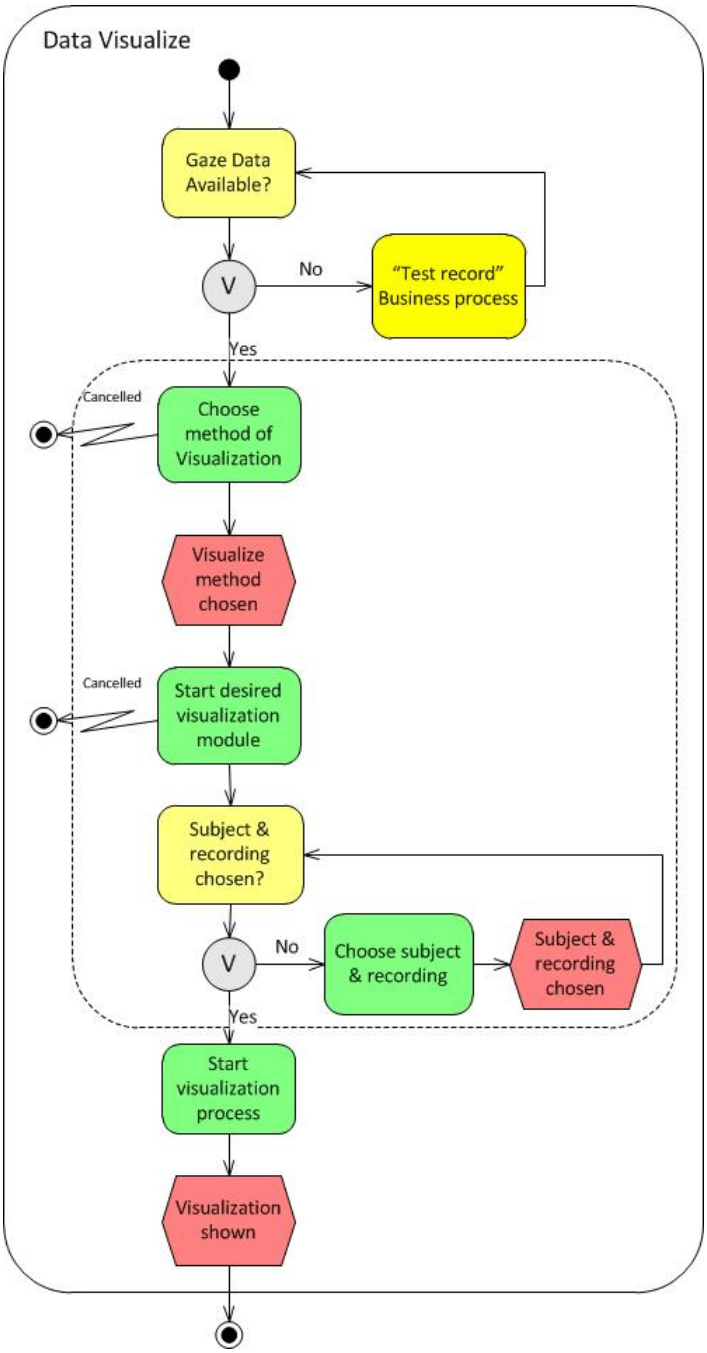


Figure 4.22: „Visualize data” business process

4.3.2 General workflow

The general workflow of the application is depicted in Figure 4.23. One would first either design a test with the TestDesignModule, then move on to record the test with the RecordModule, or switch on the ImportExport module to import existing data into the application. Either way, when the application had some data in the database (first survey), then the user can move on to use the visualization and analysis tools over these data (using the other modules).

4.3.3 Storage files

The application creates a folder named „VinhBAThesisExperiments" in the „MyDocuments" folder of the current user. Each project has its own folder with at least three sub-directories. The .va file, which has an XML layout, stores the metadata and description for each of the test. This metadata and description include the database files, the path to stimuli, sampling rates, screen size, parameters for fixation calculation and a complete description of the tests of the project. The application can import this file to restore a previously saved session. The Database subfolder contains two MSSQL database files. Because of the pure MSSQL standard, these two file can be accessed with any MSSQL enabled language or application, such as Microsoft Access or SQLExpress. All stimuli resources that are used in the test will be placed in the SlideResources folder.

4.3.4 Fixation detection

In order to move on further visualization capabilities, we must create a basis from the raw gaze data. There are a variety of taxonomy and models of eye movements, such as smooth pursuits, nystagmus, extraocular muscles, etc .... Fixations and saccades are two most important of them and will lay the foundation for other visualization and analysis modules in the scope of this application. To be precise, fixation patterns calculated in the FixationsModule will be the basis for the HeatMapModule to generate heat maps. We define fixation and saccade as follow:

- Fixations are eye movements that stabilize the retina over a stationary object of interest. It seems intuitive that fixations should be generated by the same neuronal circuit controlling smooth pursuits with fixations being a special case of a target moving at zero velocity. This is probably incorrect (Leigh & Zee, 1991, pp. 139-140). Fixations, instead, are characterized by the miniature eye movements: tremor, drift, and microsaccades. Duchowski (2007)

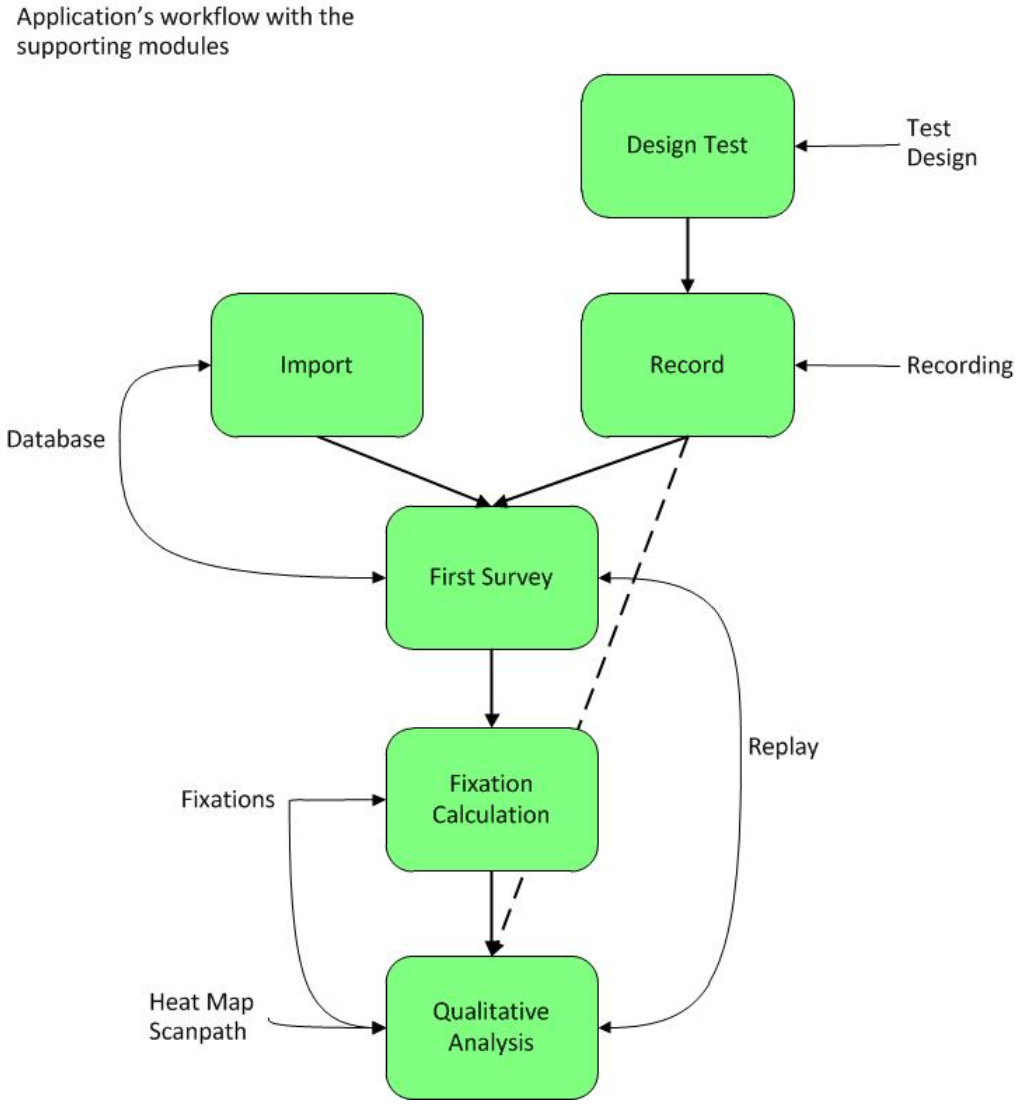


Figure 4.23: Application's workflow with the supporting modules.

- Saccades are rapid eye movements used in repositioning the fovea to a new location in the visual environment. This term comes from an old French word meaning „flick of a sail" (Gregory, 1990). Saccadic movements are both voluntary and reflexive. Saccades range in duration from 10 ms to 100 ms, which is a sufficiently short duration to render the executor effectively blind during the transition (Shebilske & Fisher, 1983). Duchowski (2007)

With the knowledge about fixations and saccades, the next thing to do is to implement some algorithms to detect and calculate fixations. In Eye Tracking Methodology, we have the following information:

The dwell-time fixation detection algorithm depends on two characterization criteria:

1. Identification of a stationary signal (the fixation).
2. Size of time window specifying an acceptable range (and hence temporal threshold) for fixation duration.Duchowski (2007)

For the scope of this report (also implemented in the application), we will do the fixation calculation using the fixation detection algorithm published by LC Technologies. It is a dispersion type algorithm with a moving window Salvucci und Goldberg (2000). The source code was ported to VB.NET but the working principal stayed the same. The source code documentation of this algorithm describes the working principle as follows: „The function detects fixations by looking for sequences of gaze-point measurements that remain relatively constant. If a new gaze point lies within a circular region around the running average of an on-going fixation, the fixation is extended to include the new gaze point. (The radius of the acceptance circle is the above user specified maximum distance.) To accommodate noisy measurements, a gaze point that exceeds the deviation threshold is included in an on-going fixation if the subsequent gaze point returuns to a position wihin the threshold. If a gaze point is not found, during a blink for example, a fixation is extended if:

1. The next legitimate gaze point measurement falls within the acceptance circle, and
2. There are less than the above minimum numbers of samples of successive missed gaze points. Otherwise, the previous fixation is considered to end at the last good gaze point measurement." (LC Technologies 2006)

Prior to the calculation, the detection algorithm applies a two steps filter. In the first step, empty values will be ignored and filtered out. This is necessary due to the different sampling rates of the gaze (and possibly mouse) data. The second step involves omitting samples with both X- and Y- coordinates equal to zero, which often marks the eye tracker output during an eye blink, and omitting also samples that lies out of the screen.

The application provides the possibility for the user to specify two parameters that will be used to fine-tune this algorithm:

- the maximum distance (in pixels) that a point may vary from the average fixation point and still be considered to be part of the fixation, and
- the minimum number of samples that defines a fixation.

These two parameters will be chosen based on the experimental sampling rate and the research domain. (Karsh & Breitenbach 1983.)

4.3.5 Implementing the modules.

This section contains a dicussion on the implementation of the different modules of the application. The workflow of the application is depicted in Figure 4.32. A new project is started either by designing a test (section 4.3.5.1) and recording new data (section 4.3.5.2), or by importing existing raw sampling data using the application's import wizard (section 4.3.5.3). When raw data are available in the application, using one of the two possible means discussed above, the user can let the application visualize or analyze them with the support of different modules, which, in the scope of this report, are the fixations, database and heat map modules. Each of these modules is specialized on its intended purpose. They can be grouped into two sets, one for visual data analysis which are Fixations and Heatmap, and one used for underlying data control, which is Database module.

The following feature depicts the VB project file structure of the custom application on Visual Studio 2010.

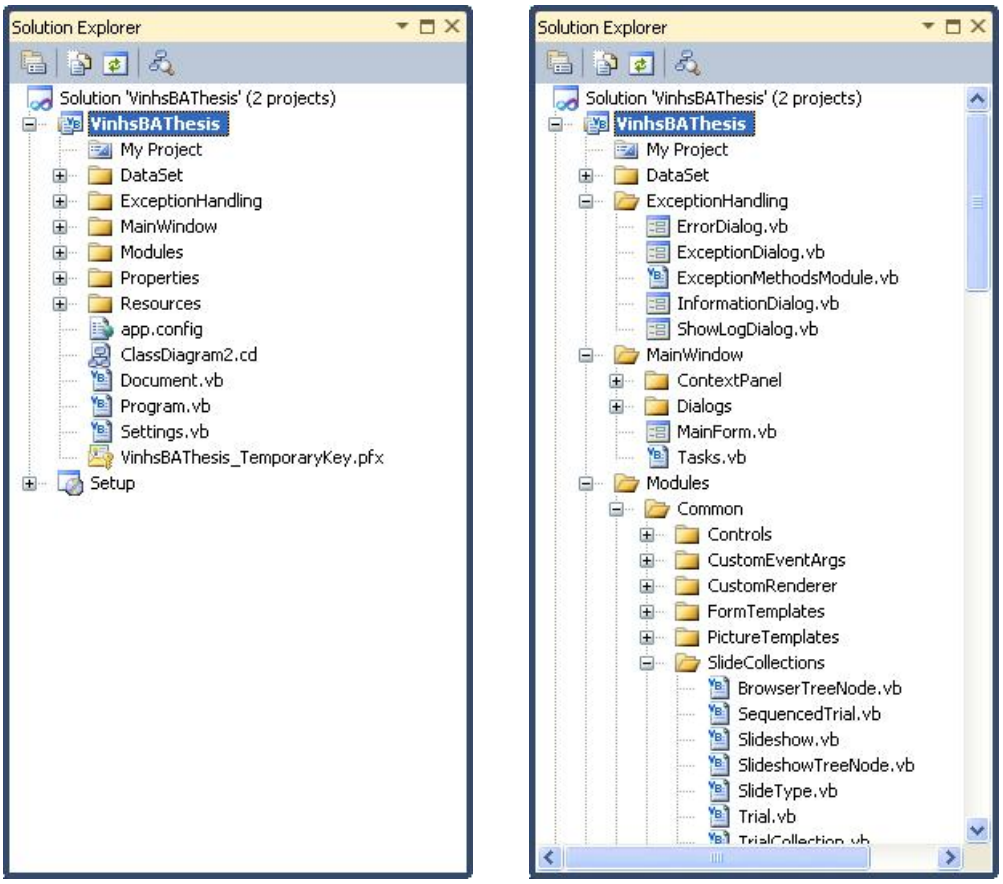


Figure 4.24: Visual Studio solution explorer for the custom application's VB project.

4.3.5.1 Namespace „MainWindow"

This namespace contains the main frame of the application, which is a Multi-document-interface window. All other modules reside in this window, which also hosts a menu, a toolstrip, a status strip and a context panel. The menu provides the basic functionality for the application like opening, saving, closing and creating new project (File menu), copy and paste methods for table and module content (Edit menu), creating and showing different modules (View menu), modification of project settings, application settings, database connection settings (Options menu) and a help section (Help menu). The toolstrip enables control over the functionality of the modules and the status strip is there to show current workflow and progress information. The main window is designed so, that the interface stays intuitive to basic users, and yet still provides enough information and fine-tuning possibility to more advanced users.

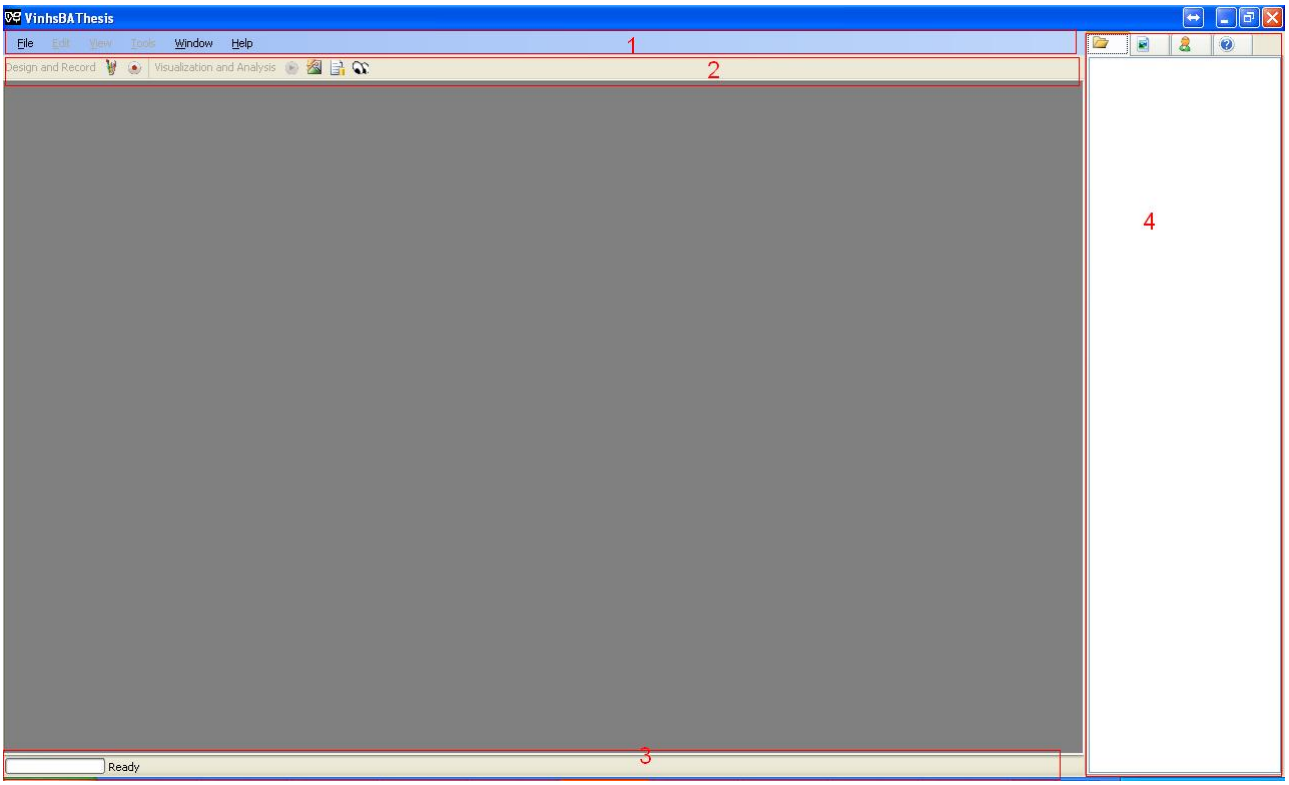


Figure 4.25: User interface of the main window, with a menu (1), a toolstrip (2), a status bar (3) and the context menu (4).

The main window is also equipped with a context panel on the right hand, which shows list of participants and all the designed tests in a project, so that the user can easily select a particular test and/or a specific participant for analysis in the current active module. The following sections describe each of the modules in detail and give an examplary screenshot created with a demo project.

4.3.5.2 Namespace „Module.TestDesign ”

The TestDesign module enables the creation of test with stimuli to be presented to the participant during an experiment. For the scope of the report, the application supports only one type of stimulus, which is the Browser (or Web) stimulus. The application supports creation of a sequence of stimuli, which is visualized by a TreeView and a TimeLine. Selecting a stimulus display its content in the preview window.

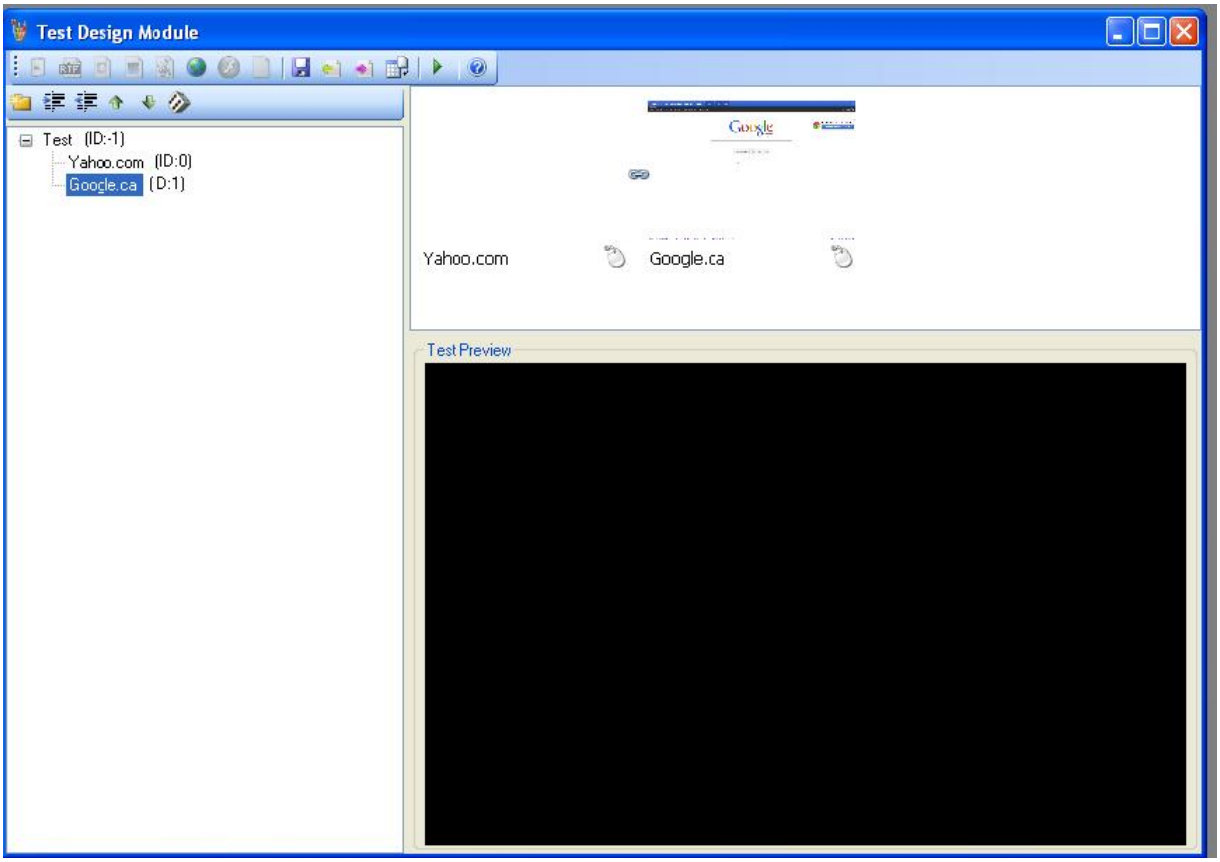


Figure 4.26: TestDesign module with tree view, timeline and preview.

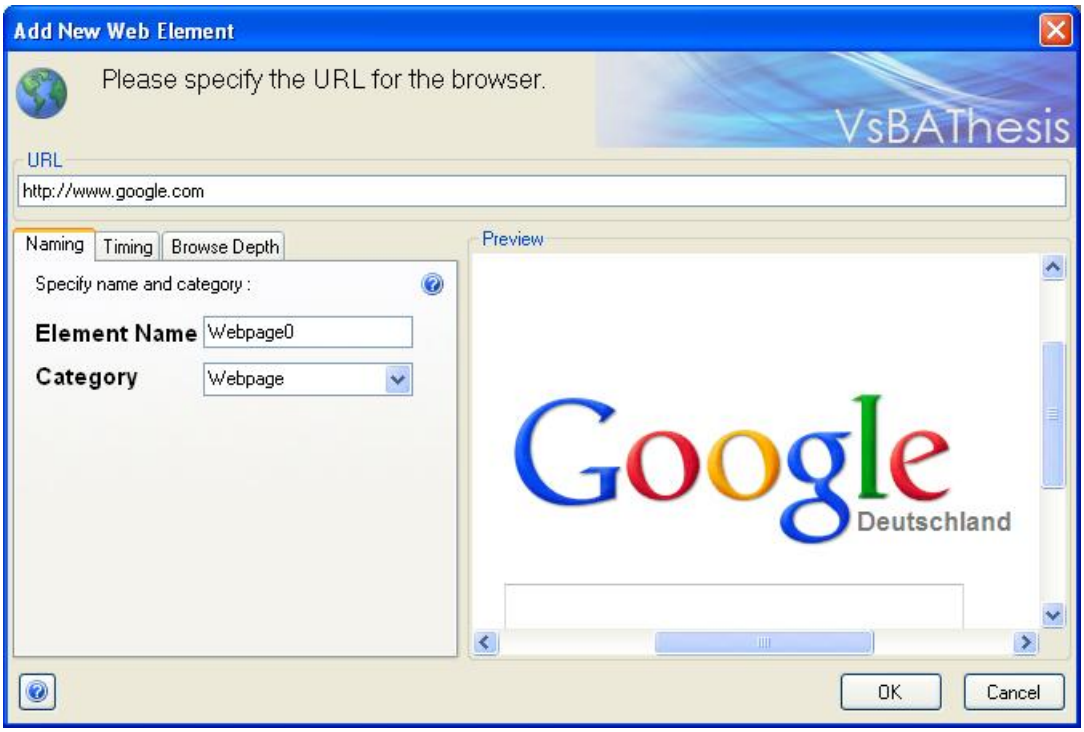


Figure 4.27: The dialog for Web stimulus creation.

The user start adding stimuli to the test by selecting from a variety of stimulus types representing by the buttons on the toolstrip. This will open a dialog with the settings necessary for that particular stimulus type. There is also a folder import mode, which will facilitate the import of a lot of stimuli files. The user can specify general stimuli properties for all imported stimuli and can later modify them via custom dialogs.

The stimuli can be reordered or grouped in a hierarchical order that is visualized via the tree view on the left side of the module. The „PreviewTest" button helps the user to preview the complete presentation for testing purposes. In this case the complete test is validated but no recording is made.

```
'Starts the PresenterModule with the given stimuli.
Private Sub btnPreviewTest_Click(ByVal sender As Object, ByVal e As
    System.EventArgs) Handles btnPreviewTest.Click
        If Not RecordModule.
            CheckForCorrectPresentationScreenResolution() Then
```

```
                Exit Sub
            End If
            Dim objPresenter As New PresenterModule()
            ' Create a newly randomized trial list.
            Dim trials As TrialCollection = Me.slideshow.
                GetRandomizedTrials
            ' Create a hardcopy of the trials.
            Dim copyOfTrials As TrialCollection = DirectCast(trials.
                Clone(), TrialCollection)
            ' Set slide list of presenter
            objPresenter.TrialList = copyOfTrials
            ' Show presenter form, that starts presentation.
        objPresenter.ShowDialog()
End Sub
```

The user also has the option to modify the stimuli by double-clicking on it in the time line to open the stimulus properties dialog, in this case, the Add New Web Element dialog in Figure 4.36, again.

The properties needed to add a Web Element will be covered in details below:

The most essential and basic information for a Web Element is the web URL, so this piece of information is not allowed to be null, in that case the application will show a message box telling the user that the URL field cannot be empty and stay on the same dialog for retry. The same rule applies for the name field of the Web Element. The Category field is optional.

```
Private Sub btnOK_Click(ByVal sender As Object,
            ByVal e As System.EventArgs) Handles btnOK.Click
            If Me.txbURL.Text = String.Empty Or Me.txbName.Text =
                String.Empty Then
                    MsgBox("The URL field cannot be empty!", vbOKOnly,
                        "Empty URL")
                    Me.DialogResult = DialogResult.Retry
            Else
                    Me.DialogResult = DialogResult.OK
            End If
End Sub
```

In the Timing tab, the user has the option to control when to end the stimuli show, either after a specific time elapse, or when the user press a particular key. The Browse Depth tab enables the user to restrict the browsing at a specific depth level.

The Preview panel hosts a web browser control that is updated in realtime, as the user types in the URL. If the URL field is invalid, it waits for a valid one.

The application tries to provide the user with a WYSIWYG process, i.e each modification is reflected instantly in the preview window on the right side of the TestDesign dialog. This feature has not been working properly at the moment, though.

The application can save the work of the user, storing them in Xml format in the project .va file. Each test can also be exported separately to other project in a special Xml format with the extension .vas

4.3.5.3 Namespace „Module.Recording"

The Recording module is designed to capture the participant - subject of the test 's view during the presentation of the stimuli in the experiment. The application is built to control and record from Tobii eye tracking systems.

To start a recording, the application's workflow involves foursteps:

1. Connecting: build the communication between the application and the hardware eye tracker.
2. Participant assigning: prompt user for participant's information and assign to the recording.
3. Calibration: start the calibration process for the Tobii eye trackers.
4. Recording: show the stimuli on the presentation screen and start collecting gaze data.

User can change the settings associated with Tobii eye trackers via a customized dialog. These settings include the eye tracker's IP address, host name, port number, the number of calibration points, their size and color. The settings that user specified in this dialog are then stored in TobiiSetting class, which encapsulates them. At the end of a recording session, the collected gaze data is stored into the application's database and is immediately available for analysis and visualization.

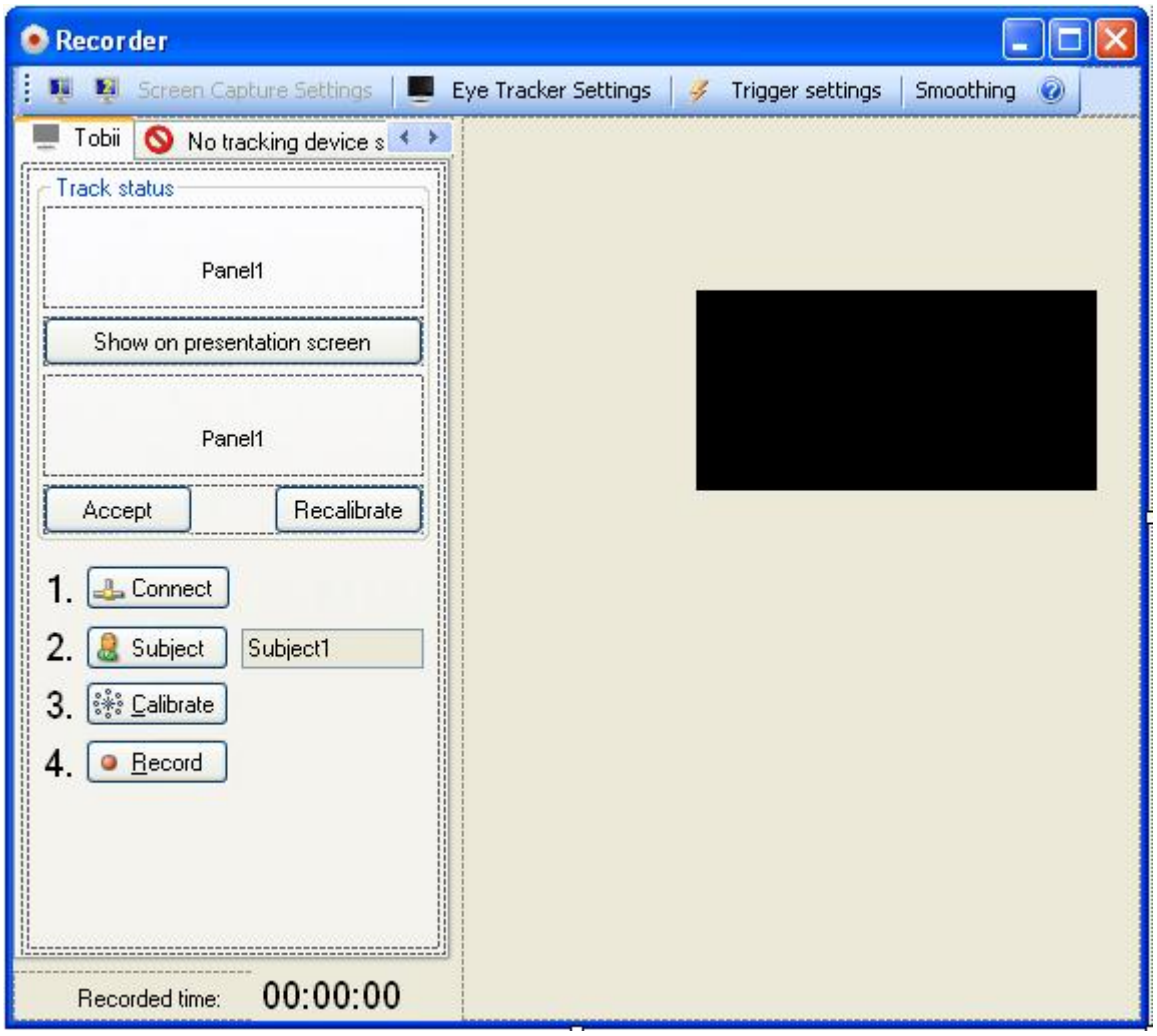


Figure 4.28: Screenshot of the Recording module.

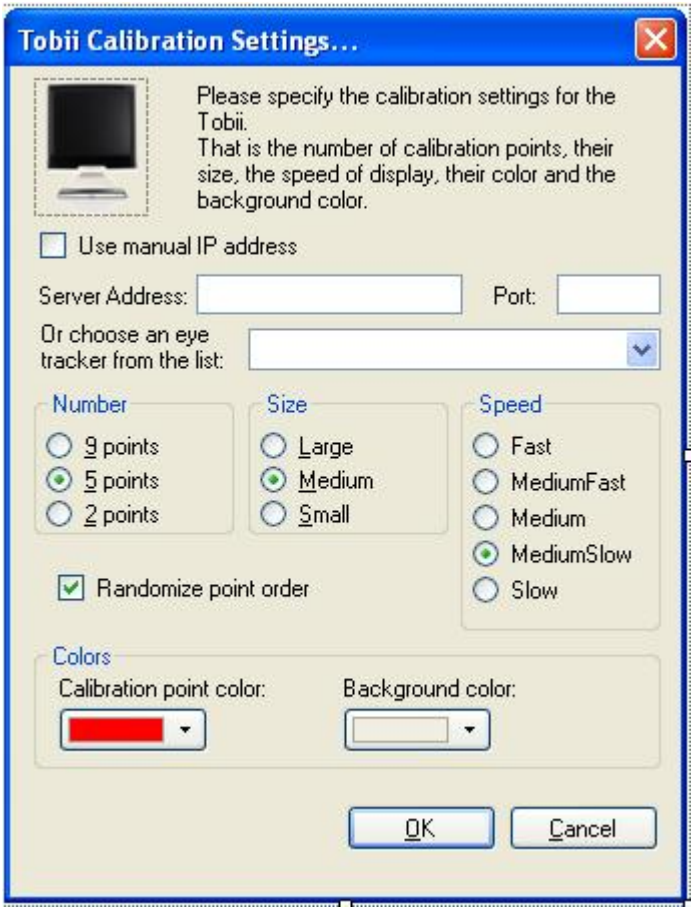


Figure 4.29: Tobii trackers-specific settings dialog.

The TobiiSetting class looks as follows:

```
' Class to save settings for the tobii eye tracking system.
' It is XML serializable and can be stored in a file via
' the XmlSerializer class.
Public Class TobiiSetting
        Private _tetServerAddress As String
    Private _tetServerPort As Integer
    ' Number of calibration points
        Private tetNumCalibPoints As TetNumCalibPoints
    ' Calibration point size
        Private tetCalibPointSize As TetCalibPointSize
    ' Calibration speed
        Private tetCalibPointSpeed As TetCalibPointSpeed
    ' Color of calibration points
        Private tetPointColor As Color
    ' Saves the background color of the calibration procedure.
    Private tetBackgroundColor As Color
    ' Saves the flag, whether the calibration point order should
    ' be randomized
        Private _tetRandomizeCalibPointOrder As Boolean
        .............
End Class
```

During load, the Recording module scans for connected Tobii eye trackers and does the following works: it updates a list of available eye trackers and displays the control panel associated with them. Starting with SDK version 2.0, Tobii enhances the user with a new feature called the Eyetracker Browser Interface, which simplifies greatly the eye tracker hardware discovery process and thus saves the user much effort from having to enter the correct IP address and port number for the eye tracker manually. The main component of this feature is the TetServiceBrowser class. We implement this feature as follows:

First we declare a variable that holds the reference to an instance of the TetServiceBrowser class and a list that holds all the discovered eye trackers:

```
Private WithEvents _serviceBrowser As TetComp.TetServiceBrowser
Private Shared _serviceList As List(Of TETServiceEntryWrapper)
```

We start the service browser during the load event of the form, and stop the service again during the closing event of the form.

The TETServiceEntryWrapper, as its name implies, is used as a wrapper class to encapsulate a TETServiceEntry, which represents an eye tracker, and is declared as follows:

```
Public Class TETServiceEntryWrapper
    Private _entry As TetServiceEntry
        Private _serviceName As String
        .......
End Class
```

This class overrides the ToString(), Equals() and GetHashCode() methods, as well as provides two properties which returns the host name of the eye tracker and a flag indicating whether this eye tracker is running:

```
Public ReadOnly Property Hostname As String
Public ReadOnly Property IsRunning As Boolean
```

Next, we implements three event handlers to handle the events that the TetServiceBrowser class might generate, which are listed below in pseudo code:

```
Private Sub ServiceBrowser_OnServiceRemoved(ByVal serviceName As
    String) Handles _serviceBrowser.OnServiceRemoved
    // remove the service entry from the service list.
        // update the eye tracker combo box
End Sub
Private Sub ServiceBrowser_OnServiceUpdated(ByRef serviceEntry As
    TetServiceEntry) Handles _serviceBrowser.OnServiceUpdated
    If the service list does not contain the service entry Then
        // Add service entry to the service list
        Else
                // First remove the entry from the service list.
                // then re-add the (newer) entry to the service
                   list
    End If
        // update the eye tracker combo box
End Sub
```

```
Private Sub ServiceBrowser_OnServiceAdded(ByRef serviceEntry As
    TetServiceEntry) Handles _serviceBrowser.OnServiceAdded
     // Call the ServiceBrowser_OnServiceUpdated procedure and pass
        // the serviceEntry as its parameter
        // update the eye tracker combo box
End Sub
```

Finally, we provide a method to update the eye tracker combo:

```
Private Sub UpdateEyetrackerCombo()
        //clear the eye tracker combo's list
        //for each eye tracker detected, check if it is
        //running, if yes then add to the combo.
End Sub
```

The central point of interest of this Recording module is, of course, how the module communicates with the Tobii eye tracker and how the module treats new raw gaze data, once it arrives. We know that the communication is enabled by the Tobii API, and the implementation in details lies within the TobiiTracker class, which inherits the TrackerWithStatusControl class, which in turn inherits the abstract class Tracker, which in turn implements the ITracker and IDisposable interfaces. The abstract class Tracker is thus the base class for all the hardware trackers. This is done in order to enable more different hardware trackers at a later time to be added to the application. The ITracker interface looks as follows:

```
Public Interface ITracker
        ' Send the new sampling data at each sampling time interval
            .
         Event GazeDataChanged As GazeDataChangedEventHandler
        ' Gets a value indicating whether the tracker is connected
        ReadOnly Property IsConnected As Boolean
        ' Do all connection routines for the specific hardware.
        Function Connect() As Boolean
        ' Do the calibration routine for the specific hardware.
        Function Calibrate(ByVal isRecalibrating As Boolean) As
            Boolean
        ' Start the recording for the specific hardware.
        Sub Record()
        ' Stop the recording for the specific hardware.
```

```
        Sub StopRecord()
        ' Do a clean up for the specific hardware.
        Sub CleanUp()
        ' Show a dialog to change settings of a specific hardware.
        Sub ChangeSettings()
End Interface
```

More hardware trackers can be added later to the system by adding entries to the HardwareTracker enumeration and inheriting the Tracker class.

```
Public Enum HardwareTracker
        ' No tracker available
        None = 0
        ' Tobii trackers available
        Tobii = 2
End Enum
```

The TobiiTracker class inherits from the TrackerWithStatusControl class, which represents all the eye trackers that provide a status control window to indicate live track status, i.e the track quality so that the experimenter has complete control over the recording. This code snippet demonstrates how we communicate with the Tobii eye tracker and start the tracking status control:

```
Public Class TobiiTracker
        Inherits TrackerWithStatusControls
        Public Overrides Function Connect() As Boolean
                Try
                'Connect to the TET server if necessary
            If Not Me.tetTrackStatus.IsConnected() Then
                                Me.tetTrackStatus.Connect(Me.
                                    tobiiSetting.TetServer
                                Address, Me.tobiiSetting.
                                    TetServerPort)
                        End If
                        If Not Me.tetTrackStatus.IsTracking Then
                    Me.tetTrackStatus.Start()
            End If
        Catch ex As Exception
```

```
            Dim dlg As New ConnectionFailedDialog
            dlg.ErrorMessage = ex.Message
            Me.CleanUp()
            Return False
        End Try
            Return True
        End Function
.........
```

Next, we handle the calibration routine:

```
Public Overrides Function Calibrate(ByVal isRecalibrating As
    Boolean) As Boolean
        // Connect the calib procedure if necessary
        Me.tetCalibProc.WindowTopmost = True
        Me.tetCalibProc.WindowVisible = True
        Me.tetCalibProc.StartCalibration(
          If(isRecalibrating,
                TetCalibType.TetCalibType_Recalib,
                        TetCalibType.TetCalibType_Calib),
                                    Me.tobiiSettings.
                          TetRandomizeCalibPointOrder)
        return true
        // Catch exception if necessary and return false
End Function
```

Next, we handle the recording routine:

```
Public Overrides Sub Record()
   // Connect to the TET server if necessary
   If Not Me.tetClient.IsTracking Then
       'Start tracking gaze data
       Me.tetClient.StartTracking()
   End If
        // Catch exception if necessary, display error  // message
           and stop recording.
End Sub
```

Then the stop record routine:

```
Public Overrides Sub StopRecord()
        Try
                If Me.tetCalibProc.IsConnected Then
               If Me.tetCalibProc.IsCalibrating Then
                Me.tetCalibProc.InterruptCalibration()
           End If
         End If
       If Me.tetClient.IsConnected Then
            If Me.tetClient.IsTracking Then
                Me.tetClient.StopTracking()
         End If
           End If
    // Catch exception if necessary and display error
        // message.
End Sub
```

Next, we discuss about how the application handles newly arrived gaze data. The application defines its own structure for gaze data and converts Tobii eye tracker's internal gaze data structure to this structure, once it arrives. The application will write into the database its own internal format for eye gaze data.

```
Public Structure GazeData
        ' Time in milliseconds from the start of the recording.
        Public Time As Long
        ' x-diameter of pupil
        Public PupilDiaX As Single?
        ' y-diameter of pupil
        Public PupilDiaY As Single?
        ' x-coordinate of gaze position in values ranging between
           0..1
        Public GazePosX As Single?
        ' y-coordinate of gaze position in values ranging between
           0..1
        Public GazePosY As Single?
End Structure
```

Then we handles the newly arrived gaze data

```
Private Sub tetClientEvents_OnGazeData(ByRef tobiiData As Object)
        Dim myGazeData As New GazeData
        Dim tobiiGazeData As GazeDataHolder = tobiiGazeData
        // Convert Tobii gazestamp into milliseconds

        If ValidityLeftEye = 0 AndAlso ValidityRightEye = 0 Then
                // Let the x, y and distance be the right and
                // left eye average and assign calculated
                // values to myGazeData
        ElseIf ValidityLeftEye = 4 AndAlso ValidityRightEye = 4
           Then
        // Set all to 0 for gaze data is invalid
        ElseIf ValidityLeftEye = 2 AndAlso      ValidityRightEye =
           2 Then
        // Set all to 0 for gaze data is invalid
        ElseIf ValidityLeftEye = 1 AndAlso      ValidityRightEye =
           3 Then
        // Set gazePos is of left eye, pupildiaX is of
                // left eye and pupildiaY is null
        ElseIf ValidityLeftEye = 3 AndAlso      ValidityRightEye =
           1 Then
        // Set gazePos is of right eye, pupildiaX is
                // null and pupilDiaY is of right eye
        ElseIf ValidityLeftEye = 0 AndAlso      ValidityRightEye =
           4 Then
        // the same as 1 & 3 case
        ElseIf ValidityLeftEye = 4 AndAlso      ValidityRightEye =
           0 Then
        // the same as 3 & 1 case
    Else
        // set everthing null
    End If
        // Raise the gaze data changed event
End Sub
```

The new raw gaze data will be written into the database for further analysis and visual-

ization. The application then immediately calculates gaze fixations according to the current settings.

4.3.5.4 Namespace „Module.Database"

With the Database module, the user can manually views and revises all tables which are associated with a project. These tables are: Subjects table which holds information about test participants, SubjectParameters table which holds extra, user-defined parameters associated with a participant, Trial and TrialEvent table which hold information about testss and events that occurred during a test, GazeFixations table which holds information about fixations of eye gaze during a test, along with RawData tables which contains gaze data received from the eye tracker. This module enables user to review and revise each single entry. The data by default is filtered by participant and test, in order to avoid long loading time in large tables. The main component of this module is the DataGridViews, which is used to display and revise data from the database. Each of which is bound to a DataSource table in the database.

If the user clicks on a participant in the SubjectDataGridView, we load the gaze data into RawDataGridView with no filter set.

```
Private Sub PopulateRawDataFromCurrentSubject()
    // Set the selected subject to be the first
        // subject on the list if no subject is chosen
         Dim currentRow As DataGridViewRow =
        Me.dgvSubjects.Rows(Me.dgvSubjects.SelectedRows(0).Index)
 Dim subjectName As String =
        currentRow.Cells("colSubjectsSubjectName").Value.ToString
 Dim table As DataTable =
                                              Queries.GetRawDataBySubject
                                                 (subjectName)
 Me.bsoRawData.DataSource = table
End If
End Sub
```

If the user selects a particular test in the TrialsDataGridView, we load the gaze data into RawDataGridView with filter set.

```
Private Sub LoadRawDataIntoDataGridView(ByVal trialTableRowIndex As
    Integer)
```

```
If Me.dgvTrials.DataSource IsNot Nothing Then
        Try
            Me.Cursor = Cursors.WaitCursor
            If Me.dgvTrials.Rows.Count > trialTableRowIndex Then
                Dim currentRow As DataGridViewRow =
                                                        Me.
                                                            dgvTrials
                                                            .Rows(
                                                            trialTableRowIndex
                                                            )
                Dim subjectName As String =
                currentRow.Cells("colTrialsSubjectName").Value.
                    ToString
                Dim trialID As Integer =
                                CInt(currentRow.Cells("
                                    colTrialsTrialID").Value)
                  Dim trialSequence As Integer =
                        CInt(currentRow.Cells("
                            colTrialsTrialSequence").Value)
                Dim table As DataTable =
        Queries.GetRawDataBySubjectAndTrialSequence(subjectName,
            trialSequence)
                    'Setup the data source
                Me.bsoRawData.DataSource = table
        Catch ex As Exception
            ExceptionMethodsModule.HandleException(ex)
        Finally
           Me.Cursor = Cursors.Default
        End Try
End If
End Sub
```

Finally, we commit the changes made by the user on the DataGridView back to the database by calling the update method on each table adapter:

```
Private Sub UpdateDatabase()
    Try
       Me.Validate()
       Me.BsoSubjects.EndEdit()
```

```
        Me.BsoSubjectParameters.EndEdit()
        Me.BsoParams.EndEdit()
          Me.BsoTrials.EndEdit()
        Me.BsoTrialEvents.EndEdit()
        Me.BsoGazeFixations.EndEdit()
         Dim dataset As DataSet1 = Document.ActiveDocument.
            DocDataSet
         Dim affectedRows As Integer =
                                                dataset.
                                                   SubjectsAdapter.
                                                   Update(dataset.
                                                   Subjects)
        // call the same update method of the adapters
                // for other tables.
                dataset.AcceptChanges()
    Catch ex As Exception
        // Process the error message
    End Try
End Sub
```

Additionally, the Database module also support import/export functionality for selected participant. This is done via a BackgroundWorker thread:

```
' Background worker thread working method for exporting the
   database      Private Sub bgwExport_DoWork(ByVal sender As Object
   ,
        ByVal e As System.ComponentModel.DoWorkEventArgs) Handles
           bgwExport.DoWork
        // Code to write database data to file.
End Sub
```

4.3.5.5 Namespace „Module.Fixations”

The Fixations module is designed to calculate, store and display the gaze fixations made by participants. The fixations information is stored in the application's database and made available to the Heatmap module for further calculation of heat maps.

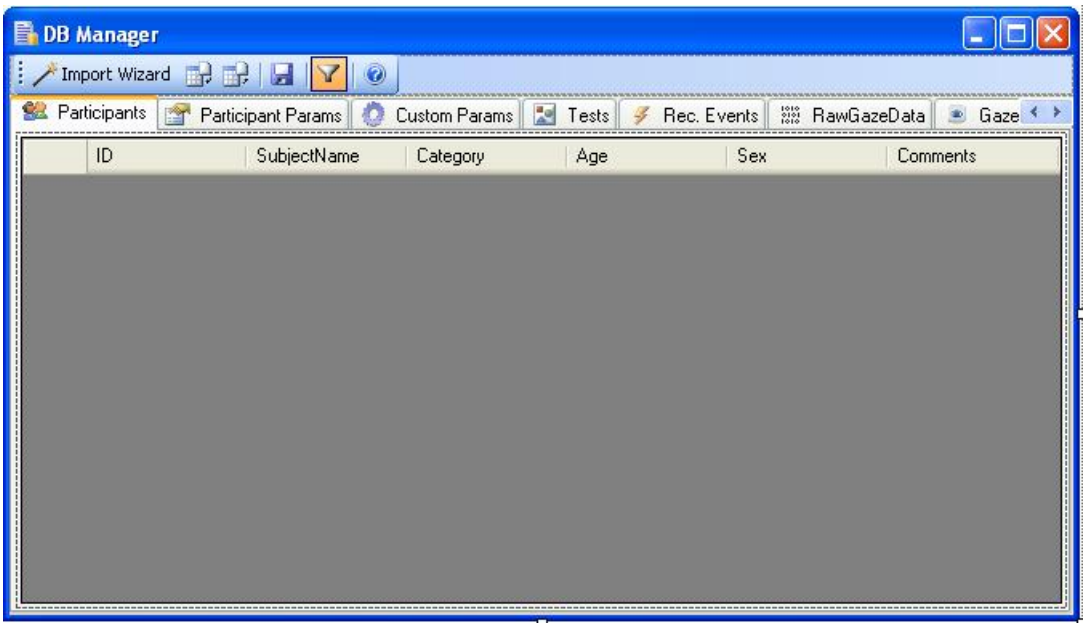


Figure 4.30: Screenshot of the Database module.

The components of this module consist of a canvas for visualization and a data table for each gaze fixation. There are five display modes available, each of them also has the option to display along fixation numbers and connections.

- Dots: fixations are drawn as filled, round dots.
- Circles: fixations are drawn as empty circles with circumference proportional to the duration of the fixation.
- Spotlight: fixations are drawn as cirles of original image of the stimulus, overlaid on grayed background, with circle's circumference proportional to the duration of the fixation.
- Heatmap: fixations are drawn as a colored heat map. For the scope of this report, heat maps are implemented as superimposed Gaussian distributions of all fixations in a recording. A description of the algorithm is discussed later.
- None: if selected then display only fixation numbers and/or connections.

The DataGridView below the module gives information about a fixation for the current recording, such as its number, belong to which participant, position, start time of the calculation. The user can filter the recording to be displayed according to its number or participant's

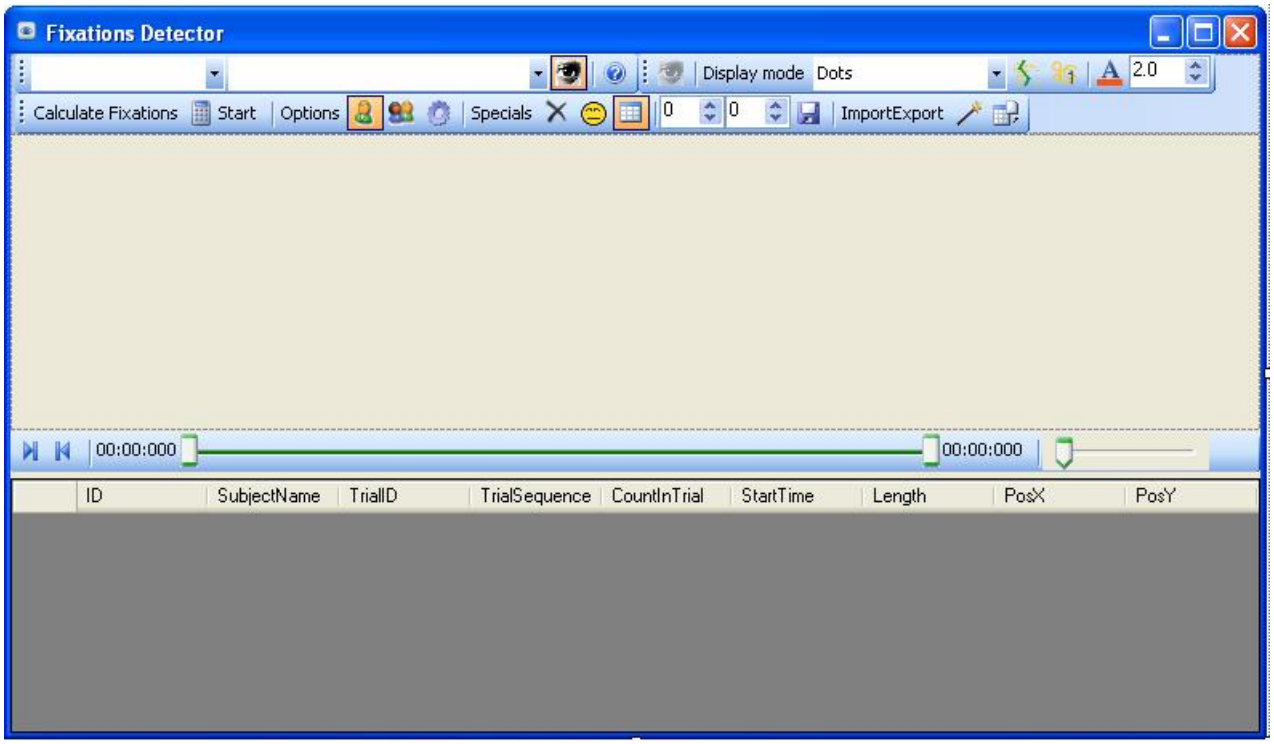


Figure 4.31: Screenshot of the Fixations module.

name, can adjust the parameters used for fixation detection in the Project Options dialog in the main window. The parameters will then be used consistently throughout the whole project. Futher options on the dialog include deletion of fixations, exclusion of a recording from further analysis by marking it as invalid (this could be the case when the calibration has a remarkable drift) and some to adjust the look and feel of the graphical elements.

The user can also filter the fixations by limiting the time range, this is done by moving the caret thumbs on the timeline.

The main component that does most of the work on this module is a BackgroundWorker thread, which calls the procedure:

```
Private Sub CalculateFixForAllSubjects(
ByVal worker As BackgroundWorker,
ByVal e As DoWorkEventArgs)
        // Get a list of all subjects
        // For each subject in the list
        ' Get trial data of current subject
        Dim trialsTable As DataTable =
Document.ActiveDocument.DocDataSet.TrialsAdapter.GetDataBySubject(
    strSubject)
Dim calculationObject As New FixationCalculation()
        ' Calculate fixations
calculationObject.CalcFixations(
SampleType.Gaze,
strSubject,
trialsTable, worker, e)
If e.Cancel Then Exit For
Next
End Sub
```

This procedure calls in the CalcFixations procedure, which is a member of the Fixation-Calculation class. The FixationCalculation class post-processes the fixation patterns detected by the FixationDetection class, which is the implementation of the algorithm by LC Technologies we have discussed in the earlier section. When new raw data is added or existing raw data is changed or deleted, a recalculation of the fixations is recommended.

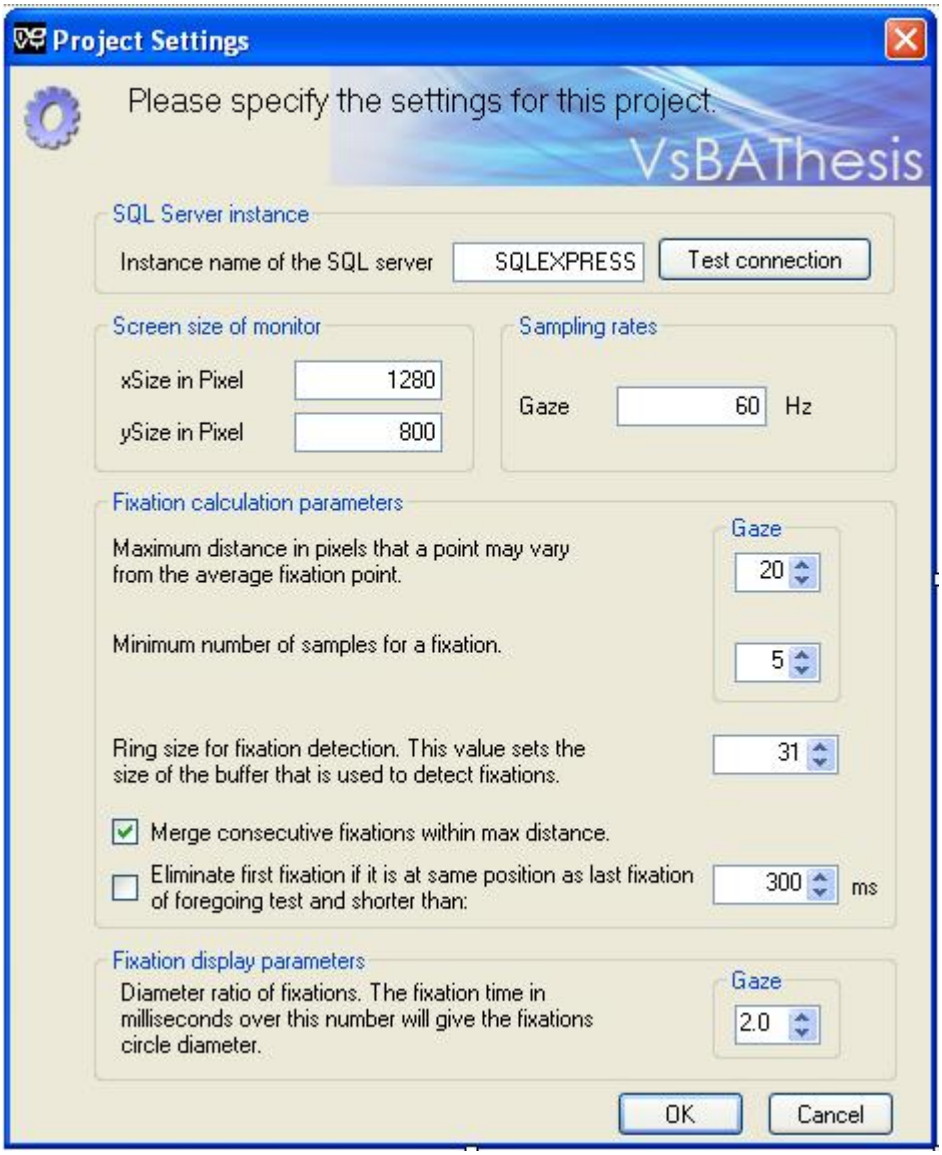


Figure 4.32: Settings that are valid for the whole project.

4.3.5.6 Namespace „Module.Heatmap"

With the Heatmap module, fixation data from selected participant can be merged and calculated. This is useful in pinpointing the regions of special attention in a stimulus. The heat map allows a quick visualization of a landscape of „visited" and „unvisited" locations on the stimulus Spakov und Miniotas (2007)

The heat map is calculated as aggregated Gaussian distributions of each fixation in a stimulus. The summed Gaussian distributions are then overlaid on the image of the original stimulus. By using a two-dimensional Gaussian kernel with an adjustable size, the distribution landscape of fixation can be smoothed at will. The Gaussian kernel is given by the following formula:

$$f(x,y) = \frac{e^{\frac{-x^2+y^2}{2\sigma^2}}}{2\Pi\sigma^2}, x, y\varepsilon[-s,s]$$

with $\mu = 0$ as mean value, standard deviation of $\sigma$and an isotropic distribution.

$\sigma$by default is set to s/5, for each fixation, the algorithm then multiplies each value of this template kernel with a factor that is proportional to the duration of the fixation. We can also call this factor the weight of the fixation.

```
' Multiply kernel with a scaling factor
Friend Shared Function MultiplyKernel(ByVal factor As Single,
ByVal size As Integer) As Single(,)
```

The multiplied kernels are then added to an array which size is equal to that of the stimulus, at the position of the associated fixations,

```
' Add a sized gaussian kernel to a distribution array of size (
    width, height) at the ' position (PosX, PosY)
Friend Shared Sub AddKernelToArray(
        ByVal distributionArray As Single(,),
                                ByVal posX As Integer,
                                ByVal posY As Integer,
                                ByVal width As Integer,
                                ByVal height As Integer,
                                ByVal size As Integer,
```

```
                                        ByVal kernel As Single(,)
                                        )
....
End Sub
```

with distributionArray is an existing gaussian distribution sum array, (posX; posY) is the position for the kernel to be added, (width, height) is the two dimensions of the distribution array, size is the size of the kernel and kernel is an array with gaussian kernels.

The whole array is then normalized to give a height or landscape map Spakov und Miniotas (2007) of size of the stimulus as the result. This map can be easily transformed into a colored heat map using a colored gradient bitmap for pixel mapping. The calculation should be handled in the background by a BackgroundWorker thread.

```
' Returns a bitmap with a gradient colored
' distributionArray of a given size
Friend Shared Function CreateHeatMap(
        ByVal heatMapBmp As PaletteBitmap,
        ByVal gradientBmp As PaletteBitmap,
        ByVal size As Size,
        ByVal distributionArray As Single(,)
        )
' This function calculates the heatmap and
' updates the view accordingly.
Private Sub DrawHeatMap(
                ByVal weightened As Boolean,
            ByVal worker AsBackgroundWorker,
                ByVal e As DoWorkEventArgs)
```

The calculation of the heat map relies on the fixation data, so it is important that by the time using the HeatMap module for visualization, the fixations table should be ready and available for use, i.e the user had to run the fixation calculation algorithm on the raw gaze data in the Fixation module beforehand.

The user can opt to generate heat maps for a particular participant or a group of participants, depending on the data available. Heat maps in general help answer sorts of question like „Where would normal users normally have the first look", or „What regions do people often ignore or got the least attention on average"

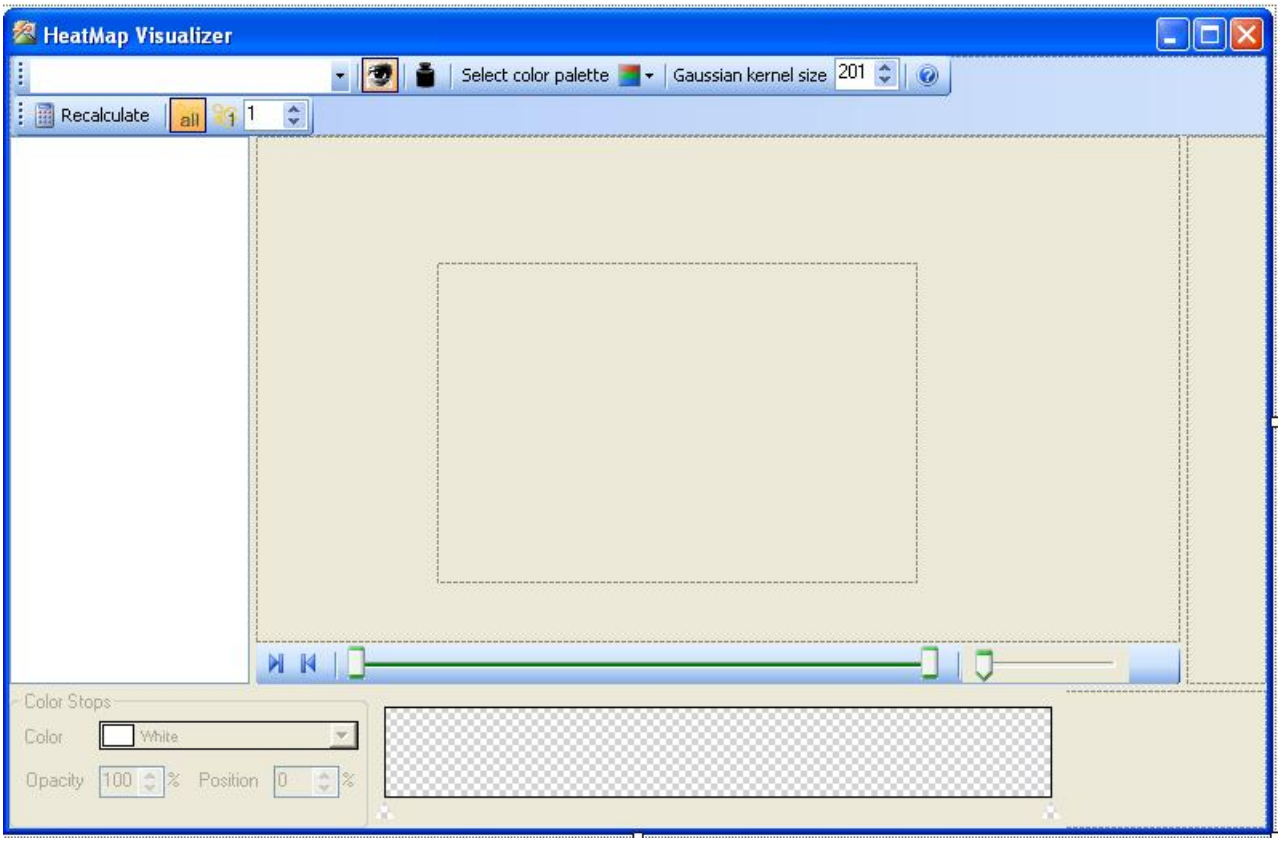


Figure 4.33: Screenshot of the Heatmap module.

4.3.5.7 Namespace „Module.ImportExport"

The application aims to support users in their workflow and also strives at improving stability and reliability by adding import/export functionality. This module is designed as a wizard-style assistant that helps users importing existing data and exporting their current work step by step.

The kind of data that the ImportExport module currently supports is raw gaze data, fixations data and project files that are created using the application, with the extension (.va). The import process involves six steps:

1. Select data file to import: this file should be in Ascii format with delimiter-separated columns.

2. Define parsing rules: these rules contain information like what the column delimiter (e.g tab, semicolon), decimal separator is and strings that should be ignored. The user can also define rules to skip lines that don't meet a specific requirement and specify whether the first row contains column names. After each change, the module updates a result table to reflect detected columns and user has the option to change column names there.

3. Map columns: if the columns from the imported file different from those in the application's database, the user will have the option to manually select the columns of the imported file from a drop-down list and map them to those of the application's database. If there is no ParticipantName column specified, the application assumes that the data is from one participant and asks for a unique participant name.

4. Define rules to distinguish different tests in a project: for example, using a table which contains the test IDs and starting time. This table can be imported in a separated file (e.g Tobii 1750 EVD file), or entered manually.

5. Define rules to read stimulus images for each test: to extract the file names of stimulus images from the imported file and assign them to the test IDs. The process is similar to that of step 4, with a preview table to reflect the parsed information. This helps user to verify that the data file is correctly parsed and information is wholely extracted.

6. Start importing: The application tries its best to parse all data in the file, including zero values and invalid values (e.g out-of-screen gaze data) and store them into database. Those zero and invalid values are handled later in fixation detection algorithms and during visualization process.

The ImportExport module can saves/loads setting files so that they can be quickly reused for further imports.

```
Private Shared Function
SerializeSettings(ByVal filePath As String) As Boolean
Private Shared Function
DeserializeSettings(ByVal filePath As
String) As Boolean
Public Function ParseFile(
        ByVal importFile As String,
      ByVal numberOfImportLines As Integer,
      ByRef columnHeaders As List(Of String)
                        ) As List(Of String())
Private Shared Function
SaveImportIntoTablesAndDB() As Boolean
Private Shared Sub
GenerateRawDataList(ByVal numberOfImportLines As Integer)
Private Shared Sub GenerateTests(
                ByVal detectonSettings As
                DetectionSettings,
                ByVal mainWindow As MainForm
                )
Private Shared Sub
GenerateParticipantAndTestList()
```

4.3.6 Implementing the Web stimulus

Applications that make use of eye trackers and deal with eye gaze data often have various types of stimulus available to choose from. Amongst them are instruction stimulus, rich text stimulus, image stimulus, flash, sound, video stimulus. . . . For the scope of this report, the application offers the Web stimulus, which is one of the most popular media types that are used in usability studies.

The core of the web stimulus is a browser object. For this, the application makes use of the web browser control which is part of the .NET controls. This browser object is wrapped by the serializable class VGBrowser. We define a web stimulus as a stimulus which contains a web browser object. So for each new web stimulus instance, we add a web browser object to the ActiveX collection of that stimulus

```
Private Function CreateWebStimulus(
ByVal browserStimu As BrowserTreeNode) As Slide
Dim stimulus As Slide =
DirectCast(browserStimu.Slide.Clone, Slide)
stimulus.VGStimuli.Clear()
Dim browser As New VGBrowser(
ShapeDrawAction.None, _
browserStimu.OriginURL, _
Pens.Black,_
Brushes.Black, _
SystemFonts.DefaultFont, _
Color.Black, _
PointF.Empty, _
Document.ActiveDocument.PresentationSize, _
VGStyleGroup.ACTIVEX, _
browserStimu.Name, _
String.Empty)
stimulus.ActiveXStimuli.Add(browser)
Return stimulus
End Function
```

We then define the BrowserTreeNode class as a node in the tree view hierarchy to wrap the web browser stimulus. This class inherits from the SlideshowTreeNode class, which in turns inherits from the base class TreeNode, and is XML-Serializable.

```
<Serializable()>
Public Class BrowserTreeNode
        Inherits SlideshowTreeNode
        Public Sub New()
        Me.Name = "-1"
        Me.Text = "Webpage"
        Me.Category = "Webpage"
        Me.OriginURL = "about:blank"
        Me.ImageKey = "Browser"
        Me.BrowseDepth = 0
    End Sub
....
```

This is the type that the Web Element dialog returns when the user has completed adding and setting up a web stimulus.

One of the problem with the web stimulus is moving content, identified as scrolling compensation. Often web pages are long and have contents that are off-screen. Scrolling makes the data on the web page become a literal moving target and the gaze position of the test subject no longer absolute, but becomes relative.

Because of this, when the test subject interact with a web page, it is not sufficient to merely store the relative offset of the gazepoint in the browser window. If the test subject is looking at the web page, then the page may have been scrolled. Storing the gazepoint with no indication of where the browser is currently scrolled to is useless, for the gaze data would not be able to actually identify where the subject was looking. Lankford (2000)

To compensate for scrolling, the application communicates with the web browser and obtain the current scroll position.

```
Dim htmlElement As HtmlElement =
browserControl.Document.GetElementsByTagName(
"HTML")(0)
        Dim bodyElement As HtmlElement =
                        browserControl.Document.Body
        Dim scrollTop As Integer =
                        If(htmlElement.ScrollTop >
                                bodyElement.ScrollTop, _
                    htmlElement.ScrollTop,
                                        bodyElement.ScrollTop)
        Dim scrollLeft As Integer =
                        If(htmlElement.ScrollLeft >
                                bodyElement.ScrollLeft, _

                                    htmlElement.ScrollLeft,
                                        bodyElement.ScrollLeft)
```

When the user scrolls the web page, the scroll event handler records this as a web browser event, which we call a scroll event and send it to the recorder with scroll offsets and a timestamp to mark it. When the recorder module receives new gaze data, then it first scales the gaze point position to screen coordinate - for Tobii eye trackers normalize gaze point position as values between 0 and 1 - and then adds optional scale offsets.

```
If newRawData.GazePosX IsNot Nothing Then
    newRawData.GazePosX =
        newRawData.GazePosX * Me.xResolution +
                                        Me.xScrollOffset
End If
If newRawData.GazePosY IsNot Nothing Then
    newRawData.GazePosY =
        newRawData.GazePosY * Me.yResolution +
                                        Me.yScrollOffset
End If
```

This allows us to determine whether or not a participant did look at a particular region on the web page regardless of where the web page was scrolled during recording or analysis. The compensation process works behind the scene, thus is hidden from the user. No intervention from user is needed.

4.3.7 Exception handling

The exception handling mechanism for the whole application is centralized in the ExceptionMethodsModule module. This module contains static methods that will deal with runtime errors occurred throughout the application during run by showing various error dialogs with details of the error (e.g error source, target site, stack trace...) and writing the log file.

```
' Handles the exception by showing a dialog with error message and
   logs to file.
Public Sub HandleException(ByVal e As Exception)
' Handles the exception and logs to file but not showing any dialog
Public Sub HandleExceptionSilent(ByVal e As Exception)
```

Chapter 5

Application Testing

5.1 Testing functional requirements

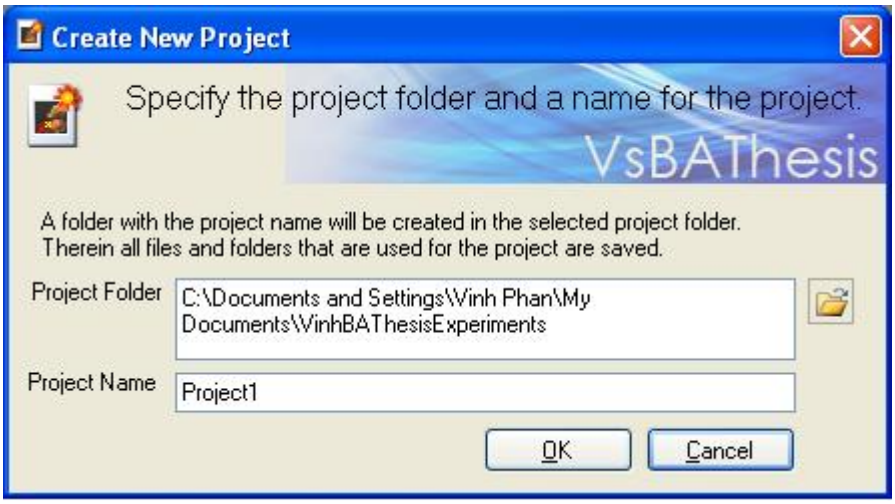
5.1.1 Test case 1: Create a new experiment

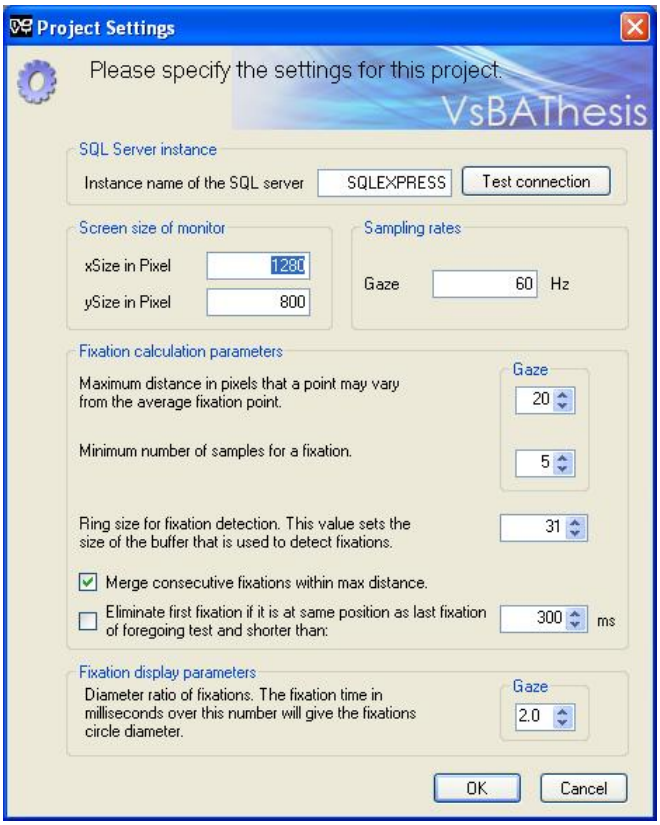
The purpose of this test is to ascertain that the application handles all the work upon creating a new project correctly, these works include creating project files and folders in the application's directory, moving the database files into the project folder and then saving the project's settings to project file.

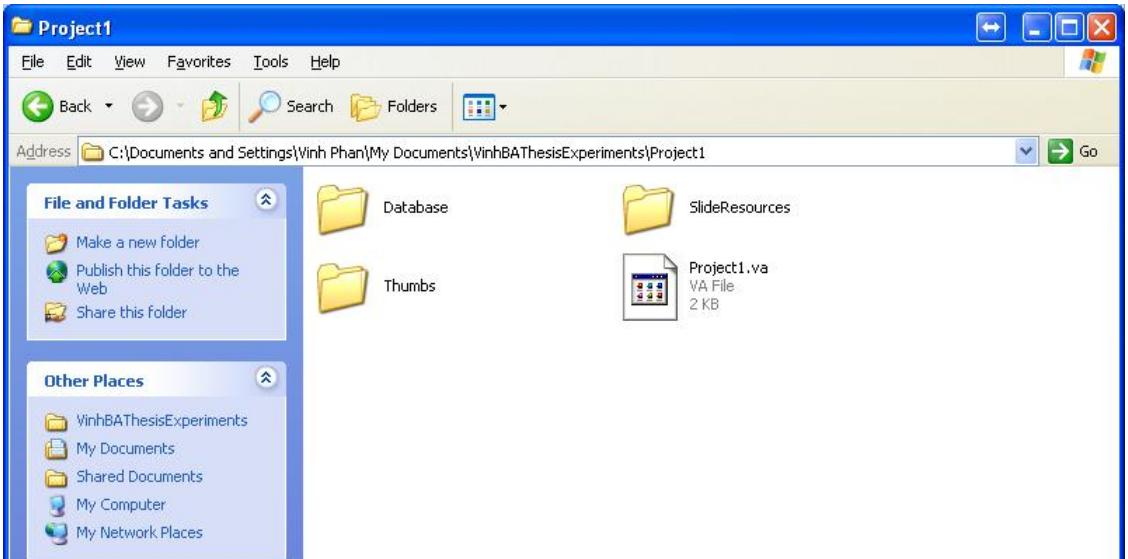
Expected result would be:

- a call to Document.ActiveDocument and Document.ActiveDocument.ExperimentSettings will not return Null.
- a folder named VBAThesisExperiments in the MyDocuments folder on the user's machine,
- inside this folder, for each of the newly created project: a folder named <ProjectName>, which contains the following subfolders: Database, Slideresources and Thumbs, together with the <ProjectName>.va project file.
- Inside the Database folder would be two database files: <ProjectName>.mdf and <ProjectName>_log.ldf.

The test is executed by starting the application, go through the steps necessary to create a new project and then check for the expected results on the file system. The result would be as depicted in the snapshots in Figure 5.1


Create New Project
Specify the project folder and a name for the project.
VsBAThesis
A folder with the project name will be created in the selected project folder.
Therein all files and folders that are used for the project are saved.
Project Folder
C:\Documents and Settings\Vinh Phan\My Documents\VinhBAThesisExperiments
Project Name
Project1
OK
Cancel


Project Settings
Please specify the settings for this project.
VsBAThesis
SQL Server instance
Instance name of the SQL server
SQLEXPRESS
Test connection
Screen size of monitor
xSize in Pixel
1280
ySize in Pixel
800
Sampling rates
Gaze
60
Hz
Fixation calculation parameters
Maximum distance in pixels that a point may vary from the average fixation point.
Gaze
20
Minimum number of samples for a fixation.
5
Ring size for fixation detection. This value sets the size of the buffer that is used to detect fixations.
31
Merge consecutive fixations within max distance.
Eliminate first fixation if it is at same position as last fixation of foregoing test and shorter than:
300
ms
Fixation display parameters
Diameter ratio of fixations. The fixation time in milliseconds over this number will give the fixations circle diameter.
Gaze
2.0
OK
Cancel


Project1
File Edit View Favorites Tools Help
Back
Search
Folders
Address
C:\Documents and Settings\Vinh Phan\My Documents\VinhBAThesisExperiments\Project1
Go
File and Folder Tasks
Make a new folder
Publish this folder to the Web
Share this folder
Other Places
VinhBAThesisExperiments
My Documents
Shared Documents
My Computer
My Network Places
Database
SlideResources
Thumbs
Project1.va
VA File
2 KB

5.1.2 Test case 2: Slideshow design and record

The purpose of this test is to assure the functionality of the TestDesignModule and RecordModule, i.e the user can create and add new stimuli into a test, then record the test with Tobii eye tracker.

Expected results would be a new trial with a Web stimulus added to it, and once the recording is finish, a raw data table filled with gaze data.

The test is executed by starting the application, creating a new project and launching the TestDesignModule. After adding and configuring the stimuli, click on Save to finish designing the slideshow.

Then launch the RecordModule and follow the four-steps recording procedure.

5.1.3 Test case 3: Heat map generation

The purpose of this test is to assure the functionality of the HeatMapModule, i.e heat maps can be generated from the recorded data.

Expected results would be heat maps with correctly overlaid fixation patterns (i.e with scrolling compensation).

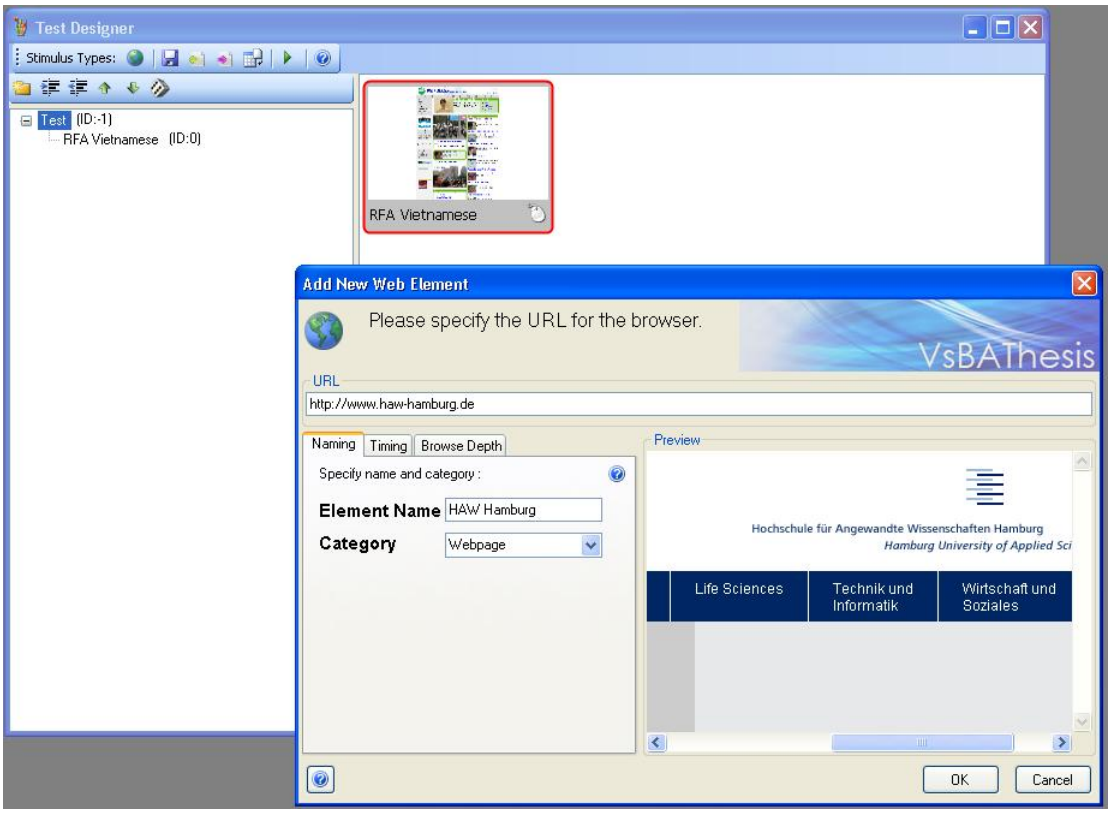
To prepare for this test, follow the Test case 2 to get gaze data, then:

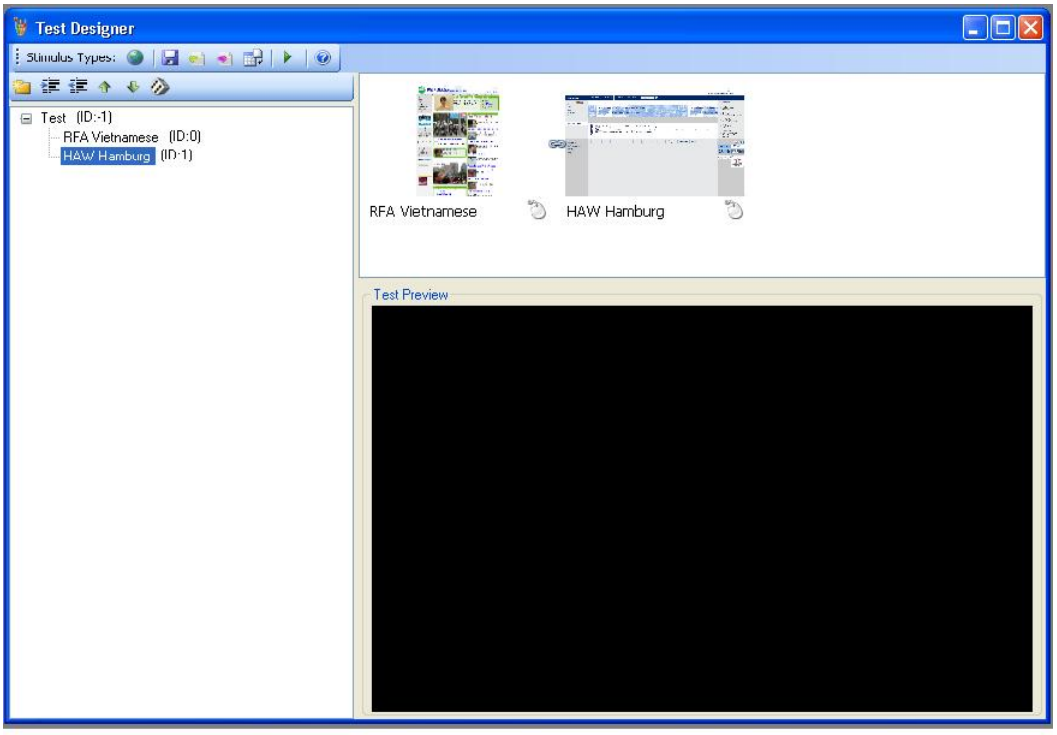
- Start the FixationsModule
- Review all the fixation data available and let fixation patterns recalculated if necessary.
- Launch the HeatMapModule, choose the participant whose data is to be visualize, choose the desired color gradient and click on Recalculate Heatmap button.

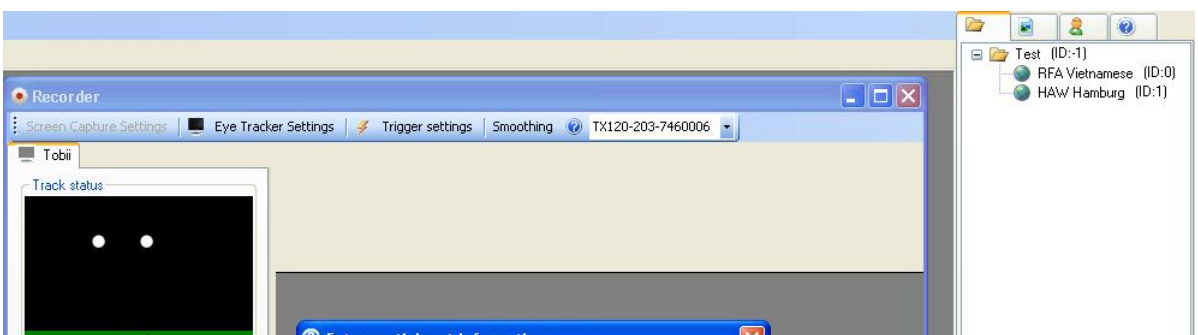
5.2 Testing non-functional requirements

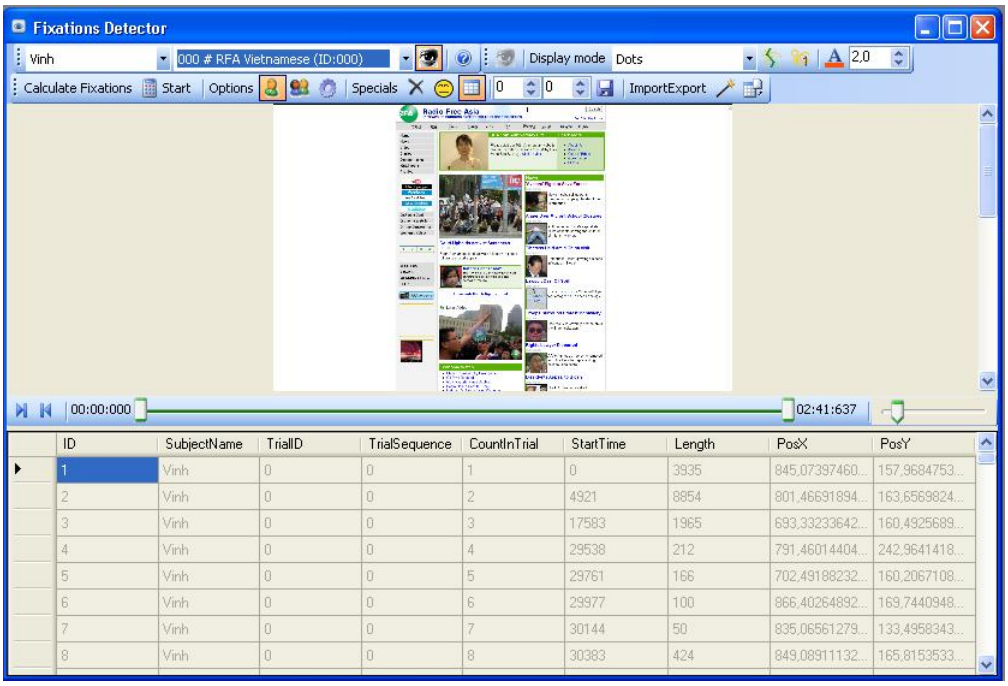
For usability testing, there must be at least one test subject available to test the software. For the scope of this report, though usability testing with a test subject was not in reality executed, below are a few considerations about the attempt to fulfil the non-functional requirements.

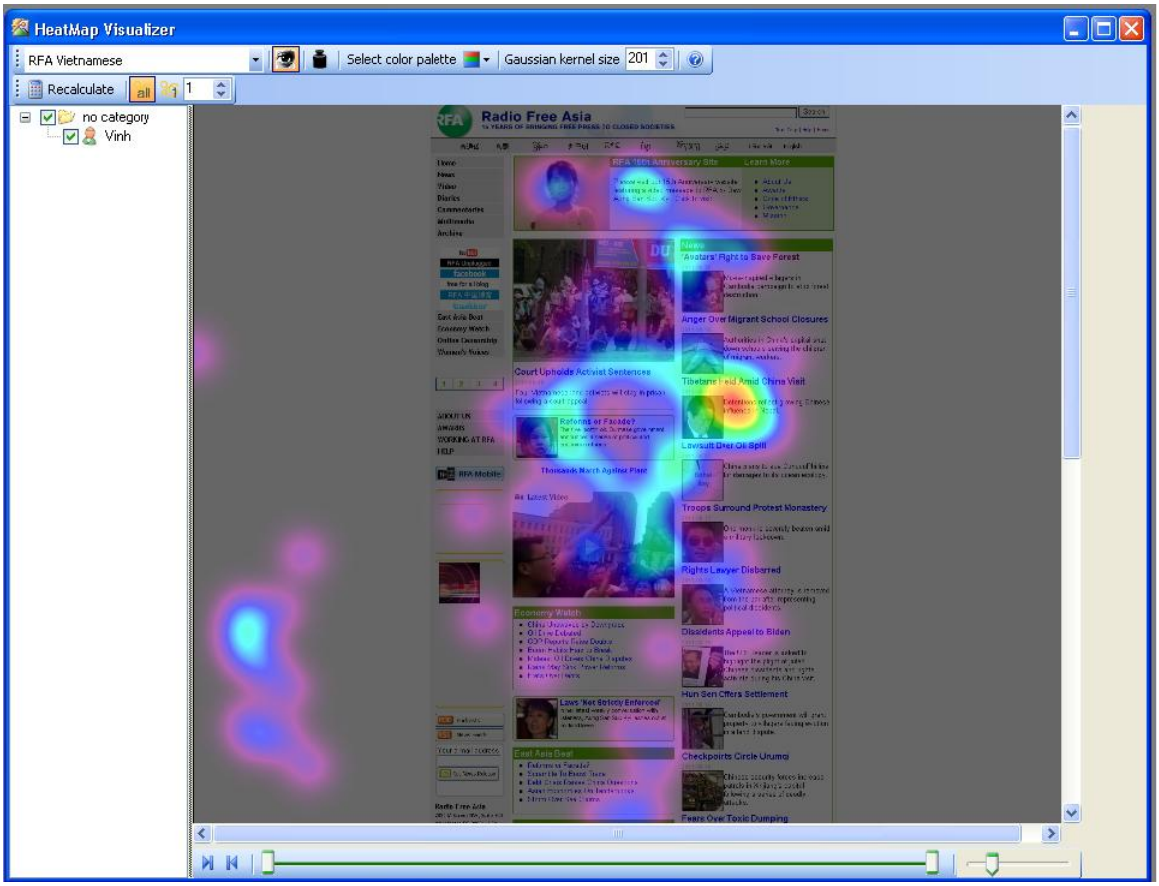
To improve usability, the application has been divided into modules and use the multi-document interface approach. For one module, such as RecordModule , a step-by-step


Test Designer
Stimulus Types:
Test (ID:-1)
RFA Vietnamese (ID:0)
RFA Vietnamese
Add New Web Element
Please specify the URL for the browser.
VsBAThesis
URL
http://www.haw-hamburg.de
Naming
Timing
Browse Depth
Specify name and category :
Element Name
HAW Hamburg
Category
Webpage
Preview
Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sci
Life Sciences
Technik und Informatik
Wirtschaft und Soziales
OK
Cancel


Test Designer
Stimulus Types:
Test (ID:-1)
RFA Vietnamese (ID:0)
HAW Hamburg (ID:1)
RFA Vietnamese
HAW Hamburg
Test Preview


Recorder
Screen Capture Settings
Eye Tracker Settings
Trigger settings
Smoothing
TX120-203-7460006
Tobii
Track status
Test (ID:-1)
RFA Vietnamese (ID:0)
HAW Hamburg (ID:1)


Fixations Detector
Vinh
000 # RFA Vietnamese (ID:000)
Display mode
Dots
Calculate Fixations
Start
Options
Specials
ImportExport
00:00:000
02:41:637
ID SubjectName TrialID TrialSequence CountInTrial StartTime Length PosX PosY
1 Vinh 0 0 1 0 3935 845,07397460... 157,9684753...
2 Vinh 0 0 2 4921 8854 801,46691894... 163,6569824...
3 Vinh 0 0 3 17583 1965 693,33233642... 160,4925689...
4 Vinh 0 0 4 29538 212 791,46014404... 242,9641418...
5 Vinh 0 0 5 29761 166 702,49188232... 160,2067108...
6 Vinh 0 0 6 29977 100 866,40264892... 169,7440948...
7 Vinh 0 0 7 30144 50 835,06561279... 133,4958343...
8 Vinh 0 0 8 30383 424 849,08911132... 165,8153533...


HeatMap Visualizer
RFA Vietnamese
Select color palette
Gaussian kernel size
201
Recalculate
all
1
no category
Vinh
Radio Free Asia


HeatMap Visualizer
HAW Hamburg
Select color palette
Gaussian kernel size
201
Recalculate
all
1
no category
Vinh
Hochschule für Angewandte Wissenschaften

design would simplify the record procedure. For another module like ImportExportModule, a wizard style interface approach would assist the user better in a complicated workflow. The TestDesignModule strives to provide the user with a WYSIWYG style interface (with a preview window upon adding Web stimulus)

The DatabaseModule is there to contribute to the usability and reliability of the application by enabling the user to revise all the data that the application uses.

There are a lot of places in the application where a hint would be useful and helps provide the user with explanation and instruction. This is achieved with the ToolTip control of the .NET framework. Also there are help documents available for the user in the Help tab of the context panel.

During the development process, static code has been separated from codes that are likely to change, if-statement and loops are kept at a minimum level to reduce the workload of the CPU, thus increasing the speed of the application.

Finally, the development environment was configured to do code analysis on every build with the Microsoft All Rules rule set.

Chapter 6

Summary

6.1 Conclusion

This custom application is a stand-alone application running on Microsoft Windows® platform and is developed with the Microsoft .NET framework. .NET is a large and powerful platform which provides various tools for different purposes, when use in combination with Visual Studio® environment enables us to create fast and powerful Windows GUI applications. The Visual Studio® has a rich set of tools and controls to support building applications with database back-end. This custom application is designed with users of the field of usability research in mind and work specifically with Tobii eye trackers, since it uses the Tobii API.

The application is GUI-based and separated into modules, according to their functionality. This approach helps making the source code more reusable through separating stable code from frequently changing code, aims at improving the maintainability and makes the application more easily to be extended. To record gaze data using Tobii eye trackers, one would use the RecordModule, to import external data for analysis and visualization / export result using the application's import-export mechanism, one would choose the ImportExport-Module. In order to work with eye gaze data, one must first understand the structure of the data, which is included in Appendix A, then know how to call into the API to capture data. The Tobii API can also be used at low level with low level programming languages, in which case, time synchronisation is also a problem to consider (see Appendix B).

Screen-based eye tracking studies have much potential of research capabilities. Usability and psychology are among research fields that utilise eye tracking the most. Even though, eye gaze tracking and studying is still in its early stage in computer science and thus poses a lot of interesting problems and challenges.

6.2 Future work

This application can be continue to be extended in various ways. First, this application is currently designed to work with Tobii eye trackers, but support for hardware trackers of other manufacturers, as well as software trackers e.g for mouse data, can also be added by implementing the ITracker interface. Additionally, the application would deliver a richer experience if more stimulus types are added. Amongst a few popular stimuli are: text, rich text, sound, movie, flash, pdf, etc. . . .

Finally, because each of the functionality of the application is on its own module, adding more modules to extend the application in term of capability is easy. A few thinkable suggestions are ReplayModule, in which the user's eye movement is recorded and the gaze patterns can be played back and exported as a movie file, or StatisticsModule in which Areas of Interest (AOIs) and empirical parameters can be defined and calculated. One can further develop his own custom statistical variables to incorporate into the application by adapting the application's source code.

Chapter 7

Glossary

Concept	Definition
custom application	an application that is developed and built on customer-side to address one's own purposes and problems and fill the gap between commercial software and personal requirements
eye tracker	a hardware device that can track human's eye movements (often using infrared)
experiment	a single work project (e.g usability study/research)
slideshow	a sequence of slides that are presented to the test subject during recording
slide	a single stimulus that is added into a trial. Represents a sequence in a trial
trial	basic unit for visualization and analysis. Each experiment can contain multiple trials. Each trial can have multiple slides.
test subject (participant)	the person that takes part in the study/research
experimenter	the person that leads/executes the study/research
heat map	a graphical representation of data where the values taken by a variable in a two-dimensional table are represented as colors
fixation	or visual fixation is the maintaining of the visual gaze on a single location
saccade	a saccade is a fast movement of an eye, head or other part of an animal's body or device
stimulus	a kind of media that will be presented to the user during recording. Can be thought as the object of the experiment.
.NET framework	a software platform from Microsoft to develop and run applications
Visual Studio	a software development kit from Microsoft
MDI	multiple document interface is an interface that acts as the parent and can contain multiple child windows
usability	usability is the ease of use and learnability of a human-made object. This object can be a software application

Appendix A

Tobii's Gaze Data

Taken from Tobii developer's guide.

This appendix gives a description of how to interpret the fields given in the gaze data. The description is valid for all Tobii APIs from where the gaze data can be received; Tobii Eye Tracker Low Level API and Tobii Eye Tracker Components API.

Error estimations; accuracy and precision of the data are not within scope of this appendix.

A.1 Data Fields

Field	Type	Description
Time stamp (second)	4 bytes signed integer	Time stamp when gaze data was sampled.
Time stamp (microsecond)	4 bytes signed integer	Microsecond fraction of time stamp when gaze data was sampled.
Left eye horizontal gaze target position	4 bytes floating point number	Gaze position related to current calibration. Increases at subjects' right.
Left eye vertical gaze target position	4 bytes floating point number	Gaze position related to current calibration. Normally increases downwards.
Left eye horizontal position as seen by the eye tracker.	4 bytes floating point number	The eye position as registered by the eye tracker. 0 is leftmost and 1 is rightmost.
Left eye vertical position as seen by the eye tracker.	4 bytes floating point number	The eye position as registered by the eye tracker. 0 is topmost and 1 is bottommost.
Left eye distance (millimeter)	4 bytes floating point number	Distance between the subjects' eye and the eye tracker.
Left eye pupil size (millimeter)	4 bytes floating point number	The length of the longest chord of the pupil ellipse.
Left eye validity code	4 bytes signed integer	An estimate of how valid all left eye data are.
Right eye horizontal gaze target position	4 bytes floating point number	Gaze position related to current calibration. Increases at subjects' right.
Right eye vertical gaze target position	4 bytes floating point number	Gaze position related to current calibration. Normally increases downwards.
Right eye horizontal position as seen by the eye tracker.	4 bytes floating point number	The eye position as registered by the eye tracker. 0 is leftmost and 1 is rightmost.
Right eye vertical position as seen by the eye tracker.	4 bytes floating point number	The eye position as registered by the eye tracker. 0 is topmost and 1 is bottommost.
Right eye distance (millimeter)	4 bytes floating point number	Distance between the subjects' eye and the eye tracker.
Right eye pupil size (millimeter)	4 bytes floating point number	The length of the longest chord of the pupil ellipse.
Right eye validity code	4 bytes signed integer	An estimate of how valid all right eye data are.

Figure A.1: Gaze data fields quick reference.

A.2 Time Stamp

This is the time when the gaze was sampled by the eye tracker. The first part is the seconds and the next field is the microseconds, ranging from 0 to 999999. For instance, if the first field is 123 and the second field is 78098, the time stamp is 123.078098 seconds.

There is no way to restart the time. The time is implicitly started first time the ttime DLL is loaded by any application. If the time stamp offset is important, store your own start time and subtract from all following time stamp.

Dependent on the time synchronize option given when eye tracker connection was made, it will be compatible with the Tobii system time either on the host running the eye tracker or on the API caller host. If they are running on the same host, this is of course not a problem.

A.3 Gaze Target Position

This is the position where the subject's gaze is at. Each eye has its own data, independent of the other eye. From the subject's perspective, it normally increases to the right and downwards. However, the range and resolution is dependent of the current calibration set. For the calibration software made by Tobii to track gaze on a computer screen for instance, the upper left corner of the screen is point (0,0) and lower right corner is (1,1). It is a good practice to follow this coordinate system when building your own calibration tool. See the sections describing the calibration process for details.

The major conditions affecting the quality of the gaze target position are:

- How good the calibration was.
- If the light conditions of target and surrounding environment are close to the conditions during calibration.
- If the gaze target is near the area that was used for calibration.
- Head and pupil motion. Less is better.

A.4 Eye Position

This is the position of the eye as seen by the eye tracker. This position has nothing to do with the gaze target position. Each eye has its own data, independent of the other eye. The upper left corner of the eye tracker sensor is (0, 0) and lower right sensor is (1, 1). Note that the values are from the eye tracker point of view. If subject is moving to the right, the values are decreasing. The purpose of this data is mainly to give a good idea of how to place the subject to make the tracking conditions at best.

A.5 Distance

This is the shortest distance between the eye tracker sensor and the eye, measured in millimeter. Each eye has its own data, independent of the other eye. The values given are best used as a set of relative values, not absolute. If absolute values are required by the application being built, have in mind that following parameters will highly affect the accuracy:

- Glasses.
- How much the cornea of the subject's eye is diverting from the assumed average human being eye cornea. It is normal that a person wearing glasses and have a cornea that diverts by a few percent will have an error of 100 mm when distance measure is 600 mm.

A.6 Pupil Size

This is the longest possible chord of the ellipse fitted to the pupil of the eye, measured in millimeter. Each eye has its own data, independent of the other eye. The values given are dependent of the distance measure, so there are the same accuracy uncertainties. Therefore, the values given are best used as a set of relative values, not absolute.

A.7 Validity Code

This is an estimate of how certain the eye tracker is that the data given for an eye really belongs to that eye. The validity code takes one of five values for each eye ranging from 0 to 4, with the following interpretation:

- 0 - The eye tracker is certain that the data for this eye is right. There is no risk of confusing data from the other eye.
- 1 - The eye tracker has only recorded one eye, and has made some assumptions and estimations regarding which is the left and which is the right eye. However, it is still very likely that the assumption made is correct. The validity code for the other eye is in this case always set to 3.
- 2 - The eye tracker has only recorded one eye, and has no way of determining which one is left eye and which one is right eye. The validity code for both eyes is set to 2.
- 3 - The eye tracker is fairly confident that the actual gaze data belongs to the other eye. The other eye will always have validity code 1.

Code	Description
0 - 0	Both eyes found. Data is valid for both eyes.
0 - 4 or 4 - 0	One eye found. Gaze data is the same for both eyes.
1 – 3 or 3 - 1	One eye found. Gaze data is the same for both eyes.
2 – 2	One eye found. Gaze data is the same for both eyes.
4 – 4	No eye found. Gaze data for both eyes are invalid.

Figure A.2: All possible combinations of validity codes.

- 4 - The actual gaze data is missing or definitely belonging to the other eye.

Hence, there are a limited number of possible combinations of validity codes for the two eyes:

A pair of gaze data with validity code 4 on both eyes, followed by a number of gaze data with validity code 0 on both eyes, is usually a sure sign of a blink.

It is recommended that the validity codes are always used for data filtering, to remove data points which are obviously incorrect. Normally, we recommend removing all data points with a validity code of 2 or higher.

Appendix B

Time Synchronization

Taken from Tobii developer's guide.

This appendix will explain how to use the eye tracker time synchronization feature and briefly describe why time synchronization and other time issues are a problem.

Implementation details are targeting the Tobii Eye Tracker Low Level API user, but may give the Tobii Eye Tracker Components API user a good understanding of how things work as well.

B.1 Background

There are several APIs in Windows that deals with time. In practice there are only two underlying methods used. One is relies on the onboard system clock. The access of this clock implicitly depends on the operating system context switching interrupt interval, which is normally 10 ms – 15 ms for Windows 2000 and Windows XP. The time functions using this method are of high accuracy and low precision kind, which means the average of a set of time stamps are ok, but each time stamp can be very inaccurate. Examples of functions that fall into this first category are; GetTickCount, timeGetTime and GetSystemTimeAsFileTime. Some of them return a millisecond or even 100-nanoseconds fraction, which is pointless when comparing two time stamps for instance. There are also a hardware dependent drift against the real time. This drift is normally small and many operating systems are set up to compensate for this drift by synchronization services. WinXP, for instance, has this synchronization built in.

The second method relies on a hardware high resolution counter found in many, but not all, Intel compatible processors since a couple of years. The benefit of this counter is the microsecond precision. However, this counter suffers from a high hardware dependent drift

against both the system clock and the real time. It also may change frequency if the system is set to hibernation or standby, and it has a known tendency to occasionally make a counter jump on some buggy hardware platforms.

No matter what source of time chosen, there will be two synchronization issues; first the one between software and real time, second the one between applications running on different hosts.

B.2 Selected Methods

Since the time precision between two gaze data points is more important than minimizing the real world time drift, the method chosen for the Tobii eye trackers is to use the processor high resolution counter. The Window API functions used are QueryPerformanceCounter and QueryPerformanceFrequency, which wraps the processor instruction rdtsc. See the Microsoft and Intel documentation for details.

The real time synchronization is ignored to avoid introducing possible error causing complexity to the system. It is more important to be able to synchronize different applications possible running on different hosts and to be able to compare time stamp, with high precision and accuracy.

For synchronization, there is a Tobii proprietary communication protocol implemented that is running in as a background thread taking care of the offset error and compensates for time drift between hosts. The synchronization may be shared among all processes and threads on a host that are connected to the same TETServer and is using the same ttime DLL.

The background synchronization is dependent on a good communication channel between the server and host. Therefore channels like modem or an Ethernet with high network load will likely give bad accuracy of the output time stamp data.

B.3 Available Synchronization Options

When applications using the tet API and the ttime API are running on another host than the Tobii Eye Tracker Server (TETServer) and gaze data time stamps must be matched with time stamp on the client side host, there is an option available to use the background time synchronization.

This option is selected by passing a parameter to the Tobii Eye Tracker Low Level API function call Tet_Connect. Three modes are available:

Tet_Connect Option	Value	Description
TET_SYNC_NONE	1	No background synchronization. The server (TETServer) host time base will be used to time stamp the gaze data. The call TT_GetServerTimeStamp will be unavailable.
TET_SYNC_SERVER	2	Background synchronization on. The server (TETServer) host time base will be used to time stamp the gaze data. The call TT_GetServerTimeStamp will be available in same thread as Tet_Connect..
TET_SYNC_LOCAL	3	Background synchronization on. The server (client (ttime/tet user) host time base will be used to time stamp the gaze data. TT_GetServerTimeStamp will be available in same thread as Tet_Connect..

Figure B.1: Synchronization modes.

Hence, the modes will have affect on following time stamps within the same thread that called Tet_Connect:

Time Stamp Source	Description
<Gaze data time stamp>	Passed to the gaze data callback function. Local time stamp base if TET_SYNC_LOCAL is chosen, else server time stamp base.
TT_GetLocalTimeStamp	Always local time stamp base.
TT_GetServerTimeStamp	Server time stamp. This function call is only available if calling thread has made a successful call to Tet_Connect using the TET_SYNC_SERVER or TET_SYNC_LOCAL parameter.

Figure B.2: Synchronization modes.

Note that this is not applicable for the Tobii Eye Tracker Components API, where there is only one call: GetTimeStamp, that returns the right time stamp dependent of the mode selected when the Connect call was made.

B.4 What Synchronization To Be Used

The system setup regarding time synchronization can be divided into following three categories:

- The first category is where background synchronization is not needed. This may happen when running applications on different hosts and none of the applications bother if time stamps are compatible or not. Another situation that falls into this category is if all applications, including the TETServer is running on the very same host. In this case, there is a perfect synchronization by default. Any process or thread, using the same ttime DLL, shares the same time. In either one of these two cases call Tet_Connect with TET_SYNC_NONE and use TT_GetLocalTimeStamp on the client host. See the example in figure below.

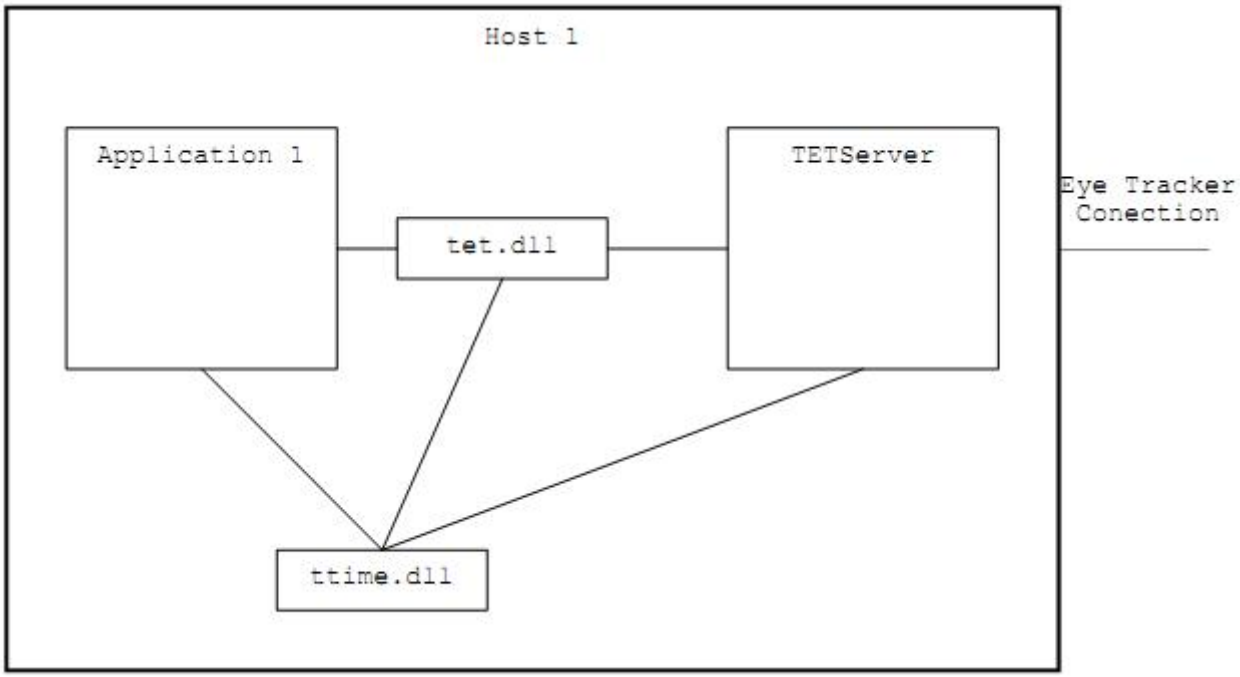


Figure B.3: No background time synchronization. Application 1 and TETServer share the same ttime the same time stamp by default.

- Synchronization is required and there are different threads or processes on the same client host that needs the same time base. Call Tet_Connect with TET_SYNC_LOCAL and TT_GetLocalTimeStamp on the client host. Any process or thread, using the same ttime DLL, shares the same time compatible with the gaze data time stamp. See the example in figure below.

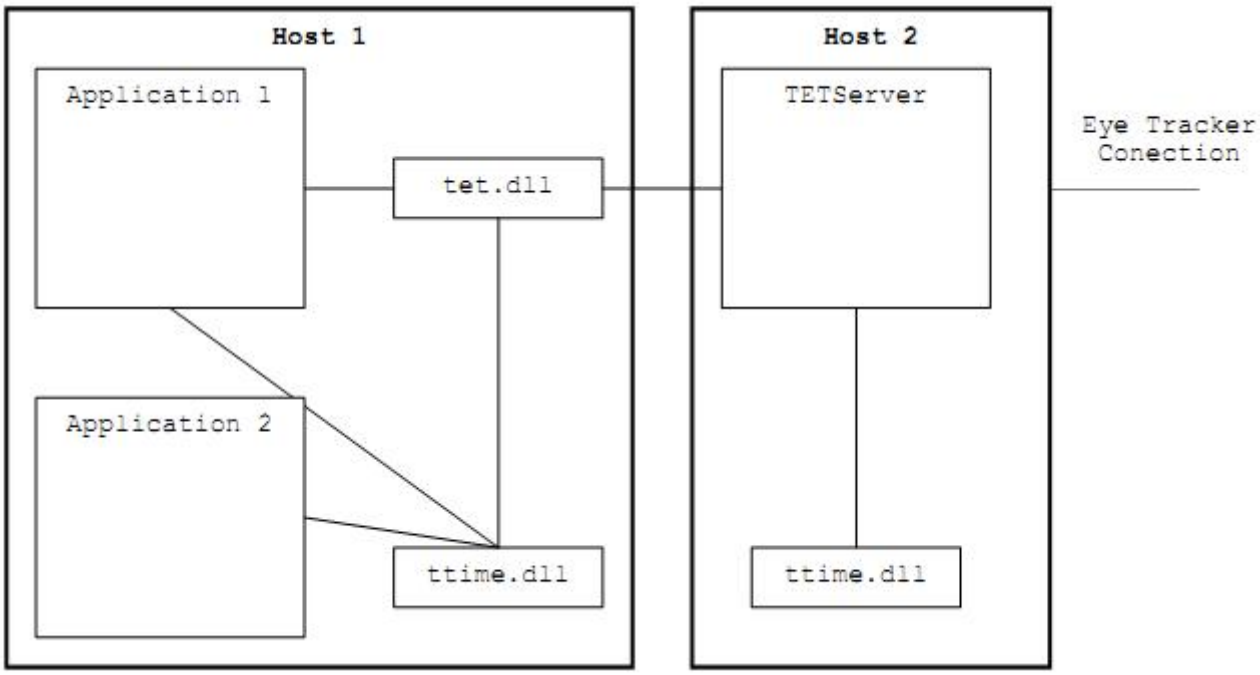


Figure B.4: Synchronization is required and Application 2 is not connected to the TETServer but needs to use compatible time stamp format. The synchronization to use on Host 1 is TET_SYNC_LOCAL.

- Synchronization is required and processes that needs to share the same time base is running on different hosts. Call Tet_Connect with TET_SYNC_SERVER and TT_GetServerTimeStamp on the client host. Only processes and threads that are connected to the same TETServer share the same time. That is, a thread calling TT_GetServerTimeStamp must have made a successful Tet_Connect with the TET_SYNC_SERVER parameter. See the example in figure below.

  Warning! Be aware of that the threads within a process have their own time context. This means that if one thread is connected to a TETServer using the TET_SYNC_SERVER and another one is not, they do not have compatible time stamps (if the server is running on another host). The figure above is still valid if Application 2 and Application 4 is threads within Application 1 and Application 3 respectively, for instance.

  Warning! Some hardware platforms can never run in a hibernation mode or system sleep mode, since they alter the way the hardware behind the Tobii time stamps runs.

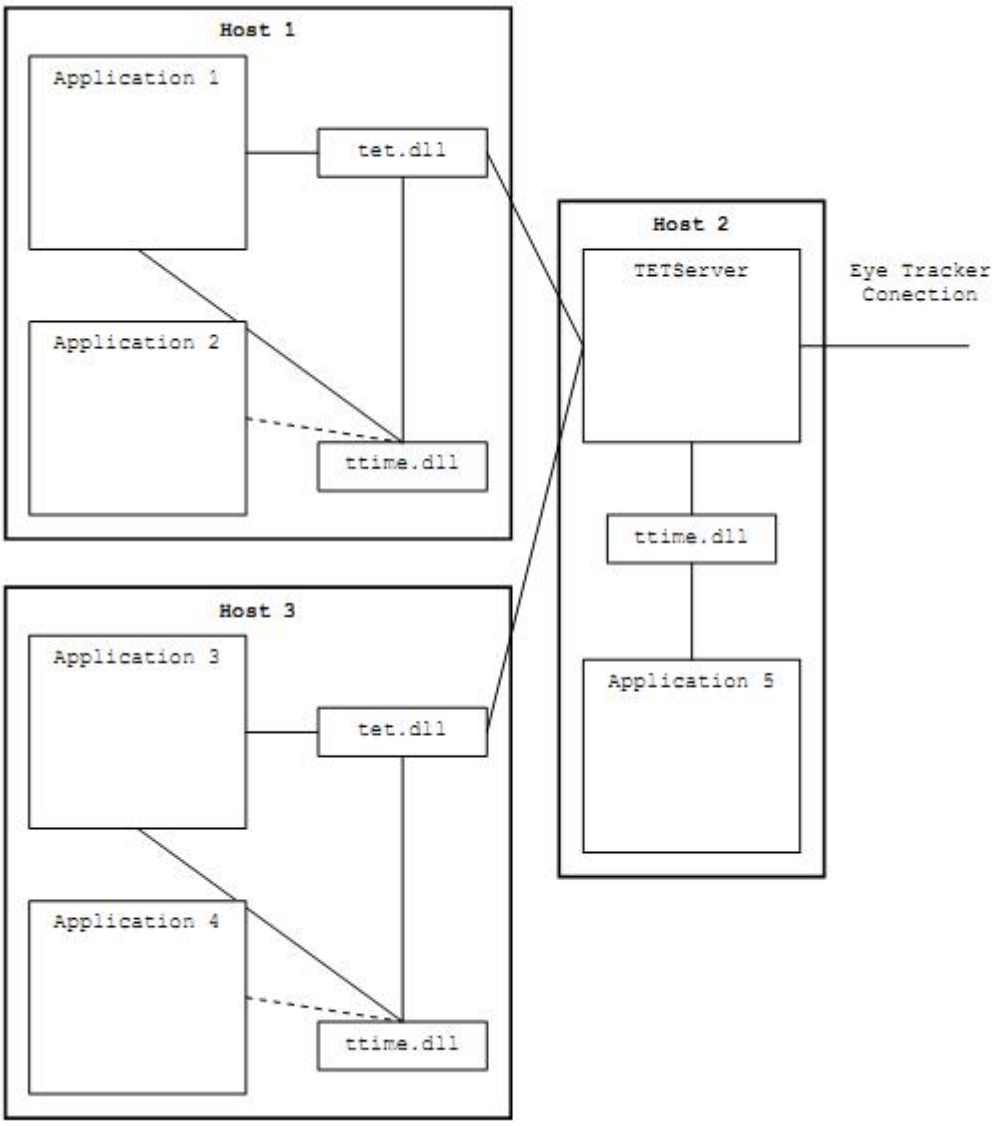


Figure B.5: Synchronization is required and Application 1 , Application 3 and Application 5 have compatible time stamp. The synchronization to use on Host 1 and Host 2 is TET_SYNC_SERVER. Note that Application 2 and Application 4 are not Connected to the server and are not compatible with any other application in this figure.

Appendix C

Finding Available Trackers On The Network

Taken from Tobii developer's guide.

This appendix will explain how to use the eye tracker time synchronization feature and briefly describe why time synchronization and other time issues are a problem.

Implementation details are targeting the Tobii Eye Tracker Low Level API user, but may give the Tobii Eye Tracker Components API user a good understanding of how things work as well.

C.1 Introduction

Tobii eyetracker generation TX (e.g. Tobii T60) introduced a new level of plug-and-play eyetracker experience. Once plugged in the eyetracker is automatically detected and can be presented to the user in a list to choose from. This removes the burden from the user to manually enter an IP address. Access to this functionality for third party programs is available through both the Tobii Eyetracking Low Level SDK (tet.dll) and the Tobii Eyetracking High Level SDK (TetComp). The interfaces used to access this functionality are called "Eyetracker Browser interface".

C.2 Redistribution Considerations

The document titled "Tobii SDK Redistribution Manual" (redist-manual.pdf) lists the special steps which need to be taken when redistributing applications using the browser interfaces.

C.3 Using the Eyetracer Browser Interface

In order to present your user with a list of available eyetrackers your application must start the Browser service. It is recommended that the service be running for as long as possible during your application

Appendix D

CD-Rom Contents

Below is the list of what is included in the CD-Rom:

- Readme.txt
- Source code
- Demo project
- Installation guide

Bibliography

[Adolph 2002] ADOLPH, Steve et a.: *Patterns for effective use cases*. Addison-Wesley, 2002. – ISBN 9780201721843

[Bittner und Spence 2003] BITTNER, Kurt ; SPENCE, Ian: *Use case modelling*. Addison-Wesley, 2003. – ISBN 9780201709131

[Cornell 1998] CORNELL, Gary: *Visual Basic 6 from the Ground Up*. McGraw Hill Professional, 1998. – ISBN 0-07-882508-8

[Duchowski 2007] DUCHOWSKI, Andrew: *Eye Tracking Methodology, Theory and Practice*. Springer, 2007

[Hennessey und Duchowski 2010] HENNESSEY, Craig ; DUCHOWSKI, Andrew T.: An Open Souce Eye-gaze Interface: Expanding the Adoption of Eye-gaze in Everyday Applications / Association for Computing Machinery, Inc. 2010. – Forschungsbericht

[Lankford 2000] LANKFORD, Chris: *Gazetracker: Software designed to facilitate eye movement analysis*. 2000. – URL `http://www.cnbc.cmu.edu/~tai/readings/active_vision/gaze_tracker.pdf`

[MSDN 2011] MSDN, Microsoft: *DataSet Class*. 2011. – URL `http://msdn.microsoft.com/en-us/library/system.data.dataset.aspx`

[Nielsen 1994] NIELSEN, Jakob: *Usability Engineering*. Morgan Kaufmann, 1994. – ISBN 0-12-518406-9

[Salvucci und Goldberg 2000] SALVUCCI ; GOLDBERG: *Identifying fixations and saccades in eye-tracking protocols*. 2000. – URL `http://portal.acm.org/citation.cfm?id=355028`

[Spakov und Miniotas 2007] SPAKOV, O. ; MINIOTAS, D.: *Visualization of Eye Gaze Data using Heat Maps.* 2007. – URL `http://itc.ktu.lt/itc362/Spakov362.pdf`

[Technology 2010] TECHNOLOGY, Tobii: *Tobii Studio(tm) Brochure*. 2010. – URL http://www.tobii.com/archive/files/18996/Tobii+Studio+Brochure.pdf.aspx

[Technology 2011a] TECHNOLOGY, Tobii: *Support Page for Tobii X120 Eyetracker*. 2011. – URL http://www.tobii.com/en/analysis-and-research/global/support-and-downloads/?product=771

[Technology 2011b] TECHNOLOGY, Tobii: *Tobii Support and Downloads Page*. 2011. – URL http://www.tobii.com/en/eye-tracking-integration/global/support/

[Technology 2011c] TECHNOLOGY, Tobii: *Tobii X120 Eyetracker Product Page*. 2011. – URL http://www.tobii.com/en/analysis-and-research/global/products/hardware/tobii-x60x120-eye-tracker/

[Technology 2011d] TECHNOLOGY, Tobii: *Tobii.com*. 2011. – URL http://www.tobii.com

[Wikipedia 2011a] WIKIPEDIA: *ISO/IEC 9126*. 2011. – URL http://en.wikipedia.org/wiki/ISO/IEC_9126

[Wikipedia 2011b] WIKIPEDIA: *Multi Document Interface*. 2011. – URL http://en.wikipedia.org/wiki/Multiple_document_interface

List of Figures

List of Tables

Hiermit versichere ich, dass ich die vorliegende Arbeit im Sinne der Prüfungsordnung nach §22(4) bzw.§24(4) ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 24.02.2011 Truong Vinh Phan