



IST-2003-511598 (NoE)

COGAIN

Communication by Gaze Interaction

Network of Excellence
Information Society Technologies

D2.3 Implementation of COGAIN Gaze Tracking Standards

Due date of deliverable: 28.02.2006
Actual submission date: 08.03.2006

Start date of project: 1.9.2004

Duration: 60 months

De Montfort University

| | | |
|---|---|---|
| Project co-funded by the European Commission within the Sixth Framework Programme (2002-2006) | | |
| Dissemination Level | | |
| PU | Public | x |
| PP | Restricted to other programme participants (including the Commission Services) | |
| RE | Restricted to a group specified by the consortium (including the Commission Services) | |
| CO | Confidential, only for members of the consortium (including the Commission Services) | |

Bates, R. and Spakov, O. (2006) *D2.3 Implementation of COGAIN Gaze Tracking Standards*. Communication by Gaze Interaction (COGAIN), IST-2003-511598: Deliverable 2.3.
Available at <http://www.cogain.org/results/reports/COGAIN-D2.3.pdf>

Main authors: Richard Bates (DMU)
Oleg Spakov (UTA)

Main contributors: Howell Istance (DMU)
Päivi Majaranta (UTA)
John Paulin Hansen (ITU)
Mick Donegan (ACE)
Gintautas Daunys (SU)
Lisa Oosthuizen (ACE)

Table of Contents

| | |
|--|-----------|
| EXECUTIVE SUMMARY | 5 |
| A single gaze communication standard | 5 |
| Future COGAIN standards | 5 |
| 1 INTRODUCTION | 6 |
| 1.1 This deliverable | 6 |
| 2 THE COGAIN STANDARD DRIVER | 7 |
| 2.1 Structure of the ETSD | 7 |
| 2.2 COGAIN ETSD drivers available | 8 |
| 2.3 Installing the COGAIN ETSD | 9 |
| 2.4 Installing the COGAIN ETSD test application | 12 |
| 2.5 Using the COGAIN ETSD test application | 14 |
| 2.6 The COGAIN ETSD test application data | 21 |
| 2.7 Summary | 28 |
| 3 THE COGAIN ETSD PROGRAMMER GUIDE TO THE COM INTERFACE | 29 |
| 3.1 Enumerations | 29 |
| 3.2 Structures | 32 |
| 3.3 Properties | 38 |
| 3.4 Methods | 40 |
| 3.5 Events | 46 |
| 3.6 Summary | 48 |
| 4 THE COGAIN STANDARD DRIVER PLAYING EYE CHESS | 49 |
| 4.1 The chess program | 49 |
| 4.2 Using the COGAIN ETSD | 50 |
| 5 CONCLUSIONS | 54 |

List of Figures

| | |
|--|-----------|
| Figure 1. Basic structure of the COGAIN ETSD standard driver for one eye tracker | 7 |
| Figure 2. Overall structure of the COGAIN ETSD standard driver for many eye trackers | 8 |
| Figure 3. Installing the COGAIN ETSD step 1 | 9 |
| Figure 4. Installing the COGAIN ETSD step 2 | 10 |
| Figure 5. Files installed by the COGAIN ETSD main directory | 10 |
| Figure 6. Files installed by the COGAIN ETSD drivers directory | 11 |
| Figure 7. Files installed by the COGAIN ETSD API directory | 11 |
| Figure 8. Files installed by the COGAIN ETSD INI directory | 12 |
| Figure 9. Installing the COGAIN ETSD Test Application | 13 |
| Figure 10. Files installed by the COGAIN ETSD Test Application | 13 |
| Figure 11. The COGAIN ETSD Test Application | 14 |
| Figure 12. Selecting an eye tracker driver on the COGAIN ETSD Test Application step 1 | 14 |

| | |
|---|----|
| Figure 13. Selecting an eye tracker driver on the COGAIN ETSD Test Application step 2 | 15 |
| Figure 14. Selecting eye tracker driver options on the COGAIN ETSD Test Application step 1..... | 15 |
| Figure 15. Selecting eye tracker driver options on the COGAIN ETSD Test Application step 2..... | 16 |
| Figure 16. Selecting eye tracker driver options on the COGAIN ETSD Test Application step 3..... | 16 |
| Figure 17. Selecting eye tracker driver options on the COGAIN ETSD Test Application step 4..... | 17 |
| Figure 18. Selecting eye tracker driver options on the COGAIN ETSD Test Application step 5..... | 18 |
| Figure 19. Selecting eye tracker driver options on the COGAIN ETSD Test Application step 6..... | 19 |
| Figure 20. Selecting eye tracker driver options on the COGAIN ETSD Test Application step 7..... | 19 |
| Figure 21. Running the COGAIN ETSD Test Application step 1 | 20 |
| Figure 22. Running the COGAIN ETSD Test Application step 2 | 20 |
| Figure 23. Running the COGAIN ETSD Test Application step 3 | 21 |
| Figure 24. Running the COGAIN ETSD Test Application step 4 | 21 |
| Figure 25. The COGAIN ETSD Test Application data step 1..... | 22 |
| Figure 26. The COGAIN ETSD Test Application data step 2..... | 22 |
| Figure 27. The COGAIN ETSD Test Application data file 1 | 23 |
| Figure 28. The COGAIN ETSD Test Application data file 2..... | 24 |
| Figure 29. The COGAIN ETSD Test Application data file 3..... | 24 |
| Figure 30. The COGAIN ETSD Test Application data file 4..... | 27 |
| Figure 31. The 'Eye Chess' program using the COGAIN ETSD – ready to play | 49 |
| Figure 32. The 'Eye Chess' program using the COGAIN ETSD – game playing..... | 50 |
| Figure 33. The 'Eye Chess' program and configuring the COGAIN ETSD step 1 | 51 |
| Figure 34. The 'Eye Chess' program and configuring the COGAIN ETSD step 2 | 51 |
| Figure 35. The 'Eye Chess' program showing the COGAIN ETSD step 1..... | 52 |
| Figure 36. The 'Eye Chess' program showing the COGAIN ETSD step 2..... | 53 |

Executive Summary

This deliverable builds deliverables D2.1 ‘Survey of De-Facto Standards in Eye Tracking’¹ and D2.2 ‘Requirements for the Common Format of Eye Movement Data’¹. Here D2.3 ‘Implementation of COGAIN Gaze Tracking Standards’ presents the prototype COGAIN Eye-Tracking Standard Driver (COGAIN ETSD). The deliverable first introduces the structure of the driver then shows how the driver may be installed and tested using a test application. The deliverable then gives a basic programmers reference guide to the driver that allows programmers to use the driver with their own third party applications. Finally the deliverable shows the driver working with an edutainment ‘EyeChess’ application – the first application to use the driver. This deliverable is not the final outcome for the driver. It is intended that this iteration of the driver is a prototype to be downloaded from the COGAIN portal and tested by third parties. Feedback will then be gathered on corrections and extensions to the driver, and it is intended that subsequent iterations of the driver will be produced in future COGAIN work.

A single gaze communication standard

The prototype COGAIN Eye-Tracking Standard Driver (COGAIN ETSD) presented here gives a single gaze communication standard that will allow any third party application to be driven by a range of *different* eye tracking hardware systems. By using the driver, it will remove the need for any third party applications to be changed or recompiled when switching between differing eye tracking hardware systems. This represents a significant step forward in increasing the uptake of gaze communications software by opening the use of differing eye tracking systems to any third party developer in the most efficient and effective way possible. The driver may be downloaded via the COGAIN portal².

Future COGAIN standards

It is anticipated that this driver will be re-evaluated once it has undergone trials with third parties. Thus it is planned that the driver will be updated to a more formal standard once feedback has been received. This will be drawn into the future plans of COGAIN. Future standards are also planned for home and domestic environmental control, and mobility control. With the final aim of producing a defined set of gaze control standards for communication control, environmental control, and mobility control.

¹ Available from <http://www.cogain.org/results/reports/>

² Available from <http://www.cogain.org/results/>

1 Introduction

1.1 This deliverable

This deliverable builds on the two previous deliverables of WP2 ‘Standardisation’; D2.1 ‘Survey of De-Facto Standards in Eye Tracking’ and D2.2 ‘Requirements for the Common Format of Eye Movement Data’, by presenting the first version of the COGAIN Eye-Tracking Standard Driver (COGAIN ETSD)¹. The COGAIN ETSD is designed to allow any third party application, such as an on-screen gaze driven communication aid, to be driven by a range of *different* eye tracking hardware systems without the need to change the third party application *in any way* when switching between those differing eye tracking hardware systems.

This deliverable represents the first standardised driver for interchangeable or ‘plug-and-play’ eye tracking systems, and is the first major step toward practical standardisation of eye-gaze communication systems. The deliverable contains a basic high-level description of the implementation of the COGAIN ETSD, followed by a programmer reference for driver operation, which shows the structure of the COGAIN ETSD at a code level, detailing briefly the methods by which the driver presents a single unified interface to a range of differing eye tracking hardware systems. Finally, the driver is shown in operation with a test application ‘EyeChess’ that demonstrates the effectiveness of the driver.

This is the first implementation of the COGAIN ETSD, and represents a prototype form of the driver. This will be made available for download from the COGAIN portal, and it is anticipated that this will attract feedback and requests for additional functionality and alteration. It is intended that the driver is reviewed once it has been in the field and attracted feedback, with a subsequent deliverable planned that will update and formalise the driver from a prototype form, to a form suitable for standardisation.

¹ Available from <http://www.cogain.org/results/>

2 The COGAIN standard driver

2.1 Structure of the ETSD

The COGAIN ETSD produces a single commonly held standard interface by ‘wrapping’ differing manufacturers systems interfaces, and translating these differing interfaces into a single COGAIN ‘virtual’ eye tracking standard interface. This approach is described in detail in the previous deliverables D2.1 and D2.2. The ETSD is developed as a software layer that is used between the actual eye tracker driver of any given eye tracking systems hardware and any end-user third party application to provide device-independent data universal access.

The ETSD consists of a set of COM objects that on one side of the object implement the COGAIN standard interface, and on the other side of the object implement a single interface unique a given eye tracking hardware system. Each COM object has a set of supporting Dynamic Link Libraries (DLL libraries) that act as converters of the original manufacturer's Application Programming Interfaces (APIs) into the standard COGAIN API. This structure is illustrated in Figure 1.

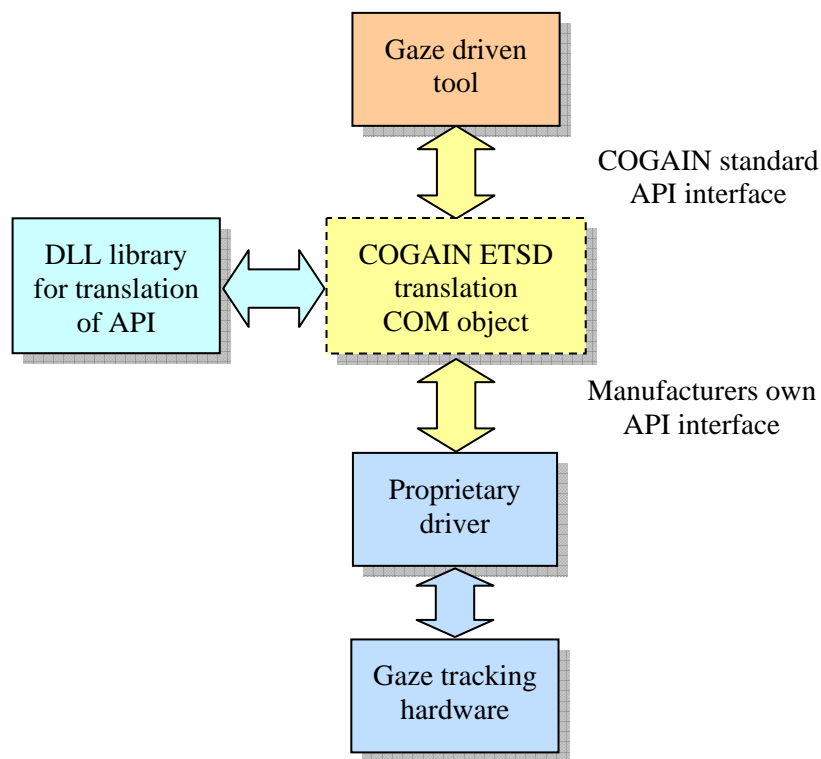


Figure 1. Basic structure of the COGAIN ETSD standard driver for one eye tracker

The benefit of using COGAIN ETSD comes from the fact that a third party end-user application implemented using ETSD can access data from a newly installed eye tracker simply by using a corresponding COGAIN ETSD translation COM object. This universal approach is illustrated in Figure 2, where a gaze driven tool needs only one standard interface (the COGAIN ETSD) to communicate with a range of eye tracking systems.

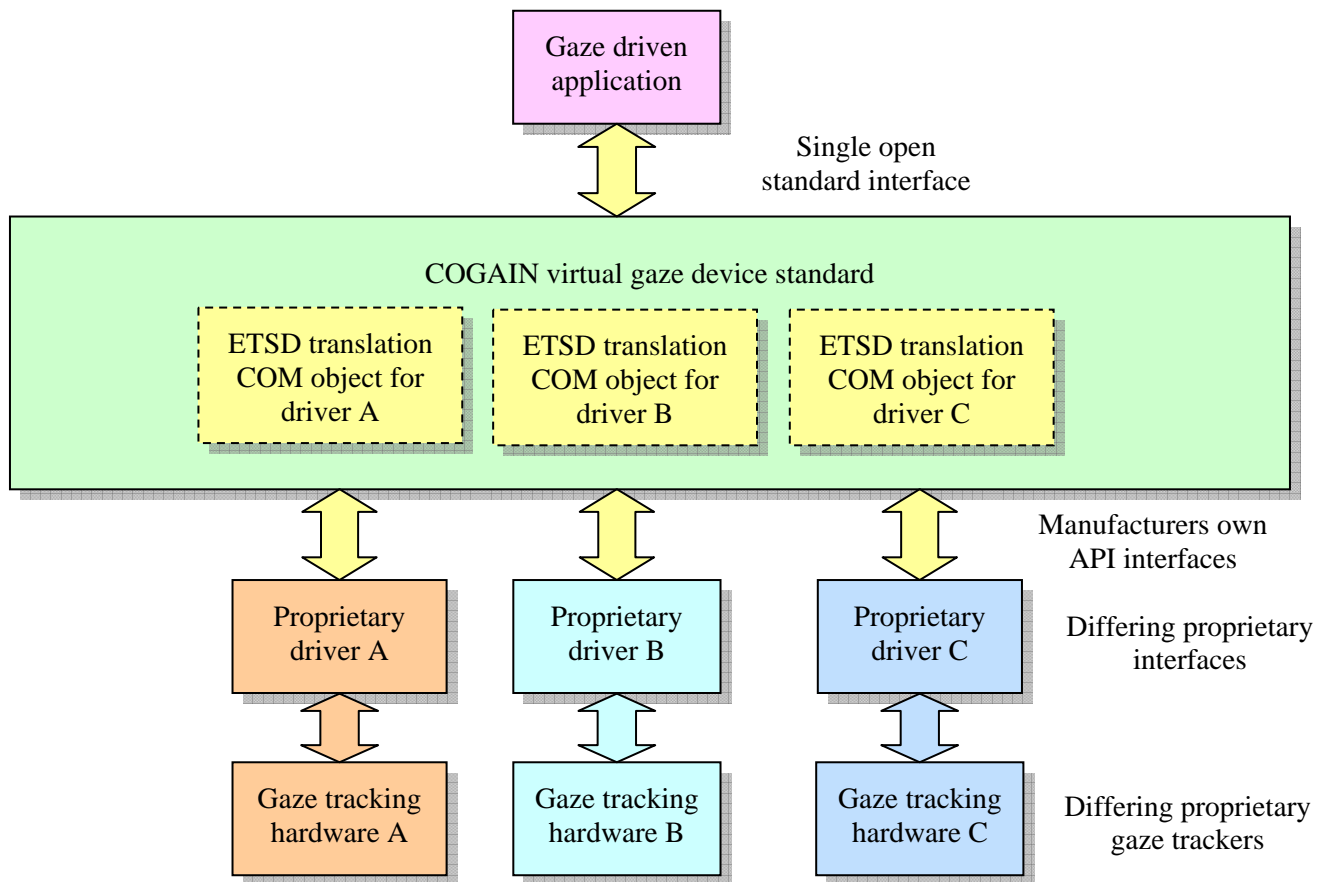


Figure 2. Overall structure of the COGAIN ETSD standard driver for many eye trackers

2.2 COGAIN ETSD drivers available

The COM object and translation approach used by the COGAIN ETSD relies on a converter being available for any eye tracking system that may be used for gaze communication, thus requiring a COGAIN driver to be written and available for every eye tracking system currently in use. However, unlike the current situation found by third party application developers where they must write a new driver *many times* for each eye tracking hardware system they may wish to use and *for every* application they may write, by using the COGAIN approach a converter only need be written *once* for each eye tracking hardware system currently in use. It is hoped that by the ending of the COGAIN project that manufacturers will be sufficiently encouraged to provide a COGAIN ETSD compliant driver as part of their own software package. In the mean-time,

COGAIN developers are implementing drivers for a range of eye tracking hardware systems. To date, three popular systems now have drivers, these are:

- SMI EyeLink
- SMI iViewX
- Tobii Technologies 1750

These drivers will be made available via the COGAIN web portal www.cogain.org, together with new drivers as they become available.

2.3 Installing the COGAIN ETSD

The COGAIN ETSD comes as a single executable file (ETUDriver_setup.exe) that can be downloaded and installed on to any Windows XP personal computer. The Windows operating system was chosen for the initial release of the driver as this is the system most used by both eye tracking hardware manufacturers and by third party gaze enabled software application developers. The installation steps are illustrated in Figures 3 to 8.

On installing the user is asked to accept the free licence from COGAIN (via contractor Oleg Spakov) for the COGAIN ETSD, Figure 3.

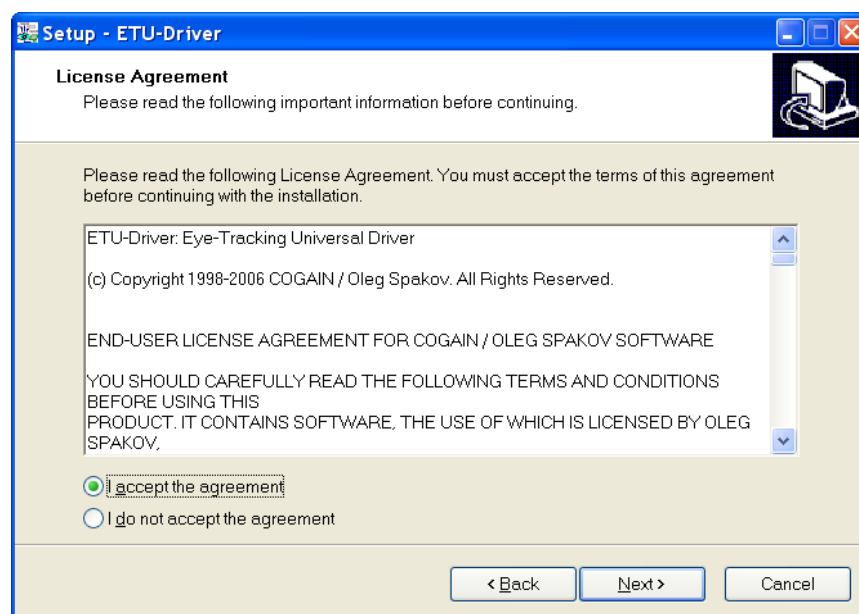


Figure 3. Installing the COGAIN ETSD step 1

The user is then asked to choose the location of the driver on their computer, Figure 4. It is recommended to accept the default location of 'ProgramFiles/ETUDriver'. This is where the driver files will be located, together with the individual converters for a range of eye tracking systems, and this is the location where third part developers may find and link to the driver.

This is the end of the installation of the driver. Installation is made as simple as possible so that users of a wide range of abilities could install the driver software easily.

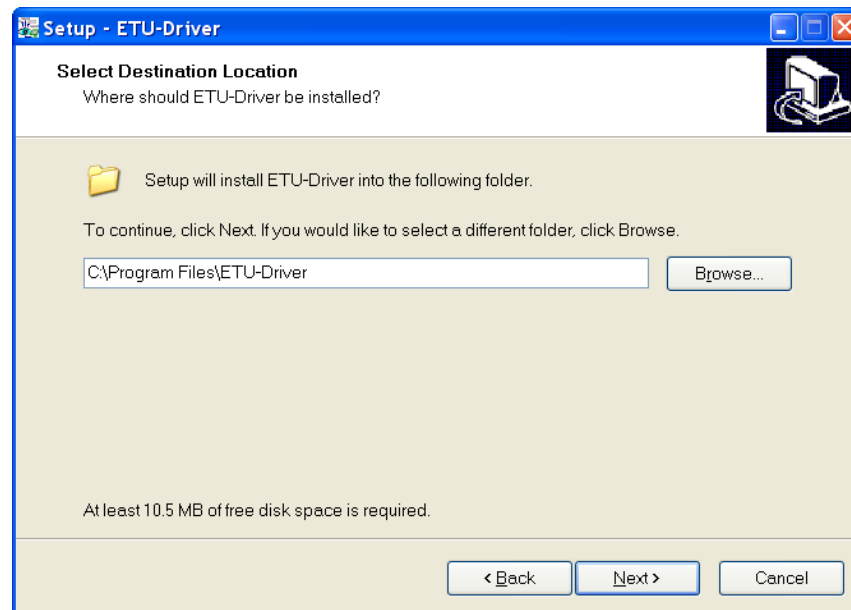


Figure 4. Installing the COGAIN ETSD step 2

Examining the file installed shows the main COGAIN ETSD interface 'ETUDriver.dll' together with a range of support files that are of use to advanced developers only, Figure 5.

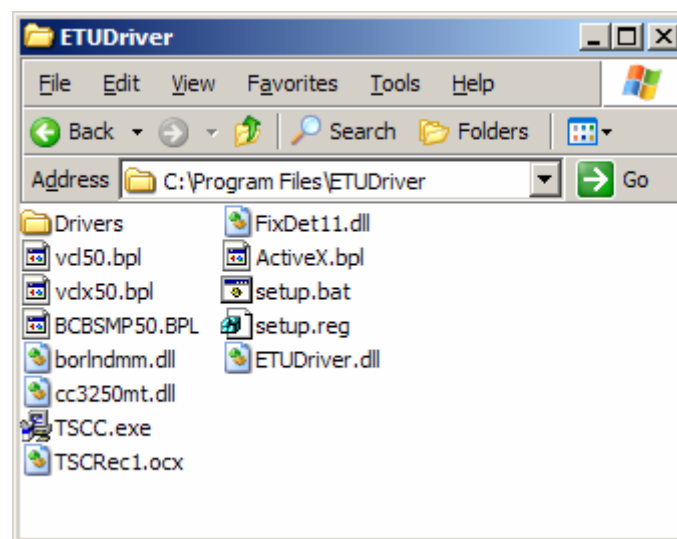


Figure 5. Files installed by the COGAIN ETSD main directory

Examining the 'Drivers' directory, Figure 6, shows the ETSD dynamic link library translator files (as illustrated in Figure 1) for the three eye tracking hardware systems currently available with the COGAIN standard interface, 'EyelinkSMI.dll', 'iViewX.dll' and 'Tobii.dll'.

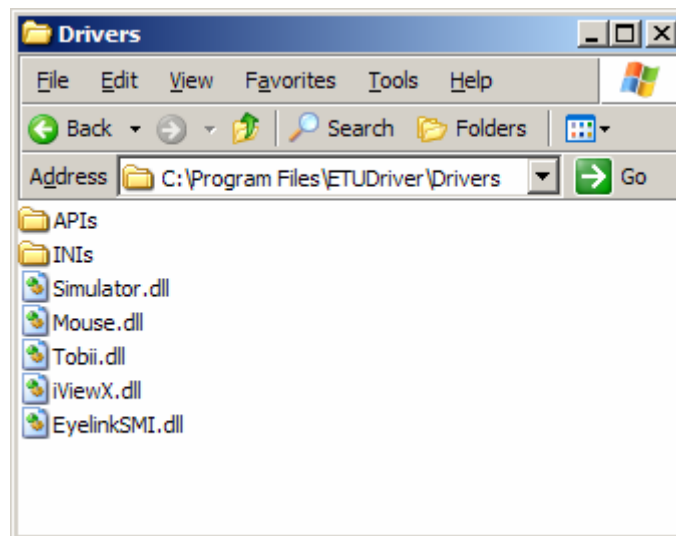


Figure 6. Files installed by the COGAIN ETSD drivers directory

Examining the 'API' directory shows the original manufacturers API dynamic link libraries that are linked to and translated by the COGAIN ETSD, Figure 7.

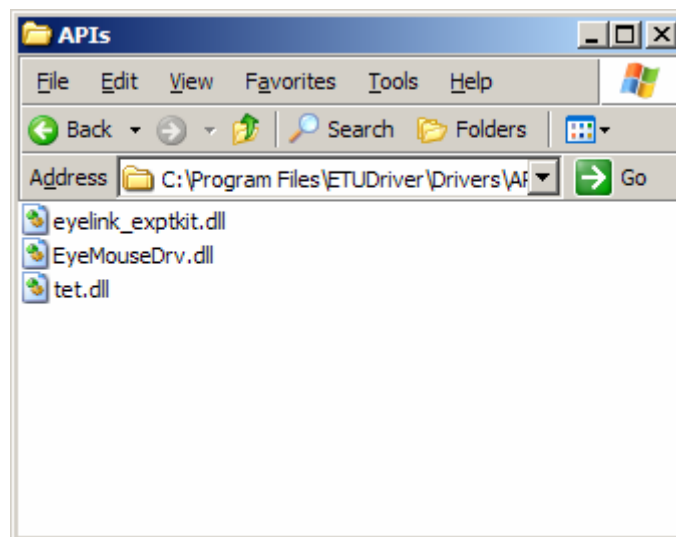


Figure 7. Files installed by the COGAIN ETSD API directory

Finally, examining the 'INI' directory, Figure 8, shows basic initialisation files that can be used by advanced developers to set manufacturers dedicated operating variables when controlling the eye tracking hardware systems. In general these need not be altered, and the contents are explained by consulting manufacturers own manuals.

This completes the installation of the COGAIN ETSD software, and the COGAIN ETSD driver interface is now available to be used by third party application developers by simply incorporating the 'ETUDriver.dll' into their application.

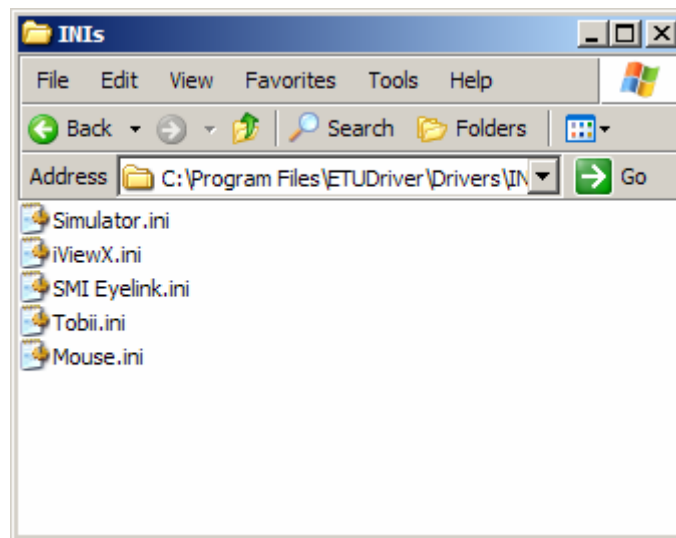


Figure 8. Files installed by the COGAIN ETSD INI directory

2.4 Installing the COGAIN ETSD test application

The COGAIN ETSD comes with a simple test application that tests each of the driver translations (see Figure 2) for the three drivers currently available. This allows developer to test the COGAIN ETSD with the three eye trackers currently supported. In addition the test application also allows simulation of an eye tracker input to the COGAIN ETSD via the desktop mouse. This is particularly useful for testing the ETSD when an eye tracker is not currently available. This is a particularly powerful feature, as it allows third party manufacturers to use the ETSD in their applications without needing actual eye tracking hardware (as the desktop mouse simulates and eye tracker connected to the system).

The test application is available as a single executable file (ETUDriverTester_setup.exe) that can be downloaded and installed. The installation steps are illustrated in Figures 9 to 10, and are started by double clicking the file (ETUDriverTester_setup.exe).

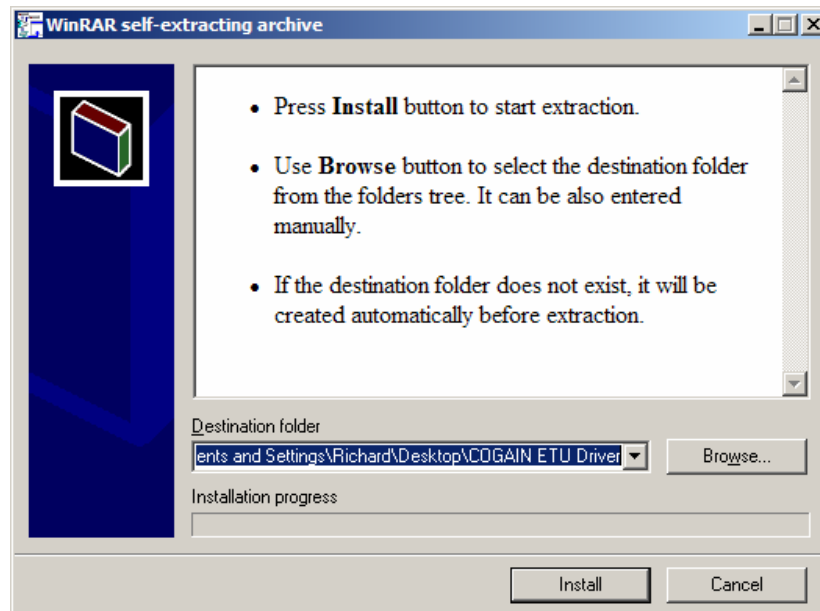


Figure 9. Installing the COGAIN ETSD Test Application

Installing the ETSD test application is a simple one-step process, Figure 9, where the user simply needs to either accept the default suggested location for the application, or may choose any directory they wish for the application.

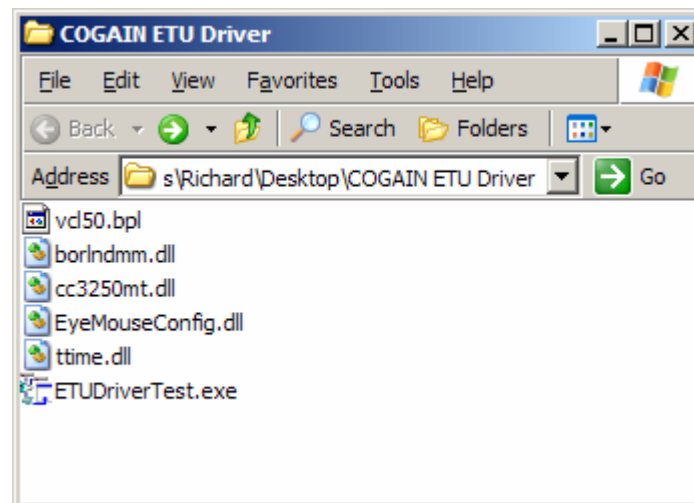


Figure 10. Files installed by the COGAIN ETSD Test Application

After installation six files are installed, Figure 10. The main file of interest for the end user is 'ETUDriverTest.exe' which starts the test application when double clicked. This ends the installation of the COGAIN ETSD test application.

2.5 Using the COGAIN ETSD test application

The COGAIN ETSD test application is started by double clicking the main 'ETUDriverTest.exe' file after installation. The application starts with a blank screen, Figure 11.

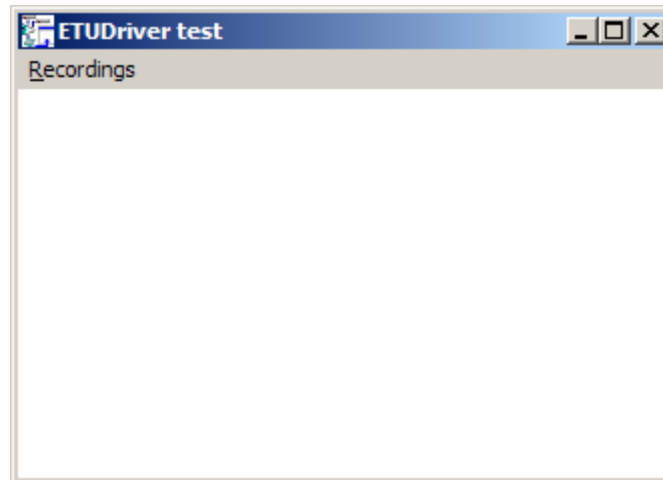


Figure 11. The COGAIN ETSD Test Application

The first step to using the test application is to select which eye tracker or eye tracker mouse emulator you wish to test. This option is accessed by first selecting 'Driver' from the 'Recordings' menu option, Figure 12.

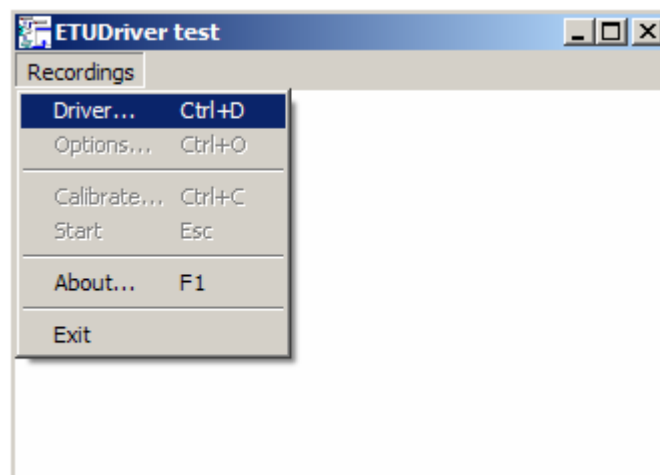


Figure 12. Selecting an eye tracker driver on the COGAIN ETSD Test Application step 1

A sub-window then allows the selection of all installed available drivers, Figure 13. Note that the 'SMI EyeLink', 'iViewX', 'Mouse' emulation of an eye tracker, a random 'simulator' of data, and the 'Tobii' eye tracker drivers are all available as choices.

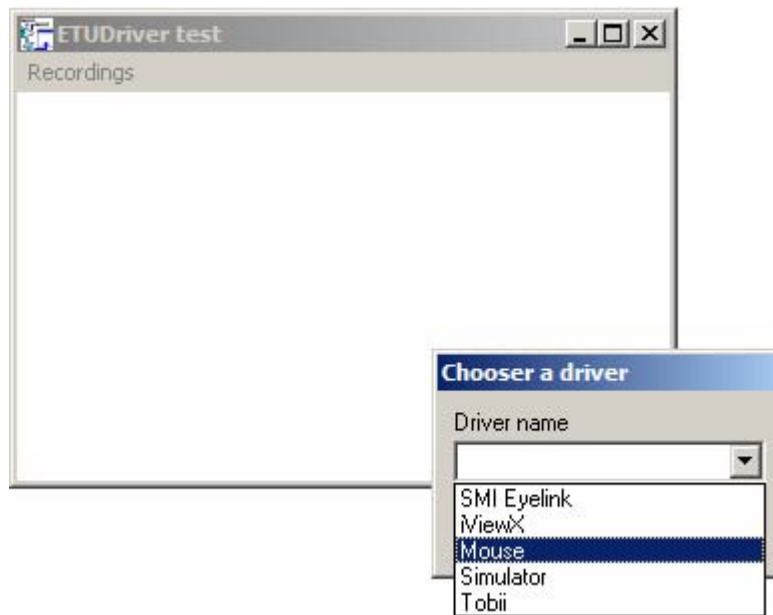


Figure 13. Selecting an eye tracker driver on the COGAIN ETSD Test Application step 2

After selecting the required driver to test, the user may set a range of options for the driver. These options are available via the 'Options' menu item, Figure 14. Note that options are not available in the menu until a driver is selected.

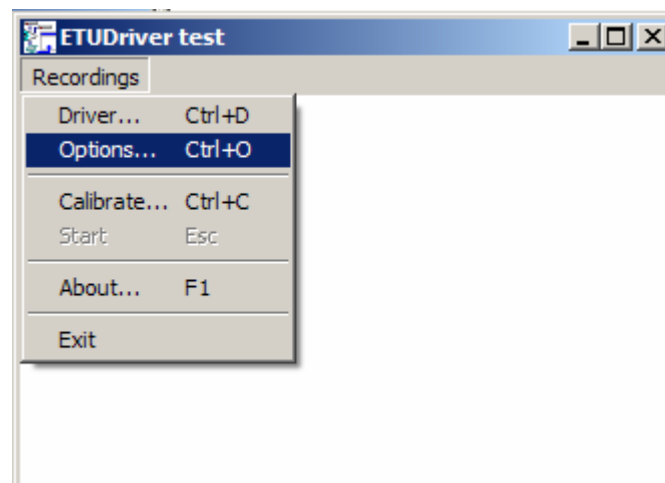


Figure 14. Selecting eye tracker driver options on the COGAIN ETSD Test Application step 1

There are a large range of options currently available in this first release of the COGAIN ETSD, Figure 15. Addressing these, in normal use the default settings (as shown in Figure 15) may be used, but the user may choose or alter:

- the data folder to save the captured test data from the driver on test
- the format for the saved data (XML is the COGAIN standard, but ASCII and binary are also available)
- down-sample the sampling rate to reduce data

- record mouse clicks (button presses) in the data
- record any keyboard keystrokes in the data
- record a video of the computer screen to a video file (full screen or windowed)
- correct eye tracker drift when recording
- bind the cursor to the gaze position (create an 'eye mouse')

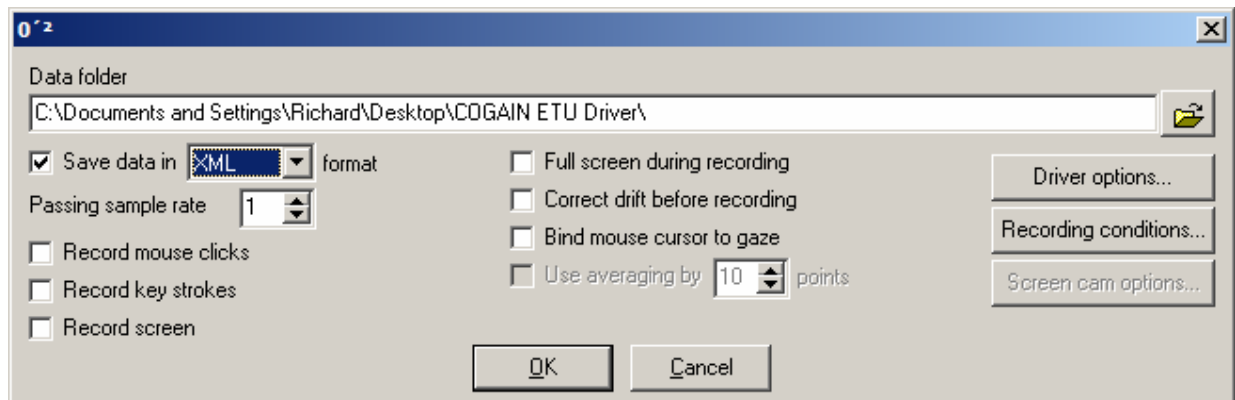


Figure 15. Selecting eye tracker driver options on the COGAIN ETSD Test Application step 2

The 'Options dialog box also allows the user to change 'Driver options' and Recording options'. Selecting 'Driver options' shows a dialog box specific to the current test driver selected, in this example, the desktop mouse eye tracker emulator, Figure 16. Addressing these, in normal use the default settings (as shown in Figure 16) may be used, but the user may choose or alter:

- Change the sample frequency (rate) that data samples are gathered
- Set how many samples to example as a set for any fixation detection
- Set the time period (update) for a fixation
- Set the boundary or path distance that contains a fixation
- Set the maximum jump (small eye movement change) that is tolerated during a fixation
- Set the minimum time that constitutes a fixation

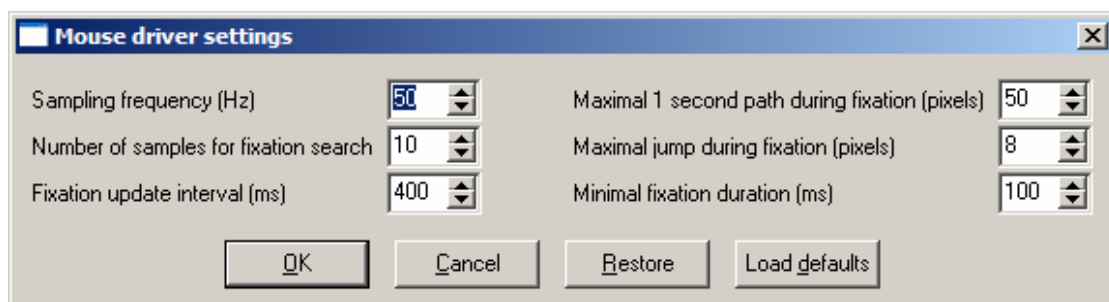
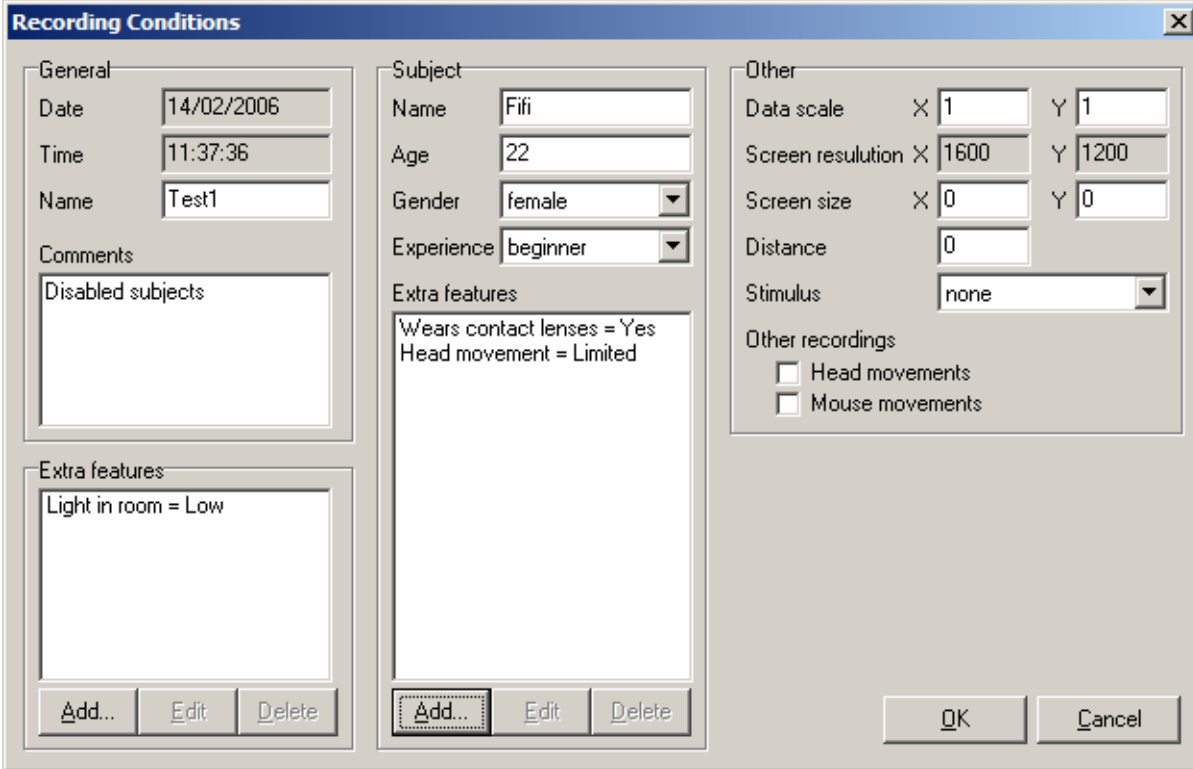


Figure 16. Selecting eye tracker driver options on the COGAIN ETSD Test Application step 3

Selecting 'Recording options' shows a dialog box that allows the user to add and modify the type of data recorded by the test application. This is a particularly useful and important feature as a user may add any types of data they wish to the recorded file without causing any incompatibility with applications that may not be designed to use this new data. This is possible by the use of XML coding in the COGAIN ETSD files and data streaming, where all data is tagged with a name or type giving an extensible data language, and data tags or types not recognised by any reading application may simply be ignored by that application without causing software problems.



The 'Recording Conditions' dialog box is divided into three main sections: General, Subject, and Other. The General section includes fields for Date (14/02/2006), Time (11:37:36), Name (Test1), and a Comments text area containing 'Disabled subjects'. The Subject section includes fields for Name (Fifi), Age (22), Gender (female), Experience (beginner), and an Extra features text area containing 'Wears contact lenses = Yes' and 'Head movement = Limited'. The Other section includes fields for Data scale (X 1, Y 1), Screen resolution (X 1600, Y 1200), Screen size (X 0, Y 0), Distance (0), Stimulus (none), and checkboxes for Other recordings (Head movements, Mouse movements). Each section has an 'Add...', 'Edit', and 'Delete' button at the bottom. The dialog box has an 'OK' button and a 'Cancel' button at the bottom right.

Figure 17. Selecting eye tracker driver options on the COGAIN ETSD Test Application step 4

Addressing the options available (as shown in Figure 17) the user may choose:

- General:
 - The test name
 - Any comments about the test condition
 - Add extra test related data types or features and associated values
- Subject:
 - Subject's name
 - Age
 - Gender
 - Experience with an eye tracker
 - Add any extra subject related data types or features and associated values
- Other:

- Data scaling (downscaling the data to a smaller screen for example)
- Screen resolution
- Screen size
- Subject's seating distance from the screen
- The type of external stimulus given to the subject (test prompts and so on)
- The option to record head movements
- The option to record desktop mouse movements

Examining the options to add new recorded test data, the user may add data types and values to the recorded or streamed data by using the 'Add...' buttons, these display a 'New extra feature' dialog box, Figure 18.

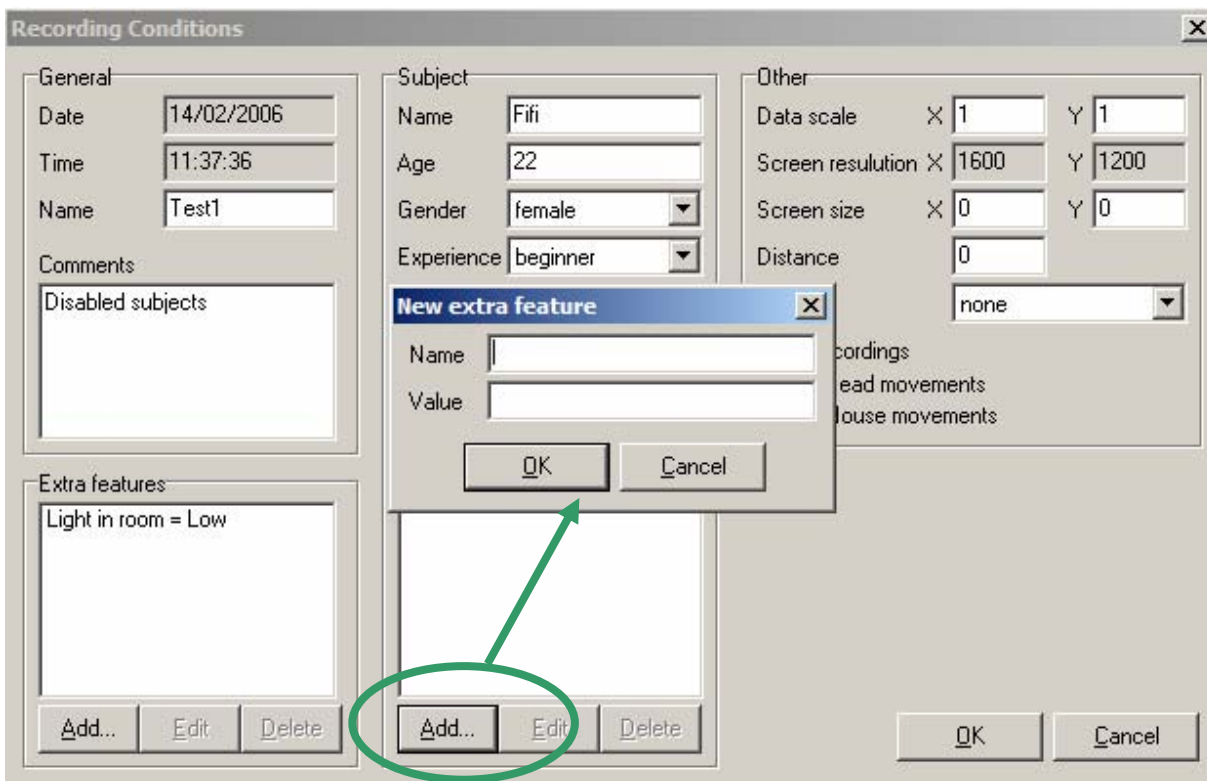
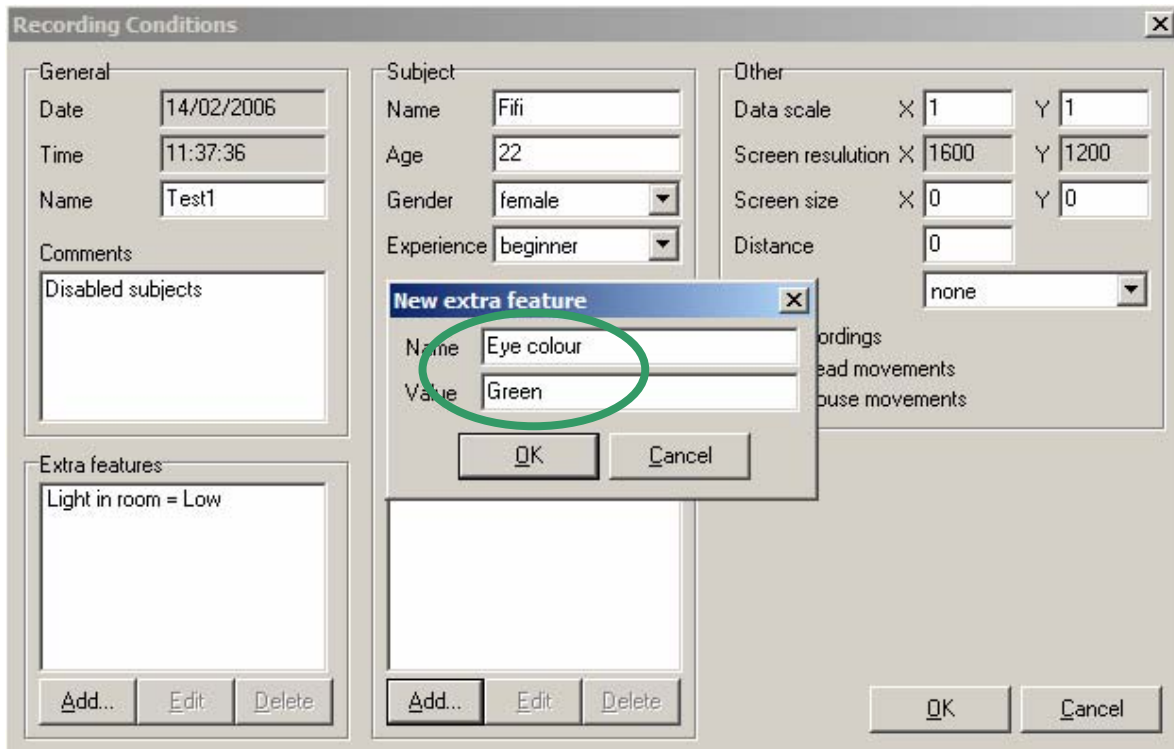


Figure 18. Selecting eye tracker driver options on the COGAIN ETSD Test Application step 5

The 'New extra feature' dialog box allows a textual 'Name' and 'Value' to be added to the data file, Figures 19 and 20 show that addition of an 'Eye colour' data type, and a value for this subject of 'Green' to the test application data.



Recording Conditions

General
 Date: 14/02/2006
 Time: 11:37:36
 Name: Test1
 Comments: Disabled subjects

Subject
 Name: Fifi
 Age: 22
 Gender: female
 Experience: beginner

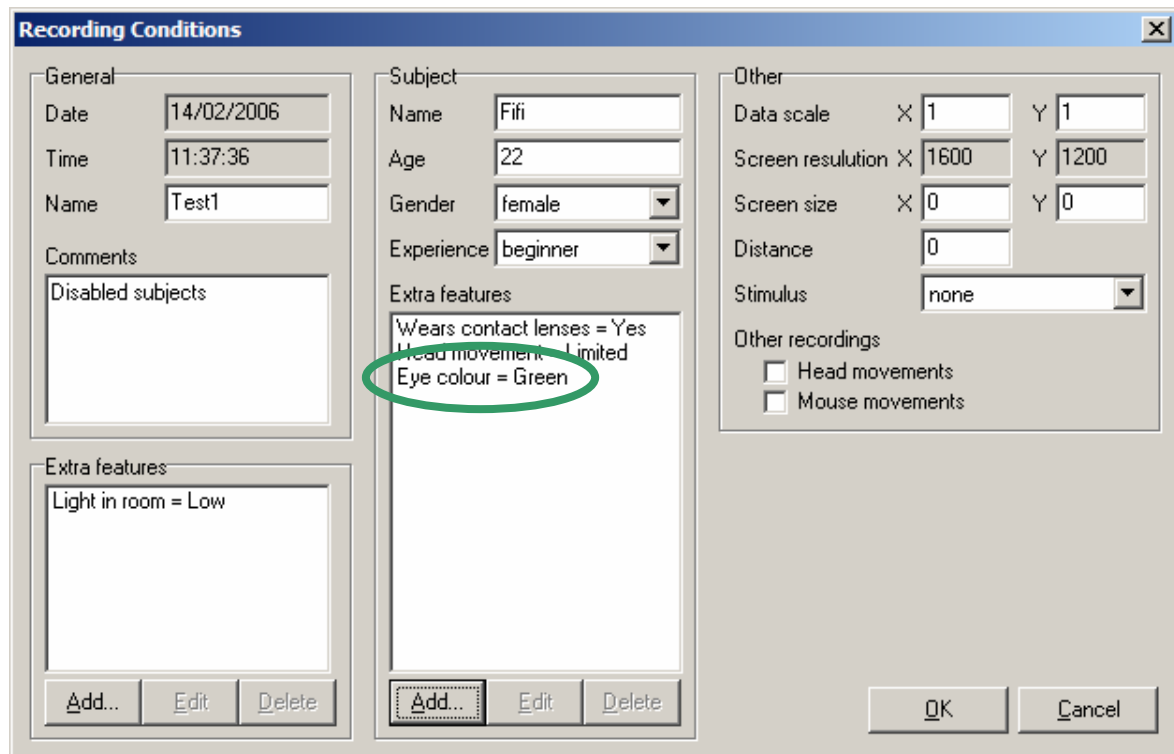
Other
 Data scale: X 1 Y 1
 Screen resolution: X 1600 Y 1200
 Screen size: X 0 Y 0
 Distance: 0
 none

Extra features:
 Light in room = Low

New extra feature
 Name: Eye colour
 Value: Green

Buttons: Add... Edit Delete (for Extra features), Add... Edit Delete (for New extra feature), OK Cancel

Figure 19. Selecting eye tracker driver options on the COGAIN ETSD Test Application step 6



Recording Conditions

General
 Date: 14/02/2006
 Time: 11:37:36
 Name: Test1
 Comments: Disabled subjects

Subject
 Name: Fifi
 Age: 22
 Gender: female
 Experience: beginner

Other
 Data scale: X 1 Y 1
 Screen resolution: X 1600 Y 1200
 Screen size: X 0 Y 0
 Distance: 0
 Stimulus: none

Extra features:
 Light in room = Low

Subject Extra features:
 Wears contact lenses = Yes
 Head movement = Limited
 Eye colour = Green

Buttons: Add... Edit Delete (for Extra features), Add... Edit Delete (for Subject Extra features), OK Cancel

Figure 20. Selecting eye tracker driver options on the COGAIN ETSD Test Application step 7

The COGAIN ETSD test application is now ready for use. The first step in use is to select 'Calibrate' from the 'recordings' menu item, this invokes any manufacturers calibration routines associated with the currently selected eye tracking driver, Figure 21.

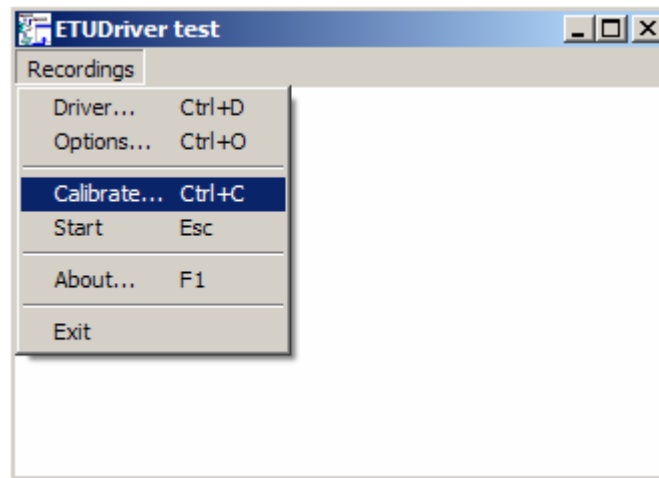


Figure 21. Running the COGAIN ETSD Test Application step 1

After calibration has been run the test application may be started by selecting 'Start' from the 'recordings' menu item, Figure 22.

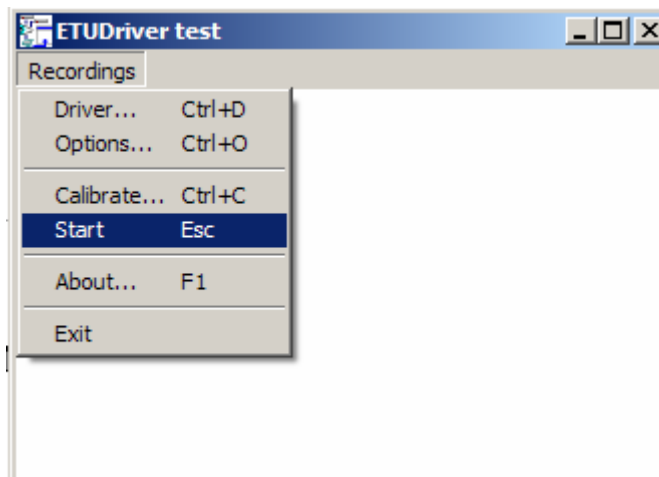


Figure 22. Running the COGAIN ETSD Test Application step 2

When running the test application shows the captured gaze data as a trail of dots, together with the start of fixations as unfilled circles, and filled red circles as detected fixations. (The ability to detect fixation is used to allow 'dwell click' tools to determine where the user is looking and fixating when using an eye tracker). The test application running and streaming and capturing data is show in Figure 23.

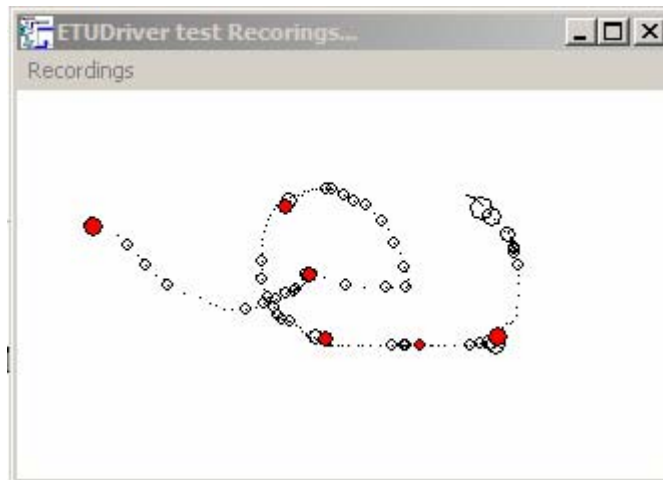


Figure 23. Running the COGAIN ETSD Test Application step 3

Once the user is satisfied that the test application has sufficiently tested the COGAIN ETSD, the application may be halted by pressing the 'Escape' keyboard key at any time. Once halted, the test application prompts the user to save any data captures by the test application from the COGAIN ETSD driver, Figure 24. This ends the operational description of the COGAIN ETSD test application.

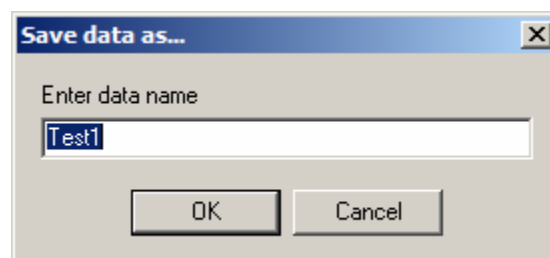


Figure 24. Running the COGAIN ETSD Test Application step 4

2.6 The COGAIN ETSD test application data

The data files captured by the COGAIN ETSD test application from the COGAIN ETSD driver may be viewed to determine if the ETSD is working correctly and controlling the eye tracking hardware and streaming data correctly. Examining the ETSD directory after using the ETSD test application shows a new directory named after the data name given by the user 'Test1' during operation of the test application, Figure 25.

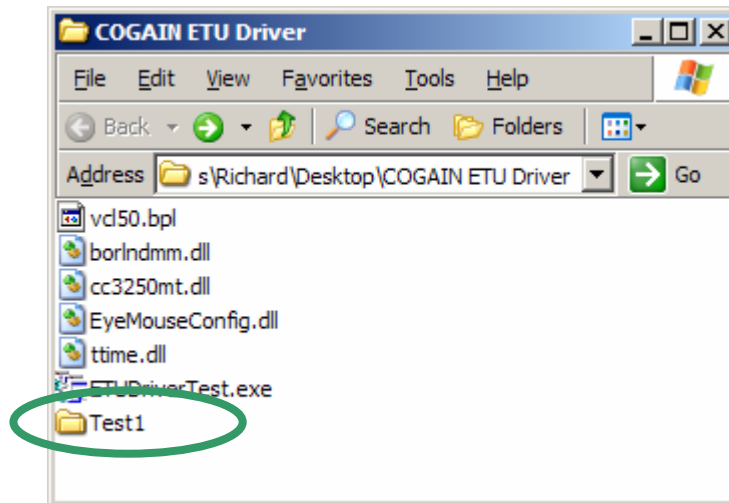


Figure 25. The COGAIN ETSD Test Application data step 1

Examining the contents of the directory shows four files of data, Figure 26.

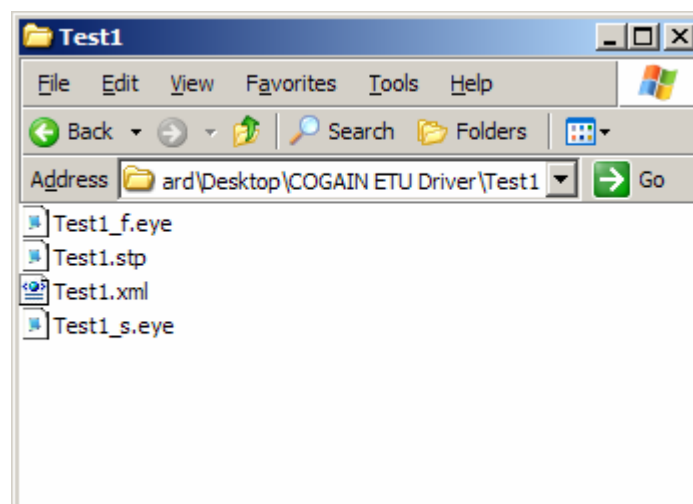


Figure 26. The COGAIN ETSD Test Application data step 2

Examining these files, 'Test1.stp' is an ASCII text file containing a summary of the set up conditions of the driver, as detailed in Section 2.5. This allows a rapid examination of the operating conditions of the driver. Note that the additional features, such as 'Eye colour', that were added are also present in this file, Figure 27.

```
[SetupGeneral]  
Date=14/02/2006  
Time=11:43:15  
Name=Test1  
Comments=Disabled subjects  
[SetupSubject]
```

```
Name=Fifi
Age=22
Gender=2
ExperienceLevel=2
[SetupSubjectExtraFeatures]
Wears contact lenses=Yes
Head movement=Limited
Eye colour=Green
[SetupConditions]
DataScaleX=1
DataScaleY=1
ScreenResolutionX=1600
ScreenResolutionY=1200
ScreenSizeX=0
ScreenSizeY=0
Distance=0
StimuliType=0
OtherRecordings=0
[SetupConditionsExtraFeatures]
Light in room=Low
[SetupDeviceState]
Frequency=50
Eye=1
Coordinates=3
HasPupilData=0
HasHeadMovementsData=0
Description=Eye tracker emulator by mouse. Version 2.0
[GazeRecording]
RecordingFolder=C:\Documents and Settings\Richard\Desktop\COGAIN ETU
Driver\
SamplesSkippingRate=1
DataSaving=1
BindGazeToCursor=0
BindUseAveraging=0
BindAveragingPointsNumber=10
DriftCorrectionBeforeRecording=0
FullScreen=0
DataFormat=3
RecordScreen=0
RecordKeyDowns=0
RecordMouseClicks=0
```

Figure 27. The COGAIN ETSD Test Application data file 1

'Test1_f.eye' is an ASCII text file, this contains a summary of eye fixations detected by the COGAIN ETSD, Figure 28.

| Num | Time | Dur | Eye | AvgX | AvgY | OffsX | OffsY |
|-----|------|------|-----|------|------|-------|-------|
| 1 | 26 | 998 | 0 | 365 | 333 | 0 | 0 |
| 2 | 1485 | 462 | 0 | 473 | 357 | 0 | 0 |
| 3 | 2409 | 457 | 0 | 461 | 323 | 0 | 0 |
| 4 | 3128 | 519 | 0 | 481 | 389 | 0 | 0 |
| 5 | 3750 | 353 | 0 | 528 | 392 | 0 | 0 |
| 6 | 4212 | 1140 | 0 | 567 | 388 | 0 | 0 |

Figure 28. The COGAIN ETSD Test Application data file 2

'Test1_s.eye' is an ASCII text file, this contains all of the eye co-ordinates, time stamps and other data streamed and captured by the COGAIN ETSD, Figure 29.

| TimeLEvent | LX | LY | LPupil | REvent | RX | RY | RPupil | OffsX | OffsY |
|------------|----|-----|--------|--------|----|--------|--------|-------|-------|
| 26 | 1 | 365 | 333 | 1000 | 0 | -32767 | -32767 | 0 | 0 |
| 38 | 1 | 365 | 333 | 1000 | 0 | -32767 | -32767 | 0 | 0 |
| 58 | 1 | 365 | 333 | 1000 | 0 | -32767 | -32767 | 0 | 0 |
| 78 | 1 | 365 | 333 | 1000 | 0 | -32767 | -32767 | 0 | 0 |
| 98 | 1 | 365 | 333 | 1000 | 0 | -32767 | -32767 | 0 | 0 |
| 118 | 1 | 365 | 333 | 1000 | 0 | -32767 | -32767 | 0 | 0 |
| 138 | 1 | 365 | 333 | 1000 | 0 | -32767 | -32767 | 0 | 0 |

Figure 29. The COGAIN ETSD Test Application data file 3

Finally, 'Test1.xml' contains the full data set in COGAIN ETSD format, Figure 30. Note how the data is encapsulated within data tags, or names, that denote what data is following. Encapsulation starts with a data name in brackets, for example <date>, this is followed by the data named for example '14/02/06', and encapsulated and ended by a negation of the data name, for example </date>. This method is followed for all data captured from the manufacturer's eye tracking hardware driver. Also note that the data is automatically indented using tabs, this allows ease of reading without affecting reading of the file by third party applications. Comments have been added in this version of the file (denoted by /* comment */) to ease reading of the file:

```
<xml version="1.0">
<start>                                /* main start of the data */
  <ident>1</ident>
  <setup>                                /* set up information */
    <date>14/02/2006</date>
    <time>11:43:15</time>
    <name>Test1</name>
    <comments>Disabled subjects</comments>
```



```
<subject>                                     /* subject information */
  <name>Fifi</name>
  <age>22</age>
  <gender>female</gender>
  <experience_level>beginner</experience_level>
  <extra_parameters>
    <parameter>                                /* extra user parameters added */
      <name>Wears contact lenses</name>
      <value>Yes</value>
    </parameter>
    <parameter>
      <name>Head movement</name>
      <value>Limited</value>
    </parameter>
    <parameter>
      <name>Eye colour</name>
      <value>Green</value>
    </parameter>
  </extra_parameters>
</subject>                                   /* end of subject information */
<tracking_conditions>                       /* start tracking conditions */
  <timescale>ms</timescale>
  <data.scales>
    <x>1</x>
    <y>1</y>
  </data.scales>
  <screen>
    <resolution>
      <x>1600</x>
      <y>1200</y>
    </resolution>
    <size>
      <x>0</x>
      <y>0</y>
    </size>
    <units>cm</units>
  </screen>
  <distance_from_subject>0</distance_from_subject>
```

```
<stimulus>none</stimulus>
<extra_parameters>
  <parameter>
    <name>Light in room</name>
    <value>Low</value>
  </parameter>
</extra_parameters>
</tracking_conditions>
<device.state>                                /* start of device options */
  <name>Eye tracker emulator by mouse. Version 2.0</name>
  <sampling_frequency>50</sampling_frequency>
  <eye>left</eye>
  <coordinates>xy</coordinates>
  <track_pupil_size>no</track_pupil_size>
  <track_head_position>no</track_head_position>
</device.state>                                /* end of device options */
</setup>

<screen>                                       /* start of main gaze data */
  <gaze>
    <samples>
      <sample>
        <timestamp>26</timestamp>
        <number>1</number>
        <eye side="left">
          <x>365</x>
          <y>333</y>
          <pupil.size>1000</pupil.size>
          <event>1</event>
        </eye>
      </sample>
      <sample>
        <timestamp>38</timestamp>
        <number>2</number>
        <eye side="left">
          <x>365</x>
          <y>333</y>
          <pupil.size>1000</pupil.size>
```

```

        <event>1</event>
    </eye>
</sample>
<sample>
    <timestamp>58</timestamp>
    <number>3</number>
    <eye side="left">
        <x>365</x>
        <y>333</y>
        <pupil.size>1000</pupil.size>
        <event>1</event>
    </eye>
</sample>                                /* end of main gaze data */

<fixations>                                /* start of fixations data */
    <fixation>
        <timestamp>26</timestamp>
        <number>1</number>
        <duration>998</duration>
        <eye>0</eye>
        <x>365</x>
        <y>333</y>
    </fixation>
    <fixation>
        <timestamp>1485</timestamp>
        <number>2</number>
        <duration>462</duration>
        <eye>0</eye>
        <x>473</x>
        <y>357</y>
    </fixation>
</fixations>                                /* end of fixations gaze data */

</gaze>
</screen>
<other_events />
</start>                                /* end of data */

```

Figure 30. The COGAIN ETSD Test Application data file 4

2.7 Summary

This ends the testing of the COGAIN ETSD functions, showing that the driver can connect to a driver and stream and save data in the COGAIN standard format. The next section gives the more detailed programmer guide to the COGAIN ETSD API functions that a third party application writer will need to connect to the COGAIN ETSD API, and through this to any supported eye tracking hardware.

3 The COGAIN ETSD programmer guide to the COM interface

The following sections detail how a programmer may interface with the COGAIN ETSD COM interface. These instructions will be easily interpreted by a programmer, and allow full use of the COGAIN ETSD driver interface.

3.1 Enumerations

EiETUDDataFormat

The data format that is using to store gaze data

```
typedef enum EiETUDDataFormat
{
    dfUnknown = 0,
    dfAscii = 1,
    dfBinary = 2,
    dfXML = 3
} EiETUDDataFormat;
```

EiETUDEye

Indicator of the eye producing gaze data. Values can be used as a mask.

```
typedef enum EiETUDEye
{
    eyeNone = 0,
    eyeLeft = 1,
    eyeRight = 2,
    eyeBoth = 3
} EiETUDEye;
```

EiETUDCoordinates

Coordinates for measurement in ETU-Driver. Values can be used as a mask.

```
typedef enum EiETUDCoordinates
{
    cooNone = 0,
    cooX = 1,
    cooY = 2,
    cooXY = 3,
    cooZ = 4,
    cooXYZ = 7
} EiETUDCoordinates;
```

EiETUDUserExperienceLevel

User experience levels

```
typedef enum EiETUDUserExperienceLevel
{
    uelUnknown = 0,
    uelNovice = 1,
    uelBeginner = 2,
    uelAdvanced = 3,
    uelExpert = 4
} EiETUDUserExperienceLevel;
```

EiETUDGender

User gender indicators

```
typedef enum EiETUDGender
{
    ugUnknown = 0,
    ugMale = 1,
    ugFemale = 2
} EiETUDGender;
```

EiETUDPassingValue

Some values that may be required to pass to converters. ETU-Driver does it automatically, thus applications normally should not do it.

```
typedef enum EiETUDPassingValue
{
    pvUnknown = 0,
    pvApplication = 1,
    pvWindowHandle = 2
} EiETUDPassingValue;
```

EiETUDSystemEvents

Several system events that could be saved in “system and experiment events” data file.

```
typedef enum EiETUDSystemEvents
{
    seUnknown = 0,
    seMouseLClick = 1,
    seMouseRClick = 2,
    seKeyPress = 3
} EiETUDSystemEvents;
```

EiETUDWindowsMessages

Windows messages using to pass data from converter and application to ETU-Driver.

```
typedef enum EiETUDWindowsMessages
```

```
{
    wmGazeEvent = 49408,
    wmDriverControlMessage = 49664,
    wmApplicationControlMessage = 49920
} EiETUDWindowsMessages;
```

EiETUDGazeEvents

Indicators of gaze event type passed in event OnDataEvent

```
typedef enum EiETUDGazeEvents
{
    geUnknown = 0,
    geSample = 1,
    geFixationStart = 2,
    geFixationEnd = 3,
    geFixationUpdate = 4,
    geBlinkStart = 5,
    geBlinkEnd = 6
} EiETUDGazeEvents;
```

EiETUDDeviceControlMessages

Type of device control messages coming from converter and application to ETU-Driver.

```
typedef enum EiETUDDeviceControlMessages
{
    dcmUnknown = 0,
    dcmStartTracking = 1,
    dcmStopTracking = 2,
    dcmShiftX = 3,
    dcmShiftY = 4,
    dcmSetOffsetX = 5,
    dcmSetOffsetY = 6,
    dcmGetDriverName = 7,
    dcmGetRecordingOptions = 8,
    dcmSetRecordingOptions = 9,
    dcmGetDeviceState = 10,
    dcmSetDeviceState = 11,
    dcmGetRecordingSetupGeneralData = 12,
    dcmSetRecordingSetupGeneralData = 13,
    dcmGetRecordingConditionsData = 14,
    dcmSetRecordingConditionsData = 15
} EiETUDDeviceControlMessages;
```

EiETUDAppControlMessages

Type of application control messages coming from converter to application via ETU-Driver.

```
typedef enum EiETUDAppControlMessages
{
    acmUnknown = 0,
    acmToggleFullScreen = 1,
    acmKeyPress = 2,
    acmGetProfileName = 3,
```

```

    acmGetProfileDir = 4
} EiETUDAppControlMessages;

```

EiETUDOtherRecordings

IDs of other recordings synchronized with gaze data and places in the same folder

```

typedef enum EiETUDOtherRecordings
{
    orNone = 0,
    orHeadMovements = 1,
    orMouseMovements = 2
} EiETUDOtherRecordings;

```

EiETUDStimulus

Types of common stimuli used in trials

```

typedef enum EiETUDStimulus
{
    stNone = 0,
    stImages = 1,
    stVideos = 2,
    stOther = 3
} EiETUDStimulus;

```

EiETUDGazeDetectedEvent

Events detected by manufacturer's software.

```

typedef enum EiETUDGazeDetectedEvent
{
    gdeNone = 0,
    gdeFixationStart = 1,
    gdeFixationEnd = 2,
    gdeFixationUpdate = 4,
    gdeBlinkStart = 8,
    gdeBlinkEnd = 16
} EiETUDGazeDetectedEvent;

```

3.2 Structures

SiETUDSample

Gaze sample containing both eyes data

```

struct SiETUDSample
{
    long Number;           // Ordinal number
    unsigned_long Time;    // Time relative to the recording start time (ms)
}

```



```

    // Next 4 variable consists of parts for data of each eye.

    long Event[2];    // if 0, this is saccade's sample, if positive
                     // integer - this is fixation's sample (value equal
                     // to the ordinal number of this fixation), if
                     // negative - this is the first sample after a blink
                     // (value equal to the ordinal number of this blink)

    double X[2];      // Gaze X
    double Y[2];      // Gaze Y

    double Pupil[2];  // Pupil size

    SiETUDFloatPoint Offset;    // Difference between screen and on-screen
                               // stimuli top-left corners
};

```

SiETUDSingleSample

Gaze sample containing single eye data

```

struct SiETUDSingleSample
{
    long Number;          // Ordinal number
    unsigned_long Time;   // Time relative to the recording start time (ms)
    short Eye;           // Eye produced this sample
    double X;            // Gaze X
    double Y;            // Gaze Y
    double Pupil;        // Pupil size

    SiETUDFloatPoint Offset;    // Difference between screen and on-screen
                               // stimuli top-left corners
};

```

SiETUDFixation

Fixation

```

struct SiETUDFixation
{
    long Number;          // Ordinal number
    unsigned_long Time;   // Time relative to the recording start time (ms)
    unsigned_long Duration; // Fixation duration
    short Eye;           // Eye produced this fixation
    double X;            // Fixation X of mass center
    double Y;            // Fixation Y of mass center
    SiETUDFloatPoint Offset;    // Difference between screen and on-screen
                               // stimuli top-left corners
};

```

SiETUDBlink

Blink

```

struct SiETUDBlink

```

```
{
    long Number;           // Ordinal number
    unsigned_long Time;    // Time relative to the recording start time (ms)
    unsigned_long Duration; // Blink duration
    short Eye;             // Eye produced this blink
};
```

SiETUDExtraEvent

System and environment events

```
struct SiETUDExtraEvent
{
    long Number;           // Ordinal number
    unsigned_long Time;    // Time relative to the recording start time (ms)
    short Type;            // Event type. Low byte - for system events (could be
                          // values from enumeration EiETUDSystemEvents), high
                          // byte - for environment/experiment events (could be
                          // values from enumeration EiETUDStimulus).
    signed_char Data[1024]; // Event-related data (e.g., image file name,
                          // key virtual code, mouse button clicked with
                          // coordinates, etc.)
};
```

SiETUDScreenRecordingOptions

Screen recording options.

```
struct SiETUDScreenRecordingOptions
{
    unsigned_short FrameRate; // Capturing frame rate, if next flag is
                          // FALSE
    VARIANT_BOOL OptimalFrameRate; // Optimal frame rate chosen by
                          // capturing software
    VARIANT_BOOL RecordCursor; // If TRUE, mouse cursor will appear in
                          // capture video
    VARIANT_BOOL RecordAudio; // If TRUE, audio will be recorded with
                          // video
    long Mode;                // Mode - either NORMAL or QUICK
                          // (captures frames only if something
                          // changes on the screen)
    VARIANT_BOOL BoostPriority; // If TRUE, assigns a REAL_TIME process
                          // priority for screen capturing
};
```

SiETUDRecordingOptions

Gaze data recording options

```
struct SiETUDRecordingOptions
{
    signed_char DirName[1024]; // Folder to store gaze data
    signed_char SkipSample;    // Rate of samples to be accepted (1
                          // means accept each sample, 2 - every 2nd,
                          // 3 - every 3rd and so on.
```

```

    VARIANT_BOOL SaveData;           // If FALSE, data is not saved on the
                                     // disk
    VARIANT_BOOL BindCursor;         // If TRUE, mouse cursor follows gaze

    VARIANT_BOOL BindUseAvg;          // If TRUE, mouse cursor coordinates are
    signed_char BindAvgPointCount;    // the average of last<BindAvgPointCount>
                                     // gaze points

    VARIANT_BOOL AutoDriftCorrection; // If TRUE, system shows auto-drift
                                     // correction window after user calls
                                     // startTracking function, but before
                                     // actual tracking starts

    VARIANT_BOOL AutoFullScreen;      // If TRUE, ETUDriver fires
                                     // OnAppControlEvents with parameter
                                     // aEventID = acmToggleFullScreen

    EiETUDDataFormat DataFormat;      // Type of gaze data format

    VARIANT_BOOL RecordScreen;        // If TRUE, screen will be captured to
                                     // video file during recording

    VARIANT_BOOL RecordKeyDowns;      // If TRUE, key-down strokes will be
                                     // captured to system and environment
                                     // events file during recording

    VARIANT_BOOL RecordMouseClicks;   // If TRUE, mouse clicks will be
                                     // captured to system and environment
                                     // events file during recording

    VARIANT_BOOL AlwaysUseOwnGazeParser; // If TRUE, ETUDriver ignores gaze-
                                     // related events detected by
                                     // manufacturer's software and uses
                                     // its own algorithms for such
                                     // detection (fixations and blink
                                     // from samples)
};

```

SiETUDDeviceState

Device state and capturing properties

```

struct SiETUDDeviceState
{
    signed_char Description[64]; // Device name and version
    long Frequency;              // Sampling frequency
    EiETUDEye Eye;               // Eye to be captured
    signed_char Coordinates;      // Gaze-point coordinates
    VARIANT_BOOL HasPupilData;    // If TRUE, device captures pupil size
    VARIANT_BOOL HasHeadMovementsData; // If TRUE, device captures head
                                     // movements
    long GazeDetectedEvents;      // Combination of EiETUDGazeDetectedEvent
                                     // flags indicating what gaze events are
                                     // detecting by manufacturer's software
};

```

SiETUDSubjectData

Data about user/subject of a trial (part of SiETUDRecordingSetupGeneralData structure)

```
struct SiETUDSubjectData
{
    signed_char Name[64];           // Subject name
    signed_char Age;                // Subject age

    EiETUDGender Gender;           // Subject gender

    EiETUDUserExperienceLevel ExperienceLevel; // Subject experience
                                           // level (in any sense)

    signed_char ExtraFeatures[2048]; // Extra feature: string that
                                     // contains pairs of "name=value",
                                     // delimited by comma
};
```

SiETUDRecordingSetupGeneralData

General features of recording trial. Part of SiETUDRecordingSetupData structure.

```
struct SiETUDRecordingSetupGeneralData
{
    signed_char Date[32];           // Recording date
    signed_char Time[16];           // Recording start time

    signed_char Name[64];           // Recording name (may differ from folder
                                     // name)

    SiETUDSubjectData SubjectData; // Subject's features

    signed_char Comments[1024];     // Any trial-related comments
};
```

SiETUDRecordingConditionsData

Other data related to recording conditions. Part of SiETUDRecordingSetupData structure.

```
struct SiETUDRecordingConditionsData
{
    SiETUDFloatPoint DataScale;     // Scale of gaze data

    SiETUDIntPoint ScreenResolution; // Screen resolution
    SiETUDFloatPoint ScreenSize;     // Screen size

    double Distance;                // Subject distance from monitor

    EiETUDStimulus Stimulus;         // Type of stimuli

    EiETUDOtherRecordings OtherRecordings; // IDs of other synchronized
                                           // recordings
};
```

```

        signed_char ExtraFeatures[2048];    // Extra feature: string that contains
                                           // pairs of "name=value", delimited by
                                           // comma
    };

```

SiETUDRecordingSetupData

All non-gaze-data recording condition options and conditions

```

struct SiETUDRecordingSetupData
{
    SiETUDRecordingSetupGeneralData General;
    SiETUDRecordingConditionsData Conditions;
    SiETUDDeviceState DeviceState;
};

```

SiETUDFloatPoint

Point with values of double precision

```

struct SiETUDFloatPoint
{
    double X;
    double Y;
};

```

SiETUDIntPoint

Point of integer precision

```

struct SiETUDIntPoint
{
    long X;
    long Y;
};

```

SiETUDConstants

A set of constants using in ETUDDriver

```

struct SiETUDConstants
{
    LPSTR DirAPIs;    // Relative paths to folders with driver's APIs,
    LPSTR DirINIs;    // converters option files (INIs) and
    LPSTR DirLOGs;    // their logs.

    LPSTR DirDrivers; // Folder with converters
    LPSTR DirImages;  // Folder with images stimuli
    LPSTR DirVideos;  // Folder with video stimuli

    LPSTR HeaderFix;  // Headers of ASCII data files: fixations,
    LPSTR HeaderSmp;  // samples,
    LPSTR HeaderBlk;  // blinks and
    LPSTR HeaderExE;  // system and environment events
};

```

```

LPSTR ExtAscii;    // Data files extensions according to format: ASCII,
LPSTR ExtBinary;  // binary and
LPSTR ExtXML;      // XML
LPSTR ExtCalibration; // Calibration files extension
LPSTR ExtSetup;    // Experiment setup file extension

LPSTR EndingSmp;   // Gaze data file endings: samples,
LPSTR EndingFix;   // fixations,
LPSTR EndingBlk;   // blinks and
LPSTR EndingExE;   // system and environment events

LPSTR DataDelimiter; // Data delimiter in ASCII files
LPSTR LineEnd;        // Line-end character(s) in ASCII files
LPSTR ExEDataDelimiter; // Data delimiter in strings with data of
                        // system and environment events

unsigned_long MaxExEDataSize; // Max size of data string in
                              // system and environment events file

long MinXY;      // Min X and Y values of valid gaze point
long MaxXY;      // Max X and Y values of valid gaze point
};

```

3.3 Properties

Each property has 1 (if it is read-only) or 2 corresponding COM interface functions – to get and set a value; The common forms of these functions are the following:

- o Get: <property_type> get_<property_name>(void);
- o Set: void set_<property_name> (<property_type> Param1 /*[in]*/);

DeviceCount

long DeviceCount

Type: read

Description: returns total number of devices

Index

long Index

Type: read-write

Description: Index of the device currently in use (-1, if no one device is using)

Name

LPSTR Name

Type: read

Description: Device name

OptionsFile

LPSTR OptionsFile

Type: read-write

Description: Options file. Empty initially, so it is supposed to be assigned by an application

LastRecordName

LPSTR LastRecordName

Type: read

Description: The name of last recording

RecordingOptions

SiETUDRecordingOptions* RecordingOptions

Type: read-write

Description: Gaze data recording options (see structure SiETUDRecordingOptions)

RecordingSetupData

SiETUDRecordingSetupData* RecordingSetupData

Type: read-write

Description: Other recording options (see structure SiETUDRecordingSetupData)

Shifting

SiETUDFloatPoint* Shifting

Type: read-write

Description: Gaze-point correction values. Typically, an application sets these values to correct detected drift

Offset

SiETUDFloatPoint* Offset

Type: read-write

Description: Stimuli top-left corner offset from top-left corner of the screen. Typically, an application sets these values to preserve correct data mapping to stimuli

Active

TOLEBOOL Active

Type: read

Description: TRUE if gaze data streaming is on

Ready

TOLEBOOL Ready

Type: read

Description: TRUE if device is ready to be used

Calibrated

TOLEBOOL Calibrated

Type: read

Description: TRUE if device is calibrated

Constants

SiETUDConstants* Constants

Type: read

Description: Constants used by ETU-Driver

3.4 Methods

Note: all methods return S_OK to indicate successful execution, S_FALSE to indicate failure to succeed, and an error number in case an exception was raised.

Calibrate

HRESULT calibrate(void);

Description: Runs calibration routine. Execution stops here until calibration is complete.

Arguments: none.

Return: S_OK if succeeded, S_FALSE otherwise.

Stop calibration

HRESULT stopCalibration(void);

Description: Stops calibration routine.

Arguments: none.

Return: S_OK if succeeded, S_FALSE otherwise.

Correct drift

```
HRESULT correctDrift(void);
```

Description: Runs calibration drift correction routine. Execution stops here until calibration is complete.

Arguments: none.

Return: S_OK if succeeded, S_FALSE otherwise.

Start tracking

```
HRESULT startTracking(void);
```

Description: start tracking routine. If the drift correction flag at start is on, calls correctDrift functions, after its finish starts tracking

Arguments: none.

Return: S_OK, if succeeded, S_FALSE otherwise.

stopTracking

```
HRESULT stopTracking(void);
```

Description: stop tracking routine.

Arguments: none.

Return: S_OK, if succeeded, S_FALSE otherwise.

passValue

```
HRESULT passValue(EiETUDPassingValue aValueType /*[in]*/,  
void* aValue /*[in]*/);
```

Description: passes some application-dependant values to converters, that may require it. ETU-Driver passes all types of values listed in EiETUDPassingValue enumeration when a new converter is loaded. Normally, applications should not use this function unless these developers know that they do, or a converter requires some unspecified in EiETUDPassingValue enumeration value.

Arguments:

aValueType – a type of passing value (see EiETUDPassingValue enumeration);

aValue – the pointer to a passing value;

Return: S_OK, if succeeded, S_FALSE otherwise.

parseDriverControlEvents

```
HRESULT parseDriverControlEvent(  
EiETUDDeviceControlMessages aControlID /*[in]*/,  
void* aData /*[in,out]*/);
```

Description: use this function as an alternative way of setting and getting some properties, or execute function not available directly from the COM interface. The list of possible control messages is in EiETUDDeviceControlMessages enumeration

Arguments:

aControlID – a type of ETU-Driver control message (see EiETUDDeviceControlMessages enumeration);

aData – the pointer to a data value, which depends on a type of control message. Some types of messages change the value in *aData*;

Return: S_OK, if succeeded, S_FALSE otherwise.

showChooseDriver

```
HRESULT showChooseDriver(void);
```

Description: show a dialog to choose a driver/converter

Arguments: none;

Return: S_OK, if succeeded, S_FALSE otherwise.

showRecordingOptions

```
HRESULT showRecordingOptions(void);
```

Description: show a dialog to view/edit recording options

Arguments: none;

Return: S_OK, if succeeded, S_FALSE otherwise.

showDeviceOptions

```
HRESULT showDeviceOptions(void);
```

Description: show a dialog to view/edit device options

Arguments: none

Return: S_OK, if succeeded, S_FALSE otherwise.

showScreenRecordingOptions

```
HRESULT showScreenRecordingOptions(void);
```

Description: show a dialog to view/edit screen recording options

Arguments: none;

Return: S_OK, if succeeded, S_FALSE otherwise.

showRecordingSetupOptions

```
HRESULT showRecordingSetupOptions(void);
```

Description: show a dialog to view/edit other recording and setup options

Arguments: none;

Return: S_OK, if succeeded, S_FALSE otherwise.

showGazeParserOptions

```
HRESULT showGazeParserOptions(void);
```

Description: show a dialog to view/edit gaze parser options;

Arguments: none;

Return: S_OK, if succeeded, S_FALSE otherwise.

addFixation

```
HRESULT addFixation(SiETUDFixation* aFix /*[in]*/);
```

Description: manually add a fixation to the fixations file while tracking is on. Do not add fixations coming through onDataEvent event – ETU-Driver does it automatically

Arguments:

aFix – pointer to a fixation;

Return: S_OK, if succeeded, S_FALSE otherwise.

addSample

```
HRESULT addSample(SiETUDSample* aSmp /*[in]*/);
```

Description: manually add a sample to the samples file while tracking is on. Do not add samples coming through onDataEvent event – ETU-Driver does it automatically

Arguments:

aSmp – pointer to a sample;

Return: S_OK, if succeeded, S_FALSE otherwise.

addBlink

```
HRESULT addBlink(SiETUDBlink* aBlk /*[in]*/);
```

Description: manually add a blink to the blinks file while tracking is on. Do not add blinks coming through onDataEvent event – ETU-Driver does it automatically

Arguments:

aBlk – pointer to a blink;

Return: S_OK, if succeeded, S_FALSE otherwise.

addExtraEvent

```
HRESULT addExtraEvent(long aType /*[in]*/,
LPSTR aData /*[in]*/);
```

Description: add a system or environment event to the extra events file while tracking is on. Use this function always you need to save such an event into a file. Do not call this function to add images captured by captureImage function – ETU-Driver does it automatically..

Arguments:

aType – type of event. Can be a value from EiETUDStimulus enumeration in the second byte, a value from EiETUDSystemEvents enumeration in first (lowest) byte, or any other custom value). For example, the function captureImage calls this function with *aType* = stImages << 8 (0x100);

aData – a pointer to a string that contains event-related data. For example, the function captureImage calls this function with *aData* set to the pointer to the string with image file name;

Return: S_OK, if succeeded, S_FALSE otherwise.

stringToFix

```
HRESULT stringToFix(SiETUDFixation* aFix /*[in,out]*/,
LPSTR aStr /*[in]*/);
```

Description: parses an ASCII string that was read from a fixations' file to fill the fixation's structure;

Arguments:

aFix – pointer to a structure that receives fixation data;

aStr – a pointer to a string that contains fixation data in ETU-Driver ASCII format;

Return: S_OK, if succeeded, S_FALSE otherwise.

stringToSmp

```
HRESULT stringToSmp(SiETUDSample* aSmp /*[in,out]*/,
LPSTR aStr /*[in]*/);
```

Description: parses an ASCII string that was read from a samples' file to fill the sample's structure;

Arguments:

aSmp – pointer to a structure that receives sample data;

aStr – a pointer to a string that contains sample data in ETU-Driver ASCII format;

Return: S_OK, if succeeded, S_FALSE otherwise.

stringToBlk

```
HRESULT stringToBlk(SiETUDBlink* aBlk /*[in,out]*/,
LPSTR aStr /*[in]*/);
```

Description: parses an ASCII string that was read from a blinks' file to fill the blink's structure;

Arguments:

aBlk – pointer to a structure that receives blink data;

aStr – a pointer to a string that contains blink data in ETU-Driver ASCII format;

Return: S_OK, if succeeded, S_FALSE otherwise.

stringToExE

```
HRESULT stringToExE(SiETUExtraEvent* aExE /*[in,out]*/,
LPSTR aStr /*[in]*/);
```

Description: parses an ASCII string that was read from a system and environment events' file to fill this event's structure;

Arguments:

aExE – pointer to a structure that receives system and environment event data;

aStr – a pointer to a string that contains system and environment event data in ETU-Driver ASCII format;

Return: S_OK, if succeeded, S_FALSE otherwise.

readSetupData

```
HRESULT readSetupData(LPSTR aFileName /*[in]*/, SiETUDRecordingSetupData*
aSetupData /*[in,out]*/, SiETUDRecordingOptions* aRecordingOptions
/*[in,out]*/);
```

Description: read the file with gaze and other data recording and setup options saved by ETU-Driver (*.stp files) into the corresponding structures;

Arguments:

aFileName – pointer to a string that contains setup file name;

aSetupData – a pointer to a buffer that receives other recording options and setup;

aRecordingOptions – a pointer to a buffer that receives gaze data recording options;

Return: S_OK, if succeeded, S_FALSE otherwise.

captureImage

```
HRESULT captureImage(LPSTR aName /*[in]*/, void* aWnd /*[in]*/, TOLEBOOL
aSaveInJpeg /*[in]*/,
signed_char aJpegQuality /*[in]*/);
```

Description: capture image of desktop or a particular window, save it into BMP or JPEG file and add an environment event (calls *addExtraEvent* function with *aType* = *stImages* << 8);

Arguments:

aName – pointer to the string with an image name (without extension);

aWnd – a handle of a window to capture. If NULL, the function capture the whole screen (desktop);

aSaveInJpeg – if TRUE, function saves the image captures in JPEG format, other wise in BMP format;

aJpegQuality – a value from 1 to 100 to set the quality of image saved if JPEG format. The value is ignored if *aSaveInJpeg* is FALSE;

Return: S_OK, if succeeded, S_FALSE otherwise.

getVersion

```
HRESULT getVersion(TOLEBOOL aShowAboutDialog /*[in]*/,  
LPSTR* aBuffer /*[out]*/);
```

Description: show and receive version of ETU-Driver installed;

Arguments:

aShowAboutDialog – if TRUE, show *About* dialog with ETU-Driver version;

aBuffer – a pointer to a buffer that receives ETU-Driver version in ASCII string format (can be NULL to);

Return: S_OK, if succeeded, S_FALSE otherwise.

3.5 Events

OnRecordingStart

```
void OnRecordingStart();
```

Description: fires when actual recording starts;

Arguments: none;

OnRecordingStop

```
void OnRecordingStop();
```

Description: fires when actual recording stops, but before ETU-Driver parses the data collected (if any);

Arguments: none;

OnRecordingSaved

```
void OnRecordingSaved(TOLEBOOL aSaved /*[in]*/);
```

Description: fires when ETU-Driver finishes parsing of data collected during recording. This event does not fire if data saving flag in recording options was off;

Arguments:

aSaved – TRUE if data was saved, FALSE otherwise (a user cancelled data saving after recording was stopped);

OnCalibrated

```
void OnCalibrated();
```

Description: fires when calibration is finished or terminated;

Arguments: none;

OnDataEvent

```
void OnDataEvent(EiETUDGazeEvents aEventID /*[in]*/,
long* aData /*[in]*/, long* aResult /*[in,out]*/);
```

Description: fires when a new data arrives from device or gaze event detector;

Arguments:

aEventID – type of event (see EiETUDGazeEvents enumeration);

aData – pointer to a data structure that depends on event type. Cast *aData* to

SiETUDSample* if *aEventID* is geSample,

SiETUDFixation* if *aEventID* is geFixationStart, geFixationEnd or geFixationUpdate;

SiETUDBlink* if *aEventID* is geBlinkStart or geBlinkEnd;

OnAppControlEvent

```
void OnAppControlEvent(
EiETUDAppControlMessages aMessageID /*[in]*/,
long* aData /*[in,out]*/, long* aResult /*[in,out]*/);
```

Description: fires when ETU-Driver requires some data or notifies or action from a parent application;

Arguments:

aMessageID – type of control message (see EiETUDAppControlMessages enumeration);

aData – pointer to a data that depends on message type (see notes below);

aResult – set the value at this pointer to 0 to indicate failure, and a positive one to indicate an execution success. The exact return value may depend on message type (see notes below);

Notes:

If *aEventID* is *acmToggleFullScreen*, cast *aData* to BOOL. An application should set window or full-screen mode when this message arrives. Set **aResult* value to 0 (or leave without changes) to indicate that the message was ignored, or to 1 otherwise;

If *aEventID* is *acmKeyPress*, cast *aData* to WORD. An application may proceed key-stroke (“virtual key” value in *aData*) sent to windows opened by ETU-Driver. Set **aResult* value to 0 (or leave without changes) to indicate that the message was ignored, or to 1 otherwise;

If *aEventID* is *acmGetProfileName* or *acmGetProfileDir*, cast *aData* to LPSTR. An application should copy the profile name or profile's folder to the *aData* (eg., `strcpy(aData, ProfileName)`), if application supports data profiles and want to save subject-related data in profile folder (for example, Tobii converter asks for profiles to store calibration files). Set **aResult* value to 0 (or leave without changes) to indicate that the message was ignored, or to a required buffer size otherwise. If *aData* is NULL, only the size of a required buffer to store string must be set.

OnSynchronization

```
void OnSynchronization(unsigned_long* aTimeHi /*[in,out]*/,  
unsigned_long* aTimeLo /*[in,out]*/);
```

Description: fires when a new data arrives from device (but before *OnDataEvent* fires). Application may set timestamps to be saved with data instead of timestamps provided by manufacturers timer. Current version (1.02) do not fire this event, it is reserved for future;

Arguments:

aTimeHi – high 4 bytes of timestamp;

aTimeLo – low 4 bytes of timestamp;

3.6 Summary

This ends the description of the COGAIN ETSD, its installation, description, testing with the basic COGAIN test application, and the COGAIN ETSD programmers reference. The next section shows the ETSD working with a third party application, 'Eye Chess'.

4 The COGAIN standard driver playing Eye Chess

4.1 The chess program

The first application that uses the COGAIN ETSD to allow plug-and-play swapping of eye tracking systems with the same application has now been written. This is a chess program specifically designed to be easy to use with eye gaze control. Figure 31 shows the chess program ready to use, and Figure 32 shows a player controlling the chess game via eye gaze and the COGAIN standard driver.



Figure 31. The 'Eye Chess' program using the COGAIN ETSD – ready to play

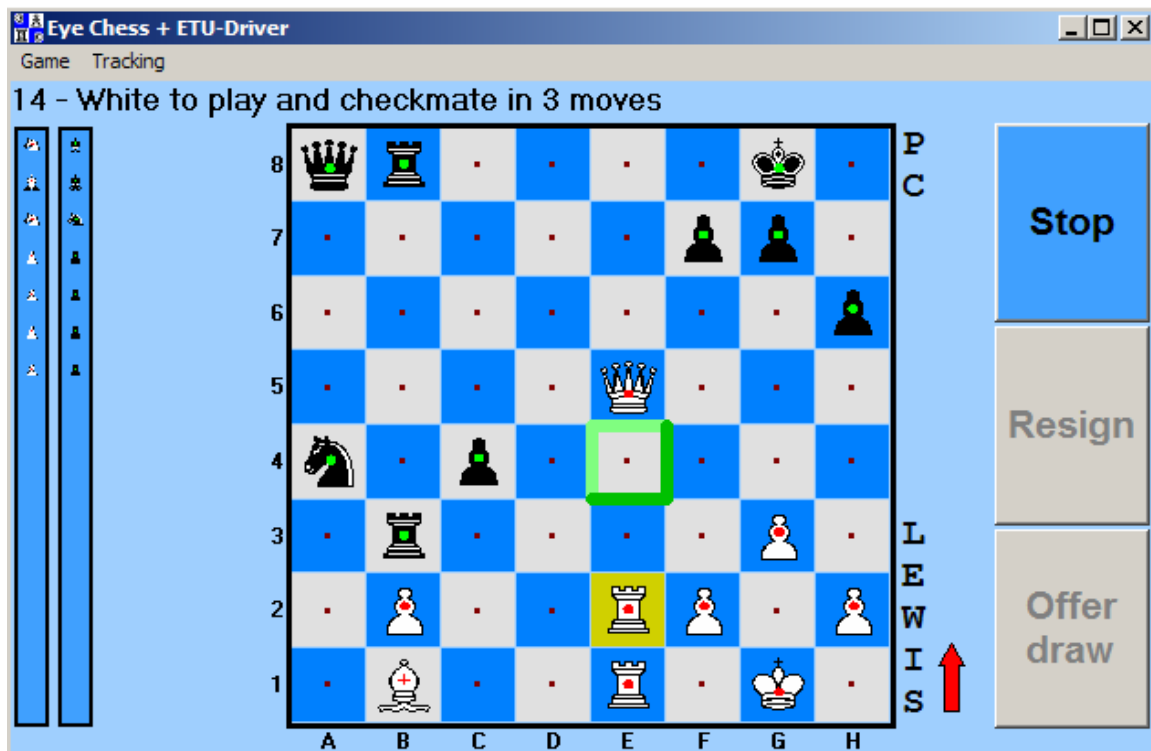


Figure 32. The 'Eye Chess' program using the COGAIN ETSD – game playing

4.2 Using the COGAIN ETSD

The developer of the chess program has incorporated the COGAIN ETSD into their application, and this has proved the same options for choosing and setting up different eye tracker drivers. This is accessed via the 'Tracking' menu item, Figure 33.

Finally, proof of the use of the COGAIN standard driver can be found by accessing the 'Tracking' and then 'About Driver' options on the third party application, Figure 35. The application then shows the COGAIN driver information, Figure 36.

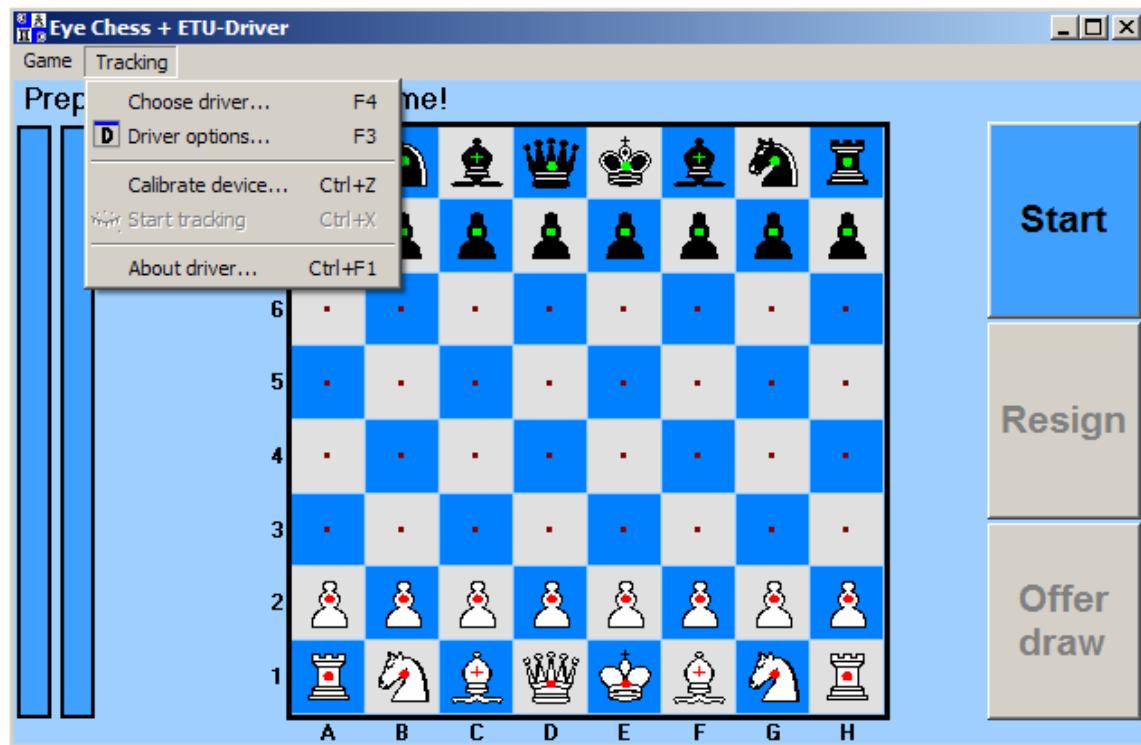


Figure 33. The 'Eye Chess' program and configuring the COGAIN ETSD step 1



Figure 34. The 'Eye Chess' program and configuring the COGAIN ETSD step 2



Figure 35. The 'Eye Chess' program showing the COGAIN ETSD step 1



Figure 36. The 'Eye Chess' program showing the COGAIN ETSD step 2

This ends the description of the COGAIN ETSD.

5 Conclusions

This deliverable has presented the first version of the COGAIN Eye-Tracking Standard Driver (COGAIN ETSD). The deliverable represents the first standardised driver for interchangeable or 'plug-and-play' eye tracking systems, and is the first major step toward practical standardisation of eye-gaze communication systems.

There is still considerable work to be undertaken developing the driver further, both in terms of ease of use, and in terms of functionality. These will be addressed in a future review of the driver after trials by third parties.

Once feedback is received, and agreement found, a subsequent deliverable will be planned that will update and formalise the driver from a prototype form, to a form suitable for standardisation.