

5DT Data Glove Ultra Series



User's Manual

5DT

Fifth Dimension Technologies

www.5DT.com

5DT Data Glove Ultra

User's Manual

Copyright © 2011 5DT (Fifth Dimension Technologies), All rights reserved. No part of this manual, software or hardware may be copied, reproduced, translated or reduced to any electronic medium or machine readable format without the written consent of 5DT (Fifth Dimension Technologies).

All trademarks and trade names are the respective property of their owners.

Version 1.3

January 2011

Table of Contents

| | |
|--|----|
| 1. Introduction..... | 1 |
| 2. Setup and Installation..... | 3 |
| 2.1. 5DT Data Glove Ultra..... | 3 |
| 2.1.1 Package Contents..... | 3 |
| 2.1.2 Connecting the Glove..... | 3 |
| 2.2. 5DT Data Glove Ultra Wireless Kit (Optional Accessory)..... | 4 |
| 2.2.1 Package Contents..... | 4 |
| 2.2.2 Connecting the Wireless System..... | 5 |
| 2.2.3 Charge the Battery Pack..... | 5 |
| 2.2.4 Connect the Gloves..... | 6 |
| 2.2.5 Connect the Receiver..... | 7 |
| 2.2.6 Starting the System..... | 7 |
| 2.2.7 Positioning the Receiver..... | 7 |
| 3. Software Installation..... | 9 |
| 3.1. CD Contents..... | 9 |
| 3.1.1 Installed Software..... | 9 |
| 4. Using GloveManager..... | 11 |
| 4.1. Starting GloveManager..... | 11 |
| 4.2. Opening USB Gloves..... | 11 |
| 4.2.1 Scanning for Gloves..... | 11 |
| 4.2.2 Opening Gloves..... | 12 |
| 4.3. Opening Serial and Wireless Gloves..... | 14 |
| 4.3.1 Scanning for Gloves..... | 14 |
| 4.3.2 Opening Gloves..... | 14 |
| 4.4. The Graph View..... | 16 |
| 4.4.1 The Menu Bar..... | 17 |
| 4.4.2 Toolbar..... | 18 |
| 4.4.3 Tab Bar..... | 18 |
| 4.4.4 The Sensor Graphs..... | 19 |
| 4.4.5 The Sensor Info Box..... | 20 |
| 4.4.6 The Status Bar..... | 22 |
| 4.5. The Sensor Selection View..... | 22 |
| 4.6. Glove Calibration..... | 23 |
| 4.7. Recording..... | 25 |

| | |
|--|----|
| 5. Using the Gloves in Autodesk MotionBuilder..... | 27 |
| 5.1. Installation | 27 |
| 5.2. Opening the Glove | 27 |
| 5.3. Opening Wireless Gloves..... | 28 |
| 5.4. Calibration | 29 |
| 5.5. Model Binding | 29 |
| 6. Using the SDK..... | 33 |
| 6.1. Introduction | 33 |
| 6.2. Using the SDK..... | 33 |
| 6.2.1 Microsoft Visual Studio C/C++ | 33 |
| 6.2.2 Microsoft Visual Studio C#..... | 34 |
| 6.2.3 Linux C/C++..... | 35 |
| 6.2.4 Mac OS X C/C++ | 38 |
| 6.3. Connecting to the second glove when using a wireless kit..... | 40 |
| 6.4. Supported Gloves | 40 |
| 6.5. Sensor Mappings for the 5DT Data Glove 5 and 5 Ultra | 41 |
| 6.6. Sensor Mappings for the 5DT Data Glove 14 Ultra | 41 |
| 6.7. Gesture Definitions..... | 43 |
| 6.8. Auto-Calibration | 45 |
| 7. Troubleshooting and Support | 47 |
| 7.1. General Troubleshooting..... | 47 |
| 7.1.1 Physical Connection Problems | 47 |
| 7.1.2 Software Connection Problems | 47 |
| 7.2. Frequently Asked Questions (FAQ)..... | 47 |
| 7.3. Support..... | 48 |
| 8. Warranty Information | 49 |
| 8.1. Thirty (30) Day Customer Satisfaction Guarantee..... | 49 |
| 8.2. 6 Months Product Warranty..... | 49 |
| 8.3. Exclusions | 49 |
| 8.4. Warranty Claim Instructions | 49 |
| 9. About 5DT | 51 |
| Appendix A – Hardware Specifications | 53 |
| Appendix B – Serial Protocol..... | 55 |
| Appendix C – SDK Function Descriptions | 57 |
| Appendix D – C# SDK Function Descriptions..... | 71 |

1. Introduction

The 5DT Data Glove Ultra is a hand data motion capturing solution for animation and virtual reality.

The 5DT Data Glove Ultra is the second generation of 5DT's high-end data glove and has been completely redesigned and optimized for the best performance and ease of use.

The 5DT Data Glove Ultra 14 measures finger flexure as well as the abduction between the fingers. Finger flexure is measured at two places (1st joint (knuckle), 2nd joint) on each finger.

The 5DT Data Glove Ultra 5 measures average finger flexure for each of the 5 fingers.

Among the features of this glove are:

- The glove is comfortable with an improved design for differently-sized hands.
- The accurate and sensitive sensors give a clean signal, minimizing the need for additional filtering.
- Diagnostic software is included with the capability to record hand data.
- The package optionally includes a plug-in for Autodesk MotionBuilder.
- The gloves' functions and data are accessible via the 5DT Data Glove SDK.
- The glove data stream from the optional serial interface kit is an RS-232 data stream with an open protocol allowing for various platform implementations.

The optional 5DT Data Glove Ultra Wireless Kit is a wireless hand data motion capturing solution for animation and virtual reality. The kit is a wireless transmission system for the 5DT Ultra Series of Data Gloves.

The 5DT Data Glove Ultra Wireless Kit allows you to capture the data from both hands and transmit them over a single wireless link. The complete hand data (from both hands) can be captured and transmitted wirelessly for up to 4 motion capture actors at a time.

2. Setup and Installation

2.1. 5DT Data Glove Ultra

2.1.1 Package Contents

When first unpacking your glove, please ensure that all the parts are there. If anything is missing please contact 5DT or your reseller immediately.

The 5DT Data Glove Ultra consists of the following:

1. 5DT Data Glove Ultra (5 or 14 Sensor)
2. 5DT Data Glove Ultra Series USB cable
3. 5DT Data Glove CD

2.1.2 Connecting the Glove

Connect the glove to the PC as shown in Figure 1.

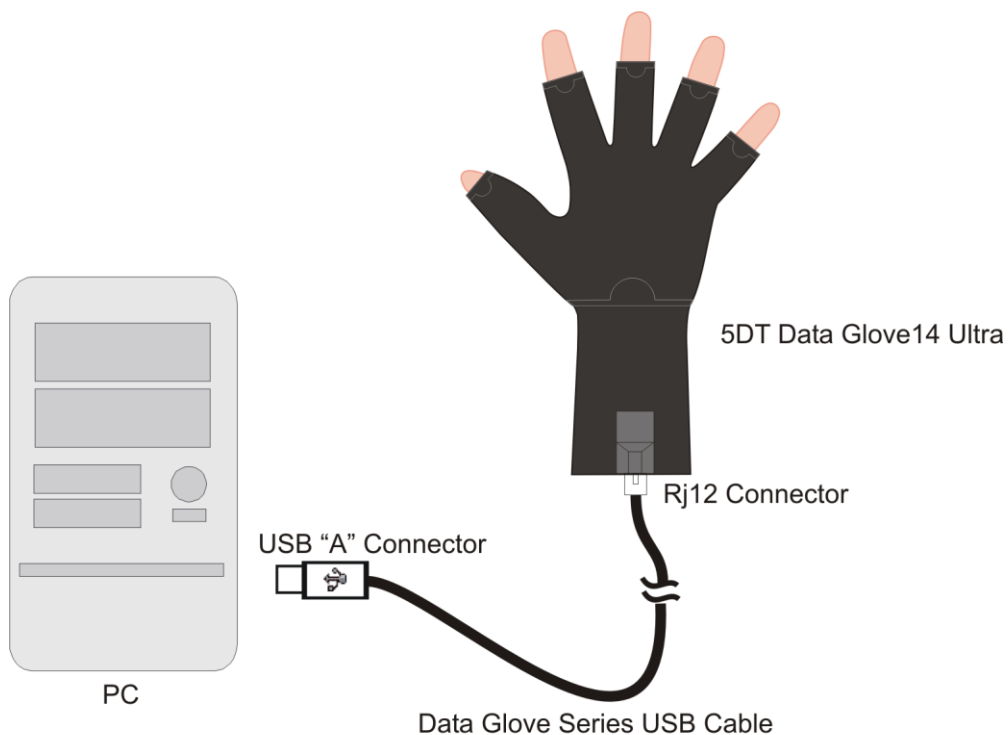


Figure 1 – Connecting the Glove to the PC.

You can use GloveManager to test the operation of the glove. Detail of this is given in Section 4 of this manual.

2.2. 5DT Data Glove Ultra Wireless Kit (Optional Accessory)

2.2.1 Package Contents

When first unpacking your gloves and wireless kit, please ensure that all the parts are there. If anything is missing please contact 5DT or your reseller immediately.



Figure 2 – Items needed for setting up the 5DT Data Glove 14 Ultra Wireless Kit

The items needed to setup the wireless kit are (see Figure 2):

1. 5DT Data Glove 14 Ultra (Left-handed)¹
2. 5DT Data Glove 14 Ultra (Right-handed)¹
3. 5DT Wireless Data Transmitter²
4. 5DT Wireless Data Receiver²
5. Battery Pack (x2) ²
6. Battery Charger²
7. Wireless Belt Plate and Belt²
8. 5V Power Supply Adapter²
9. Wireless Interface Cables (x2) ²
10. Wireless Data Receiver Cable²

11. Battery Charger Adapter Cable (Only with older charger models) ²

12. 5DT Data Glove CD (supplied with Data Gloves)

13. Glove Dongle (not shown) ³

¹ Please note Data Gloves sold separately

² Supplied with Data Glove Ultra Wireless Kit

³ The Glove Dongle is an RJ12 connector with a short pigtail connection. When only one glove is used at a time, please insert this dongle into the unused slot of the transmitter.

2.2.2 Connecting the Wireless System

Connecting the wireless systems will be described in three parts, namely charging the battery, attaching the glove side and attaching the receiver side.

Please read this section carefully

The numbers in the text refer to the numbers in Figure 2.

2.2.3 Charge the Battery Pack

Before you start using the gloves, you'll need to charge the battery pack.

You'll need the following parts:

- Battery Charger Adapter Cable
- Battery Pack
- Battery Charger

Figure 3 shows how to connect the battery charger (6) to the battery pack (5) using the battery charger cable adapter (11).

After the battery has charged for 4-6 hours, the battery charger will indicate that the pack is charged and ready to use. This indication will usually be in the form of a green LED, but may vary according the model of battery charger supplied for your region.

For rapid testing purposes, you may partially charge batteries as they contain Li-Ion cells and do not have the 'memory effect' associated with NiCad rechargeable batteries.

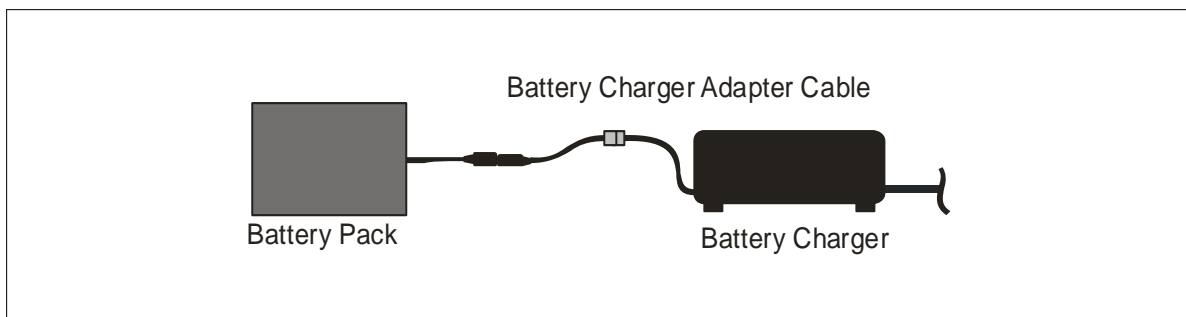


Figure 3 – Connecting the Wireless Kit – Charging the battery

2.2.4 Connect the Gloves

The glove side consists of all the parts that connect to the user:

- 5DT Data Glove 5 or 14 Ultra (Left-handed)
- 5DT Data Glove 5 or 14 Ultra (Right-handed)
- Wireless Interface Cables
- 5DT Wireless Data Transmitter
- Battery Pack
- Wireless Belt Plate & Belt

First attach the transmitter (3) and battery pack (5) to the belt plate (7). Next, plug in the cable from the Battery Pack to the Data Transmitter. You should see a green light on the top face of the transmitter. If this light doesn't come on, make sure that your battery pack is charged.

Plug both gloves (1) & (2) into the transmitter (3) using the two wireless interface cables (9) as in Figure 4.

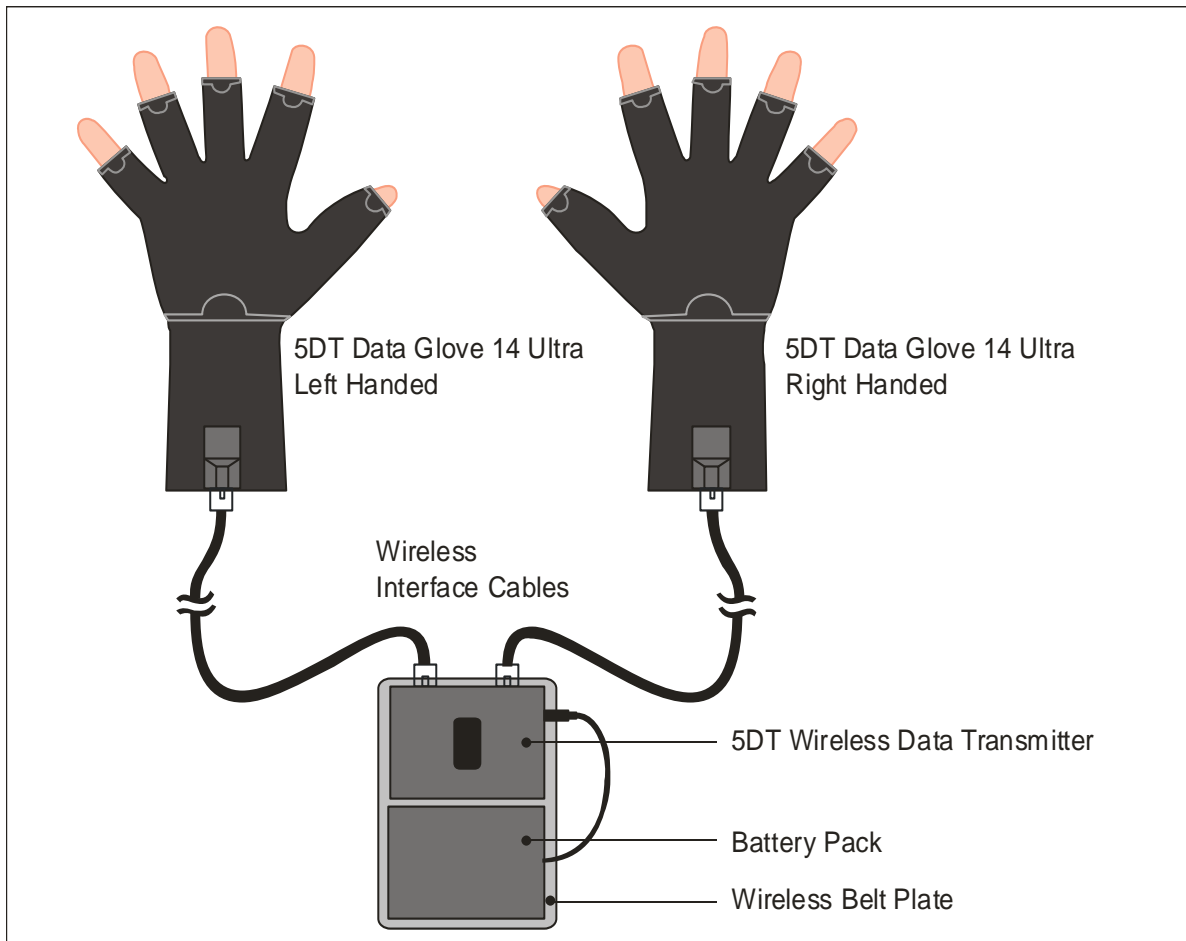


Figure 4 – Connecting the Wireless Kit – Glove Side

2.2.5 Connect the Receiver

To set up the receiver, you'll need the following parts:

- 5DT Wireless Data Receiver
- Wireless Receiver Cable
- 5V Power Supply Adapter

Connect the receiver (4) to the receiver cable (10) and connect the power supply (8) to the receiver cable as in Figure 5. Plug the female end of the receiver cable into an available COM port on the PC and plug in a standard PC power plug into the power supply.

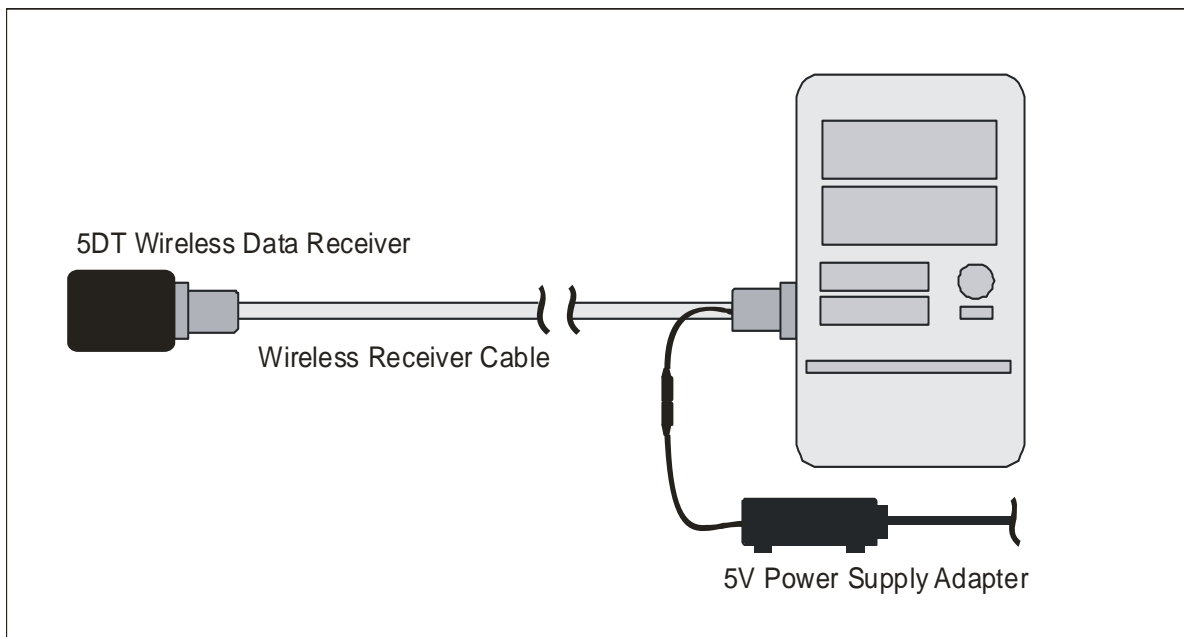


Figure 5 – Connecting the Wireless Kit – Receiver Side

2.2.6 Starting the System

Once everything is connected, switch the power on by connecting power to the power supply and turning on the switch on top of the wireless data transmitter.

The green light on the transmitter means the power is connected and the blue light indicates that a wireless connection is present. Once the power is applied, it may take a few seconds for the connection to be made.

You can then use GloveManager to test the operation of the gloves. Detail on this is given in Section 4 of this manual.

2.2.7 Positioning the Receiver

In order to get the best wireless performance, it is important to get the wireless data receiver as close to the transmitter as possible. For this purpose the wireless receiver cable has been made especially long to get the receiver into a convenient location.

3. Software Installation

Your package comes with a CD that contains documentation and utilities to help you get up and running with the gloves as quickly as possible.

Most of the supplied programs and software are intended for 32-bit Windows, and require Windows XP, Vista or Windows 7. The SDK and applications have been tested to work on 64-bit versions of these Operating Systems.

Please check the www.5dt.com website for the latest versions of the drivers, as new drivers and new versions are regularly added

The setup program should run when you insert the CD, but if it doesn't, run the program **setup.exe** in the **install** directory and follow the on-screen instructions.

3.1. CD Contents

The CD contains the following directories:

...\Acrobat

This copy of Adobe® Acrobat® Reader will allow you to read the documentation.

...\Install

This contains the software installation files.

...\Linux

This folder contains the SDK and code examples for Linux users. The SDK for Linux is described in detail in Section 6.2.3.

...\Mac

This folder contains the SDK and code examples for Mac users. The SDK for Mac is described in detail in Section 6.2.4.

...\Windows\SDK

This folder contains the SDK and code examples for Windows users. The SDK is described in detail in Section 6.

...\Windows\Plugins

This directory contains the plug-ins for MotionBuilder 2010 and 2011. Section 5 gives more detail about the plug-ins.

3.1.1 Installed Software

By default the setup program will install your files to the following directory:

C:[Program Files]5DT\Data Glove Ultra

The installation contains the following directories:

...\GloveDemo

This folder contains the 3D glove demo. Click on **5DT Glove Demo.exe** to start the demo.

...\Documents

This directory contains the electronic copy of this manual in pdf format and application notes as they become available.

...\GloveManager

GloveManager is a program that allows you to test your gloves and access advanced glove functions which may not be available in the plug-ins. More detail on this is given in Section 4.

4. Using GloveManager

The 5DT GloveManager is a useful tool accompanying the Data Glove which may be used for the following:

- Testing the 5DT Data Glove
- Obtaining good calibrated values for the 5DT Data Glove
- Logging data obtained from the 5DT Data Glove

Note: All images are shown for the Data Glove 14 Ultra. Images will differ slightly for the Data Glove 5 Ultra.

4.1. Starting GloveManager

Start the program by running GloveManager.exe, which is installed in the **\Data Glove Ultra\GloveManager** folder by default. The setup tab window will appear (see below).

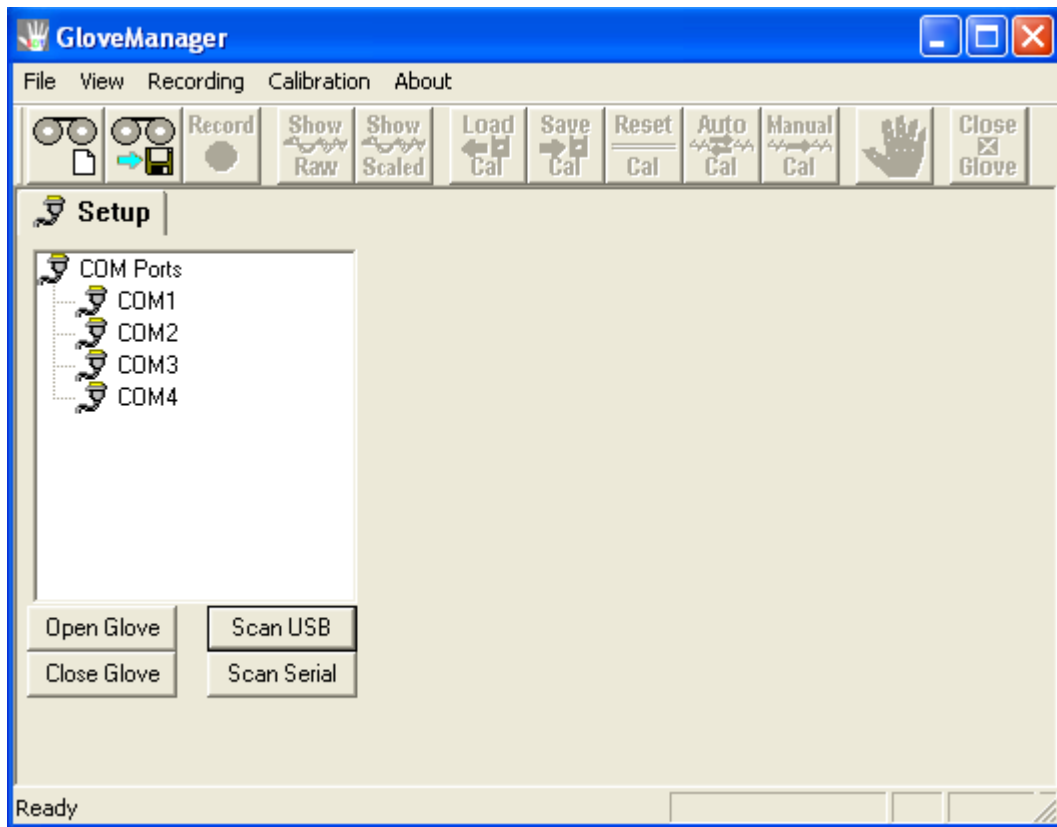


Figure 6 – The Setup Tab Window

4.2. Opening USB Gloves

4.2.1 Scanning for Gloves

Clicking the **Scan USB** button in the setup tab window causes GloveManager to scan the USB for available gloves. This is useful if you have plugged in the glove after starting GloveManager. Once the USB scan is complete the USB device tree will be updated to show available gloves.

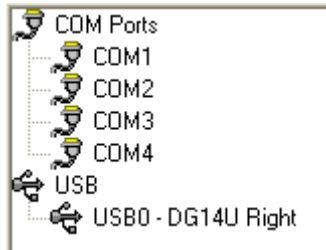



Figure 7 – Result of the USB Scan

As can be seen in Figure 7, the Port Scan has found a glove on USB0.

4.2.2 Opening Gloves

A glove may be opened by clicking the  button after a USB glove has been selected in the device tree.

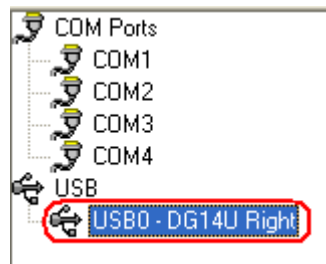


Figure 8 – Selecting the Glove to be Opened

If the glove has been successfully opened the tab window will automatically switch to the newly opened glove, as shown in Figure 9. Double-clicking on a USB glove will also cause GloveManager to attempt to open the glove.

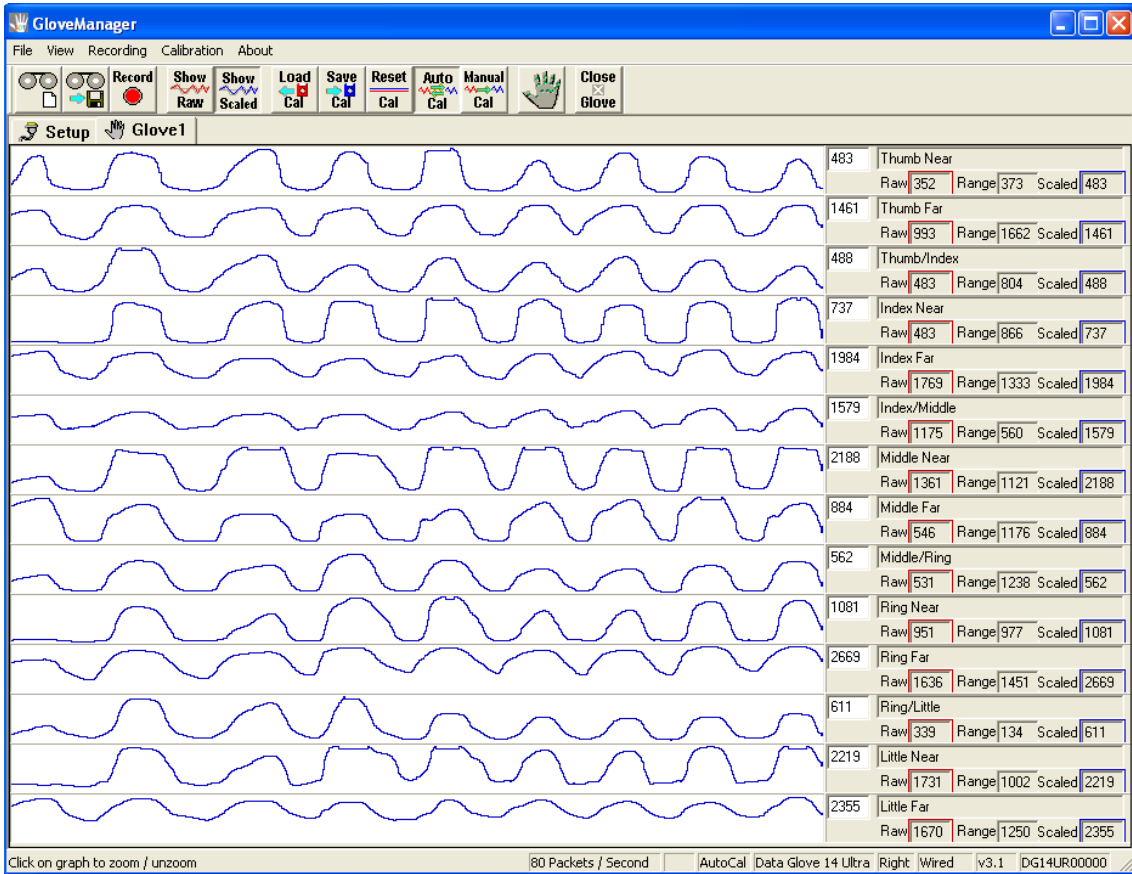
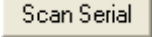


Figure 9 – GloveManager after the USB Glove is Successfully Opened

Clicking on the **Setup** tab will cause GloveManager to display the setup tab window again. To open the second wireless glove, select it from the COM port tree.

4.3. Opening Serial and Wireless Gloves

4.3.1 Scanning for Gloves

Clicking the  button in the setup tab window causes GloveManager to scan the COM ports for available gloves. This is useful if you do not know on which COM port your glove is connected to, or you simply want to scan for available gloves. Once the COM port scan is complete the COM port tree will be updated to show available gloves.

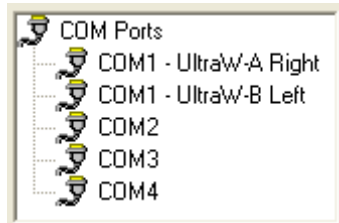



Figure 10 – Result of the Port Scan

As can be seen in Figure 10, the Port Scan has found two wireless gloves on COM1. The right hand glove is plugged into Port A of the wireless transmitter unit. The left hand glove is plugged into Port B.

4.3.2 Opening Gloves

A glove may be opened by clicking the  button after a COM port has been selected in the COM port tree.

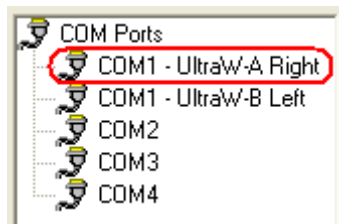


Figure 11 – Selecting the First Glove to be Opened

If the glove has been successfully opened the tab window will automatically switch to the newly opened glove, as shown in Figure 12. Double-clicking on a COM port will also cause GloveManager to attempt to open a glove on that COM port.

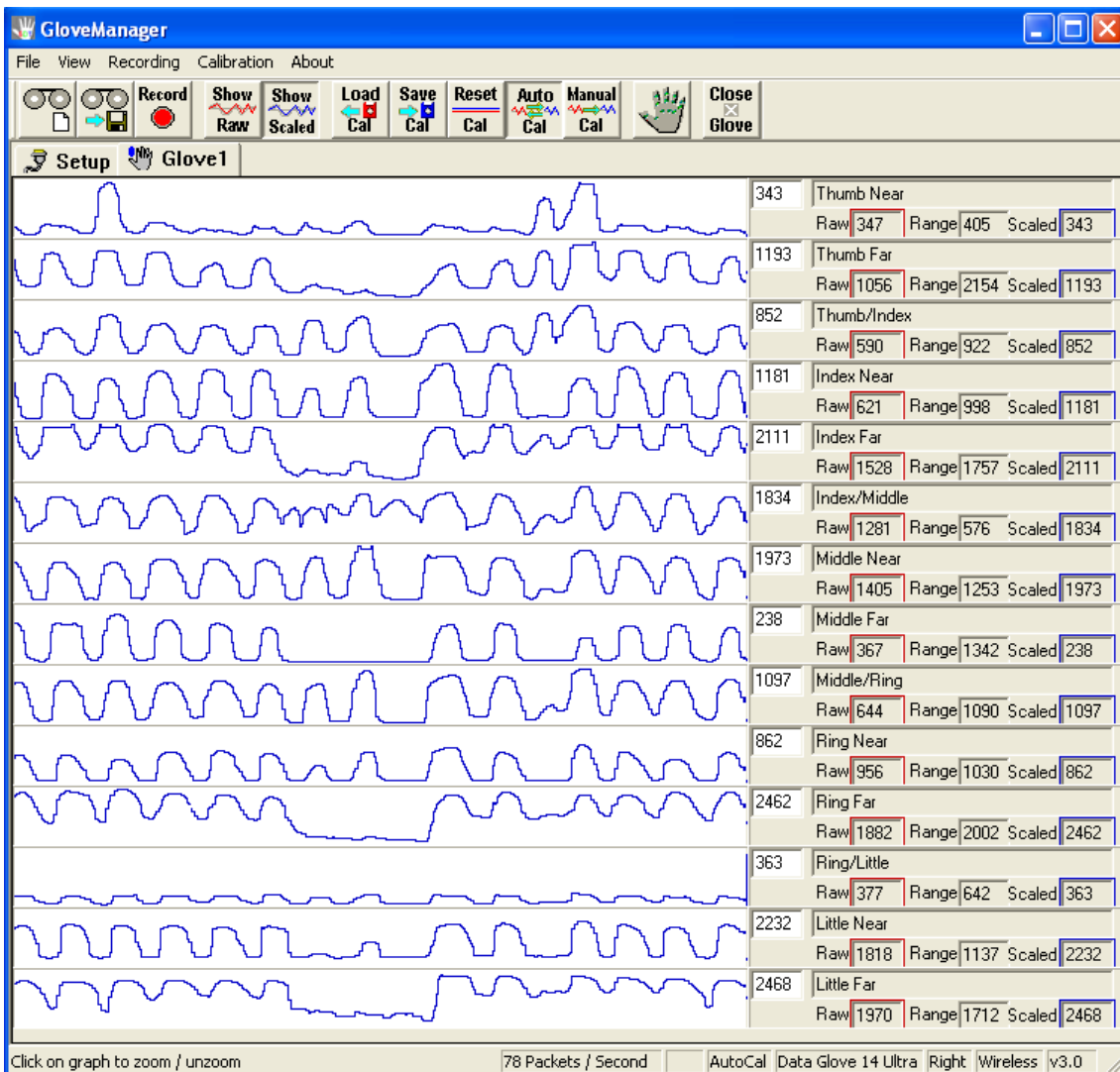


Figure 12 – GloveManager after the First Glove is Successfully Opened

Clicking on the **Setup** tab will cause GloveManager to display the setup tab window again. To open the second wireless glove, select it from the COM port tree.

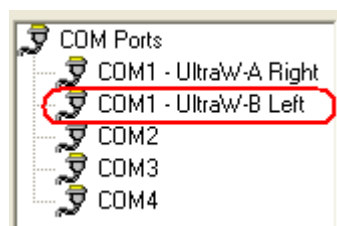


Figure 13 – Selecting the Second Glove to be Opened

Click on the **Open Glove** button again to open the second glove. If GloveManager was successful in opening the second glove, a new glove tab window will appear.



Figure 14 – The Tab Bar with Two Gloves Open

You may switch between open gloves by clicking on either of the Glove Tabs.

4.4. The Graph View

The following information fields are available in Graph View:

- Menu bar
- Toolbar
- Tab bar – Allows you to tab between gloves
- Sensor graph – A real-time graph of the glove sensor
- Sensor name and values – Displays the sensor name and the following real-time values (the amount of information displayed is dependent on the resolution available):
 - Raw sensor value [Value between 0 and 4095]
 - Sensor dynamic range [Value between 0 and 4095]
 - Scaled sensor value [*Scaled* value from 0 to 4095]
 - Lower calibrated value [Value between 0 and 4095]
 - Upper calibrated value [Value between 0 and 4095]
 - Graph top value [Value between 0 and 4095]
 - Current graph value [Value between 0 and 4095]
 - Graph bottom value [Value between 0 and 4095]
- Status bar – Displays status information, such as:
 - Program status and information messages
 - The packet rate
 - An indication if recording is currently on
 - An indication if auto calibration is currently on
 - The model of the current glove
 - The handedness of the current glove
 - An indication if the current glove is wired or wireless
 - The firmware version of the current glove

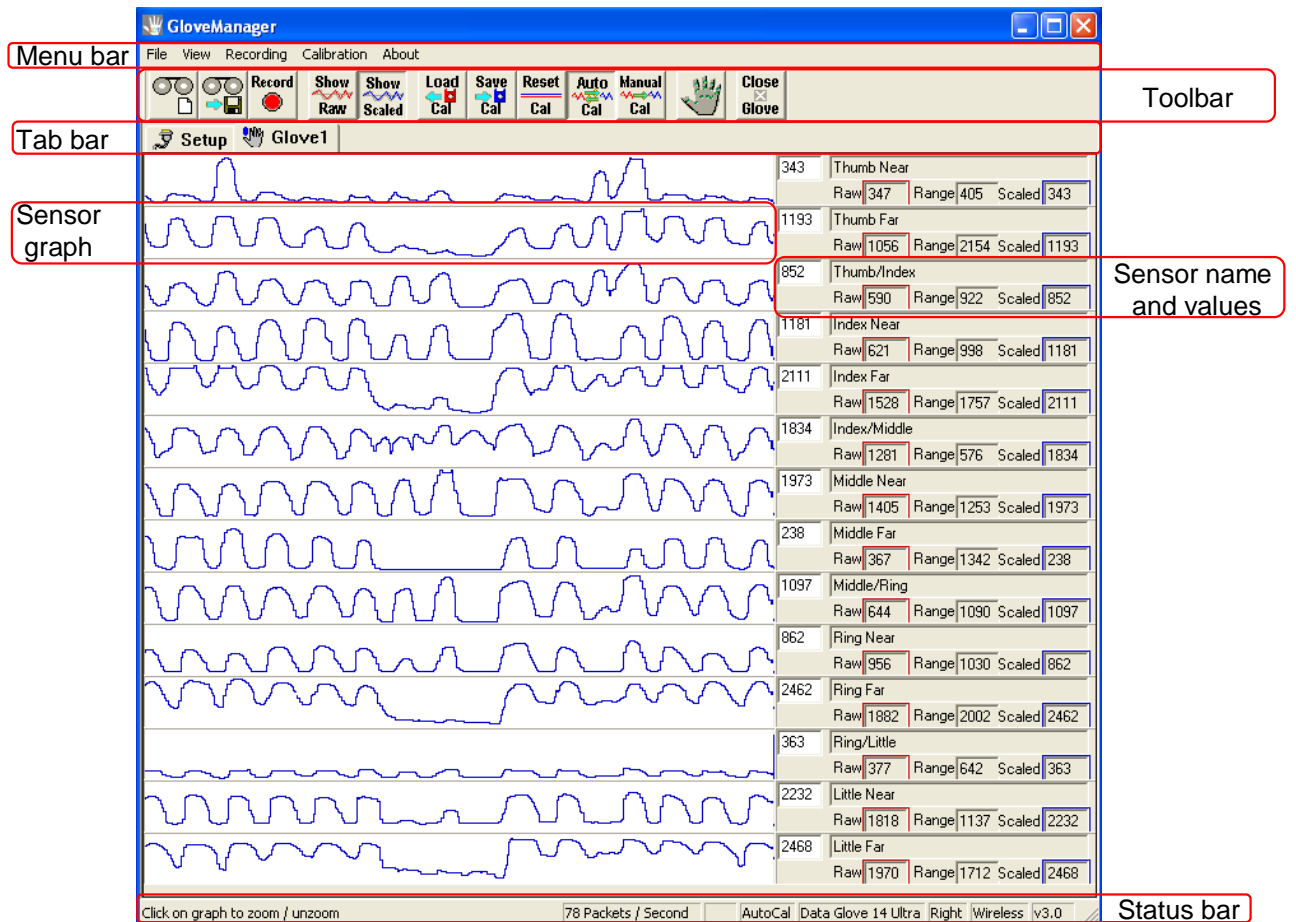


Figure 15 – Available Fields in Graph View

4.4.1 The Menu Bar

The following commands are available from the menu:

| Item | Shortcut | Description |
|----------------------------------|----------|---|
| File New Recording | Ctrl+N | Creates a new recording file |
| File Save Recording | Ctrl+S | Saves the current recording |
| File Save Recording As... | | Saves the current recording as a new file name |
| File Close Current Glove | Alt+C | Closes the currently visible glove |
| File Exit | | Exits GloveManager |
| View Toolbar | | Shows/Hides the Toolbar |
| View Status Bar | | Shows/Hides the Status Bar |
| View Show View Setup | V | Toggles between the sensor selection and the graph view |
| View Show Scaled Graphs | S | Shows/Hides the Scaled Graphs |
| View Show Raw Graphs | X | Shows/Hides the Raw Graphs |
| Recording Record | Ctrl+R | Toggles recording of glove data on or off |
| Calibration Auto Calibration | A | Toggles between auto and manual calibration |
| Calibration Manual Calibration | M | Toggles between auto and manual calibration |
| Calibration Reset Calibration | R | Resets the auto calibration values, and turns auto calibration on |

| Item | Shortcut | Description |
|---------------------------------------|----------|---|
| Calibration Load Calibration Values | Alt+O | Load the calibration values from file, turns auto calibration off |
| Calibration Save Calibration Values | Alt+S | Save the current calibration values to file |
| About About GloveManager... | | Displays information about GloveManager |

Table 1 – Menu Commands Available in GloveManager

4.4.2 Toolbar

The following buttons are available on the toolbar:













| Icon | Shortcut | Description |
|---|----------|---|
|  | Ctrl+N | Creates a new recording file |
|  | Ctrl+S | Saves the current recording |
|  | Ctrl+R | Toggles recording of glove data on or off |
|  | X | Shows/Hides the raw graphs |
|  | S | Shows/Hides the scaled graphs |
|  | R | Resets the auto calibration values, and turns auto calibration on |
|  | Alt+O | Load the calibration values from file, turns auto calibration off |
|  | Alt+S | Save the current calibration values to file |
|  | A | Toggles between auto and manual calibration |
|  | M | Toggles between auto and manual calibration |
|  | V | Toggles between the sensor view setup and the graph view |
|  | Alt+C | Closes the currently visible glove |

Table 2 – Buttons Available on the GloveManager Toolbar

Some items may be disabled when no glove is open.

4.4.3 Tab Bar

The tab bar allows you to switch between the Setup view and the currently open gloves. The blue dot on the tab icon indicates a wireless glove.



Figure 16 – The Tab Bar

4.4.4 The Sensor Graphs

The sensor graph is a graphical representation of the data from the sensors as it arrives. Two graph representations can be shown:

The scaled graph (showing the values scaled between 0 and 4095)

The raw graph (showing the raw values between 0 and 4095)

4.4.4.1 Scaled Graph

The scaled graph shows the scaled version of the sensor value, normalized between 0 and 1. The default color for this graph is blue.

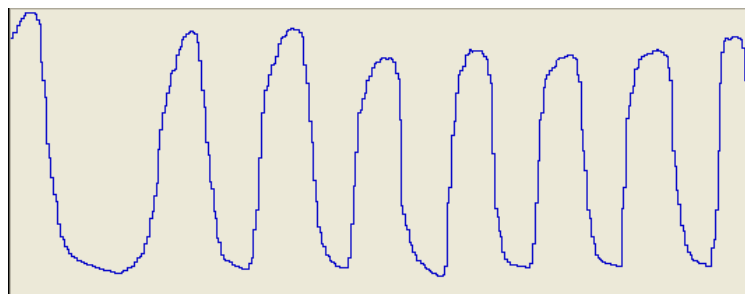


Figure 17 – The Scaled Graph

4.4.4.2 Raw Graph

The raw graph represents the raw sensor value coming from the glove. Additional to the actual graph of the raw values (drawn in red by default), the upper (drawn in green by default) and lower calibration (drawn in orange by default) values are also plotted, as shown in Figure 18.

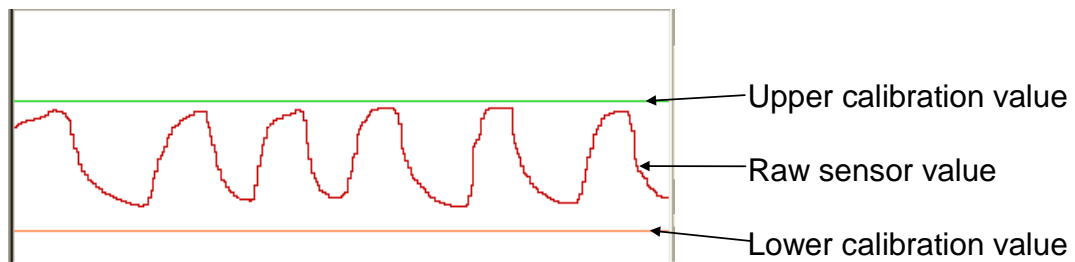


Figure 18 – The Raw Graph

4.4.4.3 Graph Zooming

You may zoom into a graph by simply clicking on it, as illustrated below:

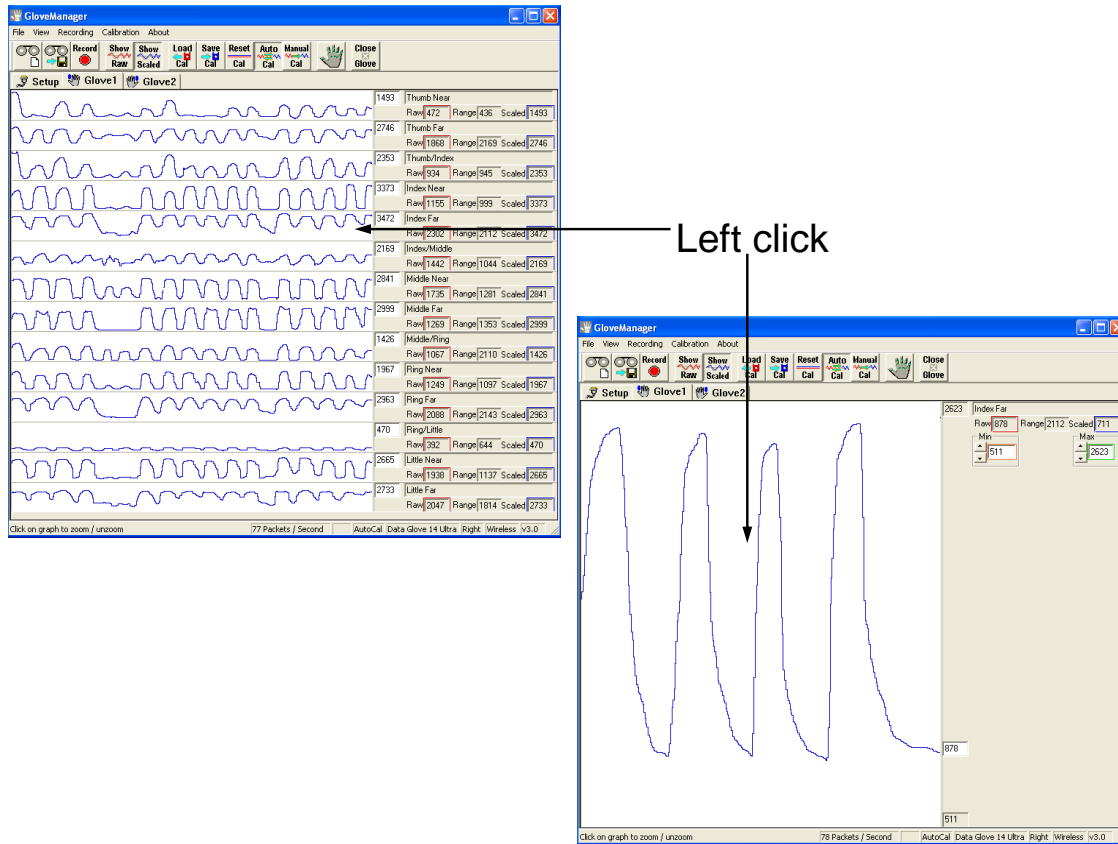


Figure 19 – An Illustration of the Zoom Procedure

The zoomed view gives you a more detailed view of the graph, and allows you to do fine-tuning of the calibration value.

4.4.5 The Sensor Info Box

The sensor info box provides the following information (see Figure 20):

- Raw sensor value [Value between 0 and 4095]
- Sensor dynamic range [Value between 0 and 4095]
- Scaled sensor value [*Scaled* value from 0 to 4095]
- Lower calibrated value [Value between 0 and 4095]
- Upper calibrated value [Value between 0 and 4095]
- Graph top value [Value between 0 and 4095]
- Current graph value [Value between 0 and 4095]
- Graph bottom value [Value between 0 and 4095]

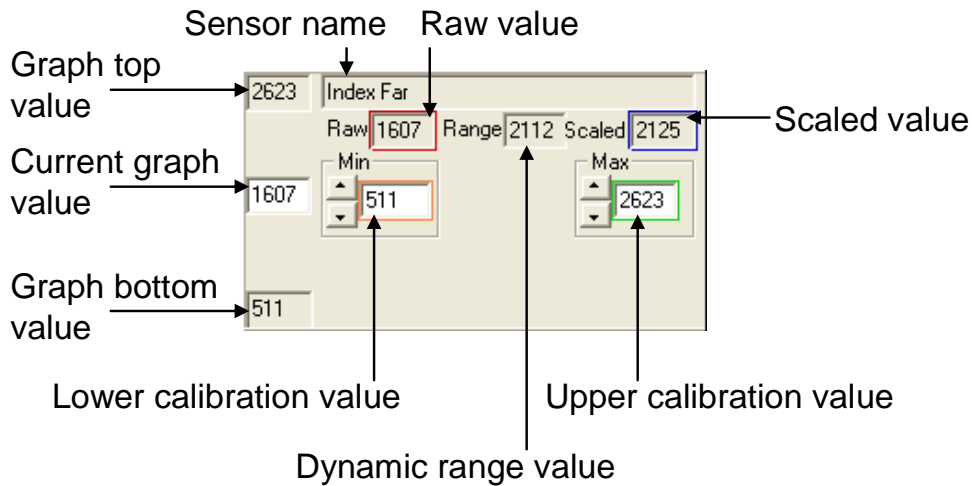


Figure 20 – The Sensor Info Box

The upper and lower calibration values may be fine-tuned by adjusting the minimum and maximum values of the dynamic range. These values are automatically adjusted during auto calibration. Please note that the calibration mode is automatically switched to manual calibration during fine-tuning.

The sensor info box has three levels of detail, depending on the pixel resolution available to GloveManager. The highest level of detail is shown in Figure 20. As the pixel resolution decreases it is switched to the second level of detail as in Figure 21. The value shown next to the sensor name is:

- the current scaled value (when scaled graphs are visible)
- the current raw value (when raw graphs are visible)

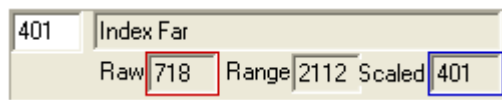


Figure 21 – The Second Level of Detail of the Sensor Info Box.

If the pixel resolution is decreased more, only the sensor name and current graph value are shown.



Figure 22 – The Lowest Level of Detail of the Sensor Info Box.

4.4.6 The Status Bar

The status bar displays various program and glove related information, including:

- Program status and information messages
- The packet rate
- An indication if recording is currently on
- An indication if auto calibration is currently on
- The model of the current glove
- The handedness of the current glove
- An indication if the current glove is wired or wireless
- The firmware version of the current glove

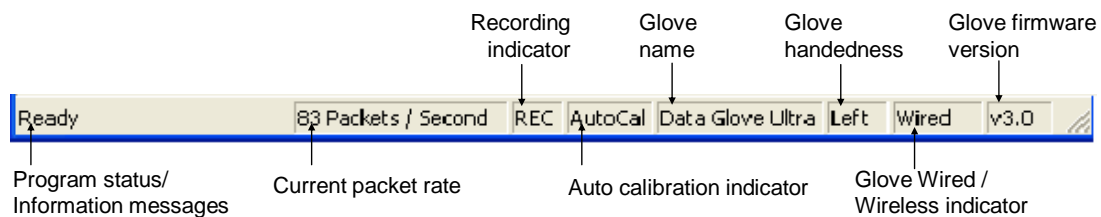



Figure 23 - The Status Bar


4.5. The Sensor Selection View

The sensor selection view may be obtained by clicking the  button on the toolbar, by selecting the **View | Show View Setup** option from the menu or by pressing the **V** shortcut key. The view is shown in Figure 24.

The hand and sensor image on the left allows you to select sensors to be used during recording. Selecting none means that this glove will not be used during the recording process.

The hand and sensor image on the right allows you to select specific sensors to be shown in graph view. Please note that the program will not allow you to display zero sensors.

Clicking the , , , , , or buttons causes the corresponding sensors to be selected or deselected. You may also select or deselect a sensor by clicking on it (●: Sensors to be used during recording or ●: Sensors to be shown in graph view).

You may go back to graph view by clicking on the or  buttons, by selecting the **View | Show View Setup** option from the menu or by pressing the **V** shortcut key.

4.6. Glove Calibration

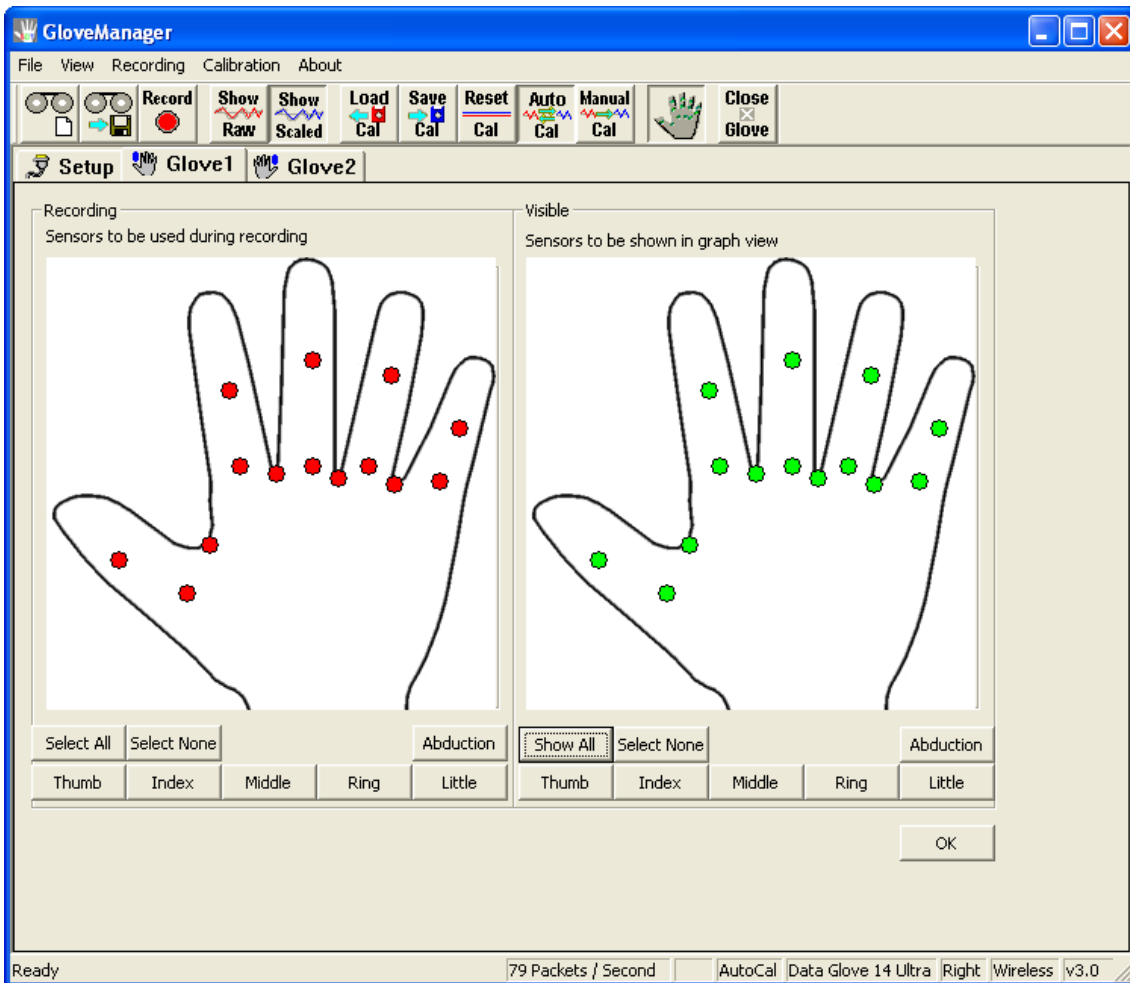



Figure 24 – The Sensor Selection View

Glove calibration values may be loaded from file by clicking the  button, by selecting the **Calibration | Load Calibration Values** menu option, or by using the **Alt+O** shortcut key. The file open dialog box will appear.

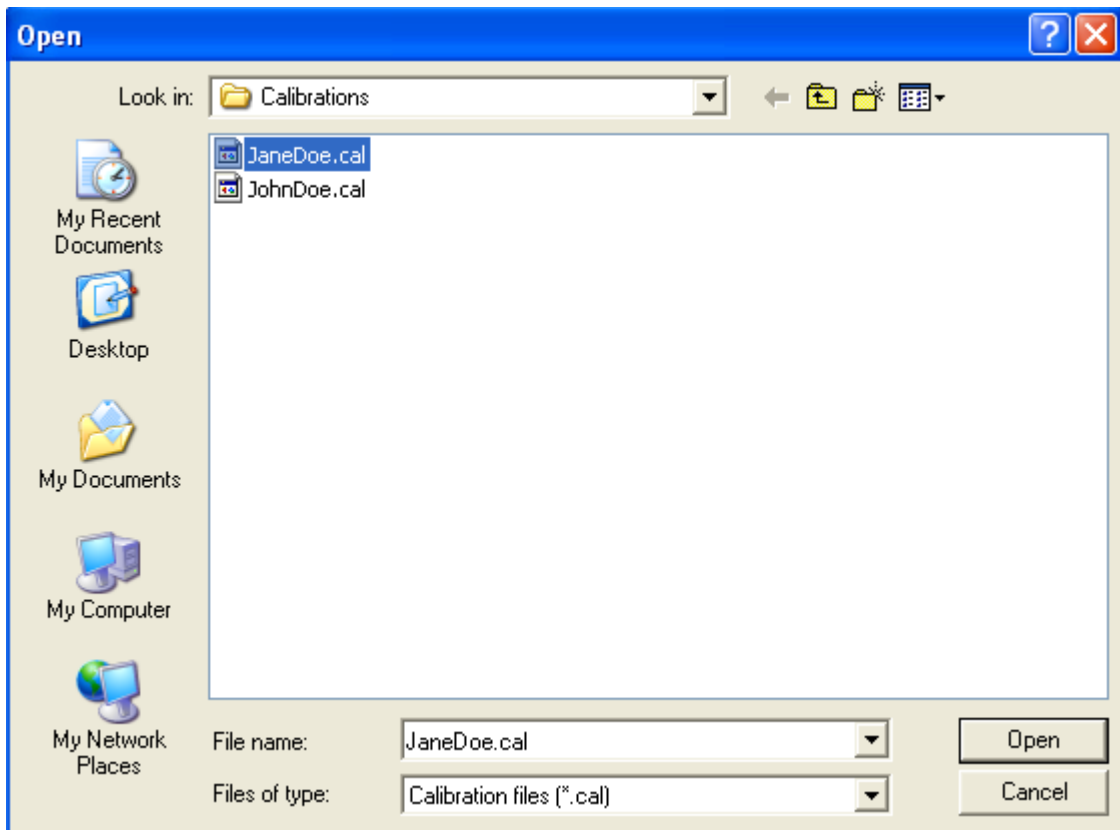




Figure 25 – The File Open Dialog for Calibration Files

Select a calibration file (extension .cal) and click on the  button to open the file.

Glove calibration values may be saved to file by clicking the  button, by selecting the **Calibration | Save Calibration Values** menu option, or by using the **Alt+S** shortcut key. The file save dialog box will appear.

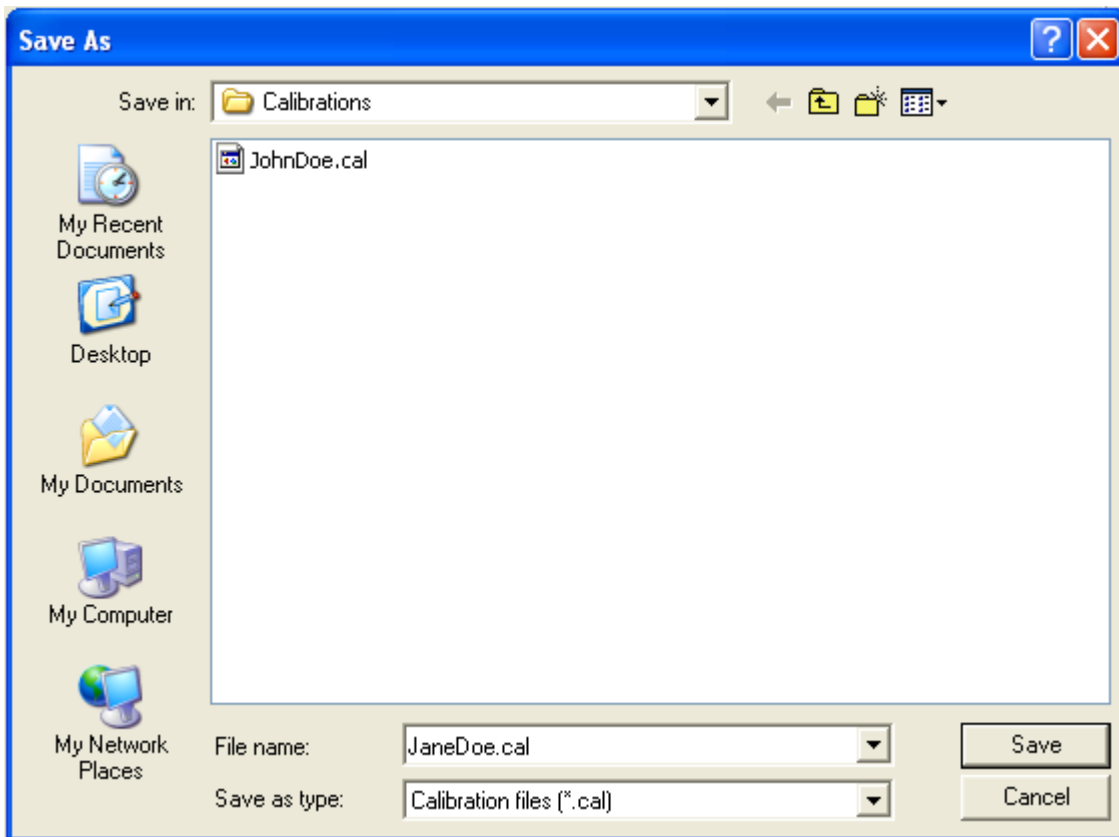




Figure 26 – The File Save Dialog for Calibration Files


Type in your file name and click the  button to save the calibration values to file. The file format is also compatible with the 5DT Data Glove SDK.

4.7. Recording



The incoming glove data may be recorded by clicking the  button, by selecting the **Recording | Record** menu option, or by using the **Ctrl+R** shortcut key. This will cause GloveManager to record the latest raw and scaled values at a fixed rate of 60 Hz. Only the sensors selected for recording in the sensor setup view (see section 4.5) will be recorded. These values are recorded in memory and will be lost unless saved to file.



The values may be saved to file, by clicking the  button, by selecting the **File | Save** or **File | Save As...** menu option, or by using the **Ctrl+S** shortcut key. Currently only the CSV (Comma Separated Values) file format is supported.

5. Using the Gloves in Autodesk MotionBuilder

Please note that the Autodesk MotionBuilder driver is an optional extra available with the glove. If a MotionBuilder license has not been purchased with the glove the plug-in will only work for 2 minutes and recording functionality will be disabled.

5.1. Installation

To install the driver, follow the steps given below.

- Copy the file **ordevice5dt16.dll** and **fglove.dll** to the C:\[Program Files]\Autodesk\ MotionBuilder\bin\win32\plugins\ directory.
- Run MotionBuilder.
- Verify that the driver is installed by clicking on the Asset Browser tab in the Asset Browser window. Then from Templates select Devices. The installed driver should be visible as shown in the red rectangle in Figure 27 below.

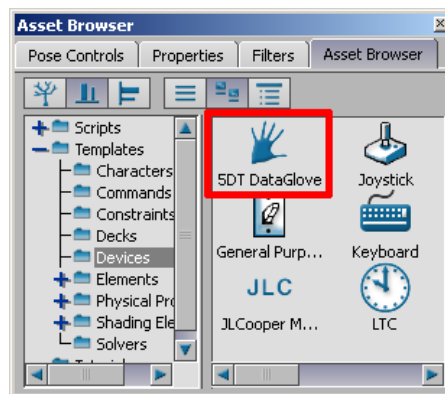


Figure 27 - Verifying that the 5DT Data Glove Plugin has been Installed

5.2. Opening the Glove

From the device pane of the Asset Browser, create a glove in MotionBuilder by clicking and dragging the 5DT DataGlove icon to the viewer window (see Figure 28).

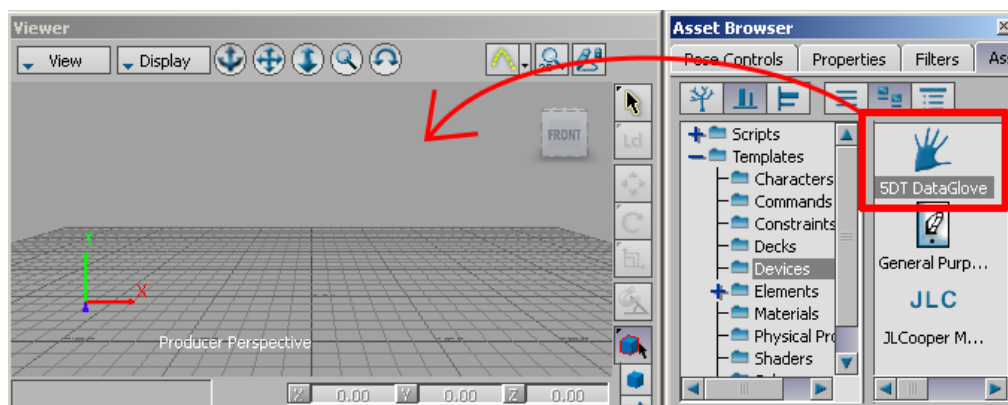


Figure 28 - Clicking and Dragging the 5DT DataGlove Icon to Create a New Glove

In the navigator pane, click on the newly created glove. The following window will appear:

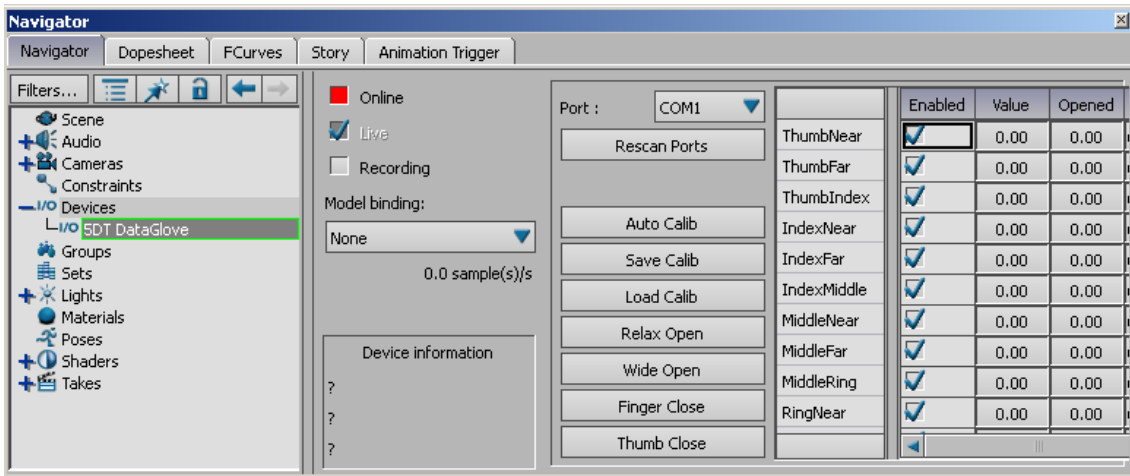


Figure 29 - The Glove Control Pane

If you know which port the glove is connected to, select it from the ports drop-down list.

If you are unsure as to which port your glove is connected to you may click on the **Rescan Ports** button. The driver will then scan the USB and all COM ports for available gloves and update the Ports drop-down list. Please note that the scan process may take a while.

Once the correct port has been selected, click on the red **Online** check box. If the red check box turns green, a successful connection has been made.

5.3. Opening Wireless Gloves

The new 5DT Data Glove Ultra Wireless Kit uses one COM port for receiving data from both gloves connected to the transmitter unit. The driver handles this transparently by allowing you to open two gloves on the same port. Only once the first glove is opened will the second one be opened by the driver. You may open and close either of these gloves any time you wish.

The process of opening two wireless gloves is illustrated next:

1. The first glove is created by dragging the 5DT DataGlove icon from the device pane into the viewer window and then performing a port scan. Now click on the Online button.
2. Click on the **Online** check box. It should turn green and the Device information should be updated with information from the current glove.
3. The second glove is created by dragging the 5DT DataGlove icon from the device pane into the viewer window. The driver automatically picks up that another wireless glove may be opened on COM1-B and adds this to the ports list.
4. The second glove is opened by selecting the correct port and clicking on the **Online** check box.

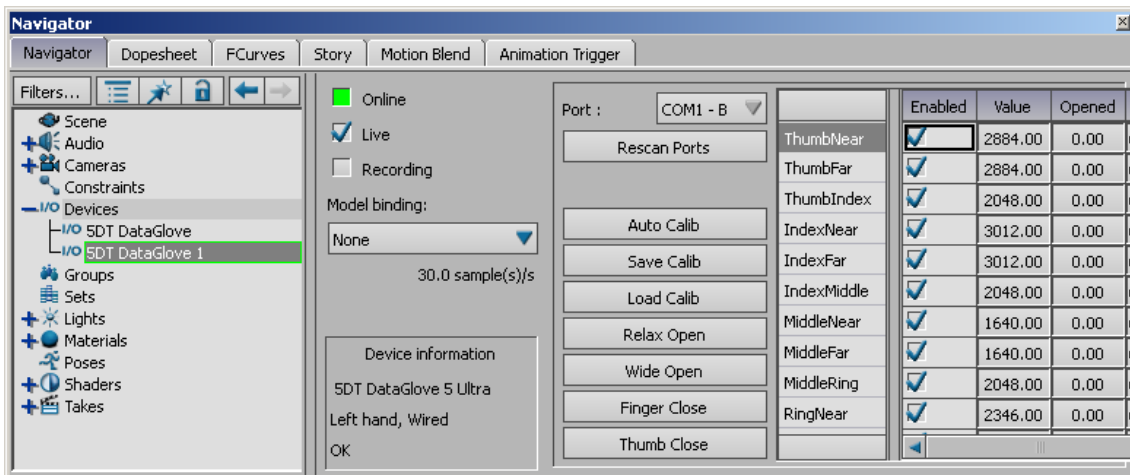


Figure 30 - Glove B has been Opened

5.4. Calibration

You may use auto calibration to calibrate the glove, but it is recommended that you use the pre-defined calibration steps.

To use auto calibration, click on the **Auto Calib** button. The button text will change to **Click to Stop**. Once you are satisfied with the calibration values, click the button again to stop calibrating.

The following calibration process is recommended.

1. Hold your hand in a relaxed, open position, and click on the **Relax Open** button.
2. Open all of your fingers as wide as possible and click on the **Wide Open** button.
3. Close all of your fingers (except your thumb) and click on the **Finger Close** button.
4. Close your thumb and click on the **Thumb Close** button.
5. Calibration is now complete.

5.5. Model Binding

To create a model binding, select **Create...** from the Model binding drop-down menu as in Figure 31.

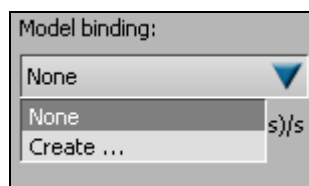


Figure 31 - The Model Binding Drop-down List

After the Create command has been issued the wire frame hand model is created in the viewer and the model's name is added to the Model binding drop-down list.

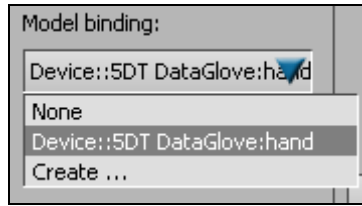


Figure 32 - The Model Binding Drop-down List with Hand

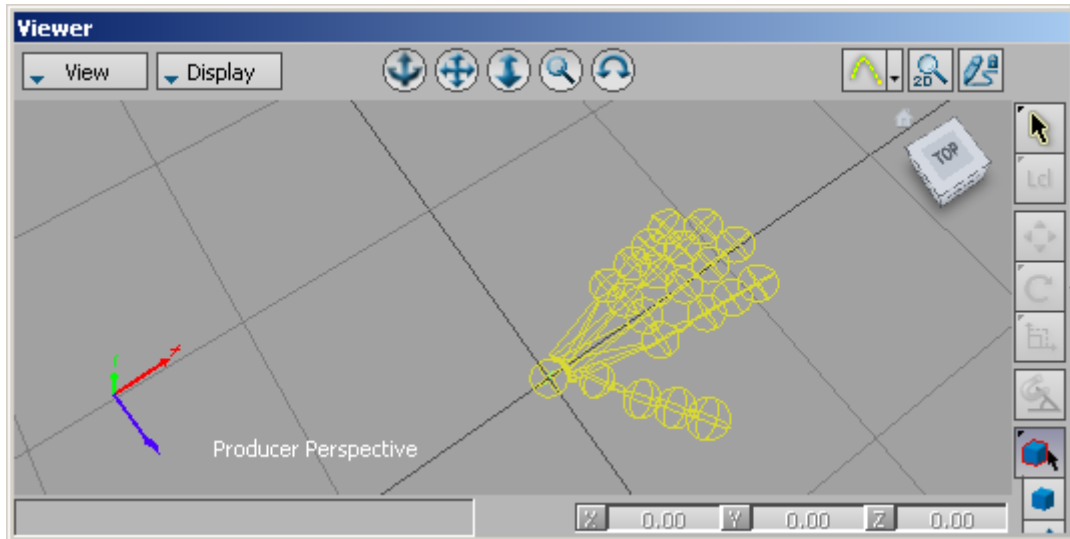


Figure 33 – The Glove Model as Shown in the Viewer Window

This glove can now be automatically connected to an actor. To create an actor, click and drag the Actor icon from the Characters pane in the Asset Browser.

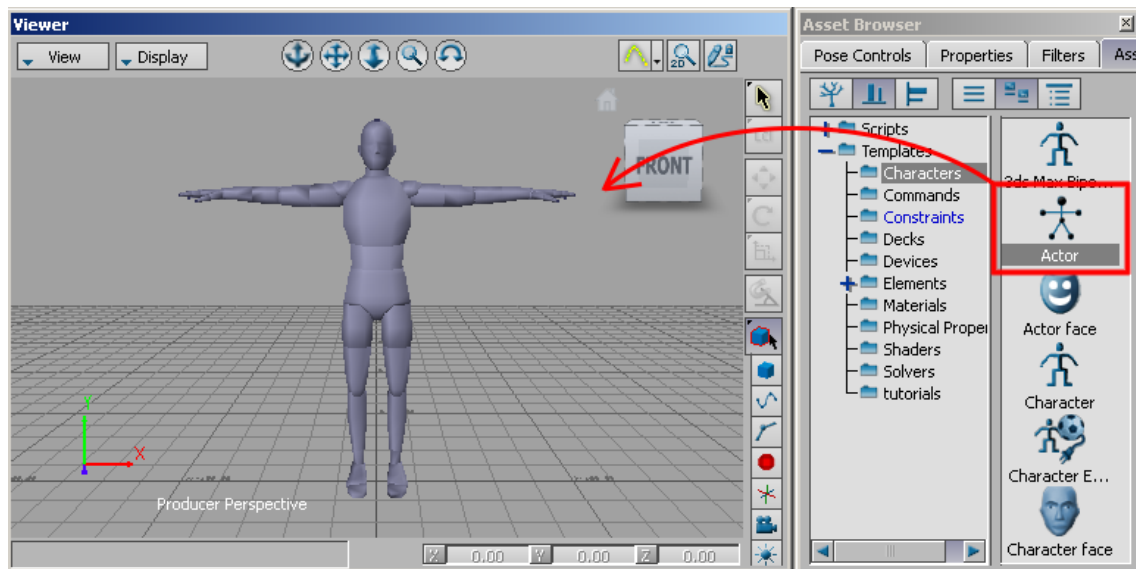


Figure 34 – Creating an Actor

If the actor asset is selected in the Navigator window, the Actor Settings window will appear.

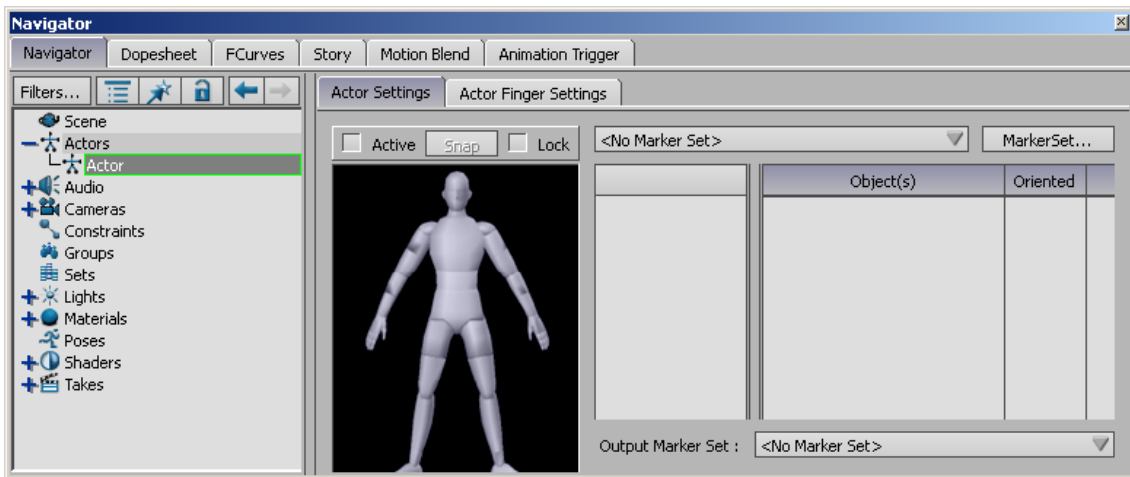


Figure 35 – The Actor Settings Window

If no Marker Set exists for this actor create one by clicking on the **MarkerSet...** button. A popup menu will appear. Select the **Create** option.

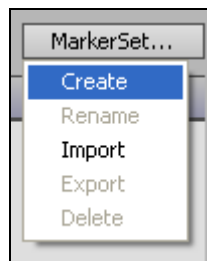


Figure 36 – Creating a Marker Set

Once a Marker Set has been selected, select the hand marker to which the glove should be referenced. Drop the root of the glove device (under the Scene branch) into the Glove Reference in the Model column. Click on the Active checkbox to make the actor active (real-time animations are shown).

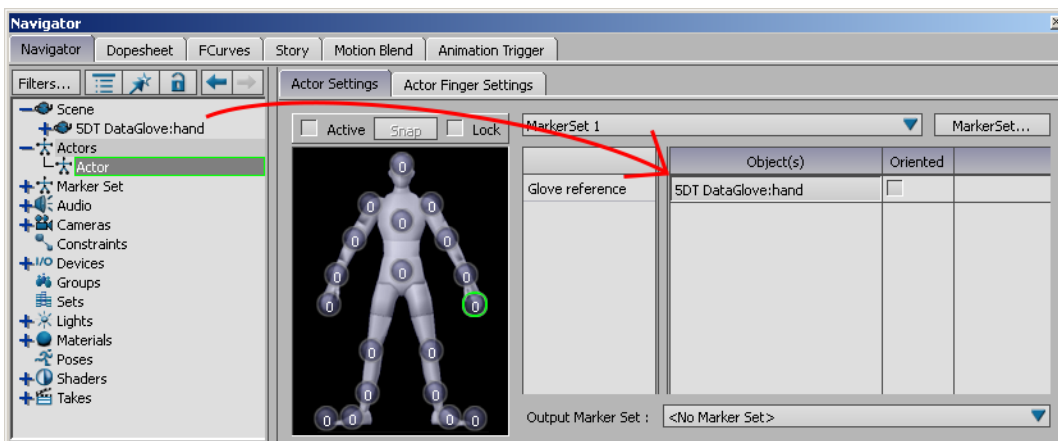


Figure 37 – Assigning a Glove Reference to a Marker Set

6. Using the SDK

6.1. Introduction

The 5DT Data Glove SDK provides access to the 5DT range of data gloves at an intermediate level.

Various platforms and programming languages are supported.

- Microsoft Visual Studio C/C++
- Microsoft Visual Studio C#
- Linux C/C++
- Mac OS X C/C++

The SDK functionality includes:

- Multiple instances
- Easy initialization and shutdown
- Basic (raw) sensor values
- Scaled (auto-calibrated) sensor values
- Calibration functions
- Basic gesture recognition
- Cross-Platform Application Programming Interface

The SDK functions are described in detail in Appendix C and D.

6.2. Using the SDK

6.2.1 Microsoft Visual Studio C/C++

6.2.1.1 Installation

The 5DT Data Glove Driver is easy to implement using the following guidelines:

1. Make sure that the header file `fglove.h`, the library file `fglove.lib` and the dynamic link library file `fglove.dll` reside in the current (application) directory, or somewhere that they can be found. The file `fglove.dll` may be copied into your Windows system directory.
2. Include the header file `fglove.h` in the application where necessary.
3. Add the library file `fglove.lib` to the link process.

There is also a debug version of the driver (`fgloved.lib`, `fgloved.dll`) which outputs debug messages to the debugger.

6.2.1.2 Example applications

In the examples directory, there is example source code that makes use of the glove SDK. The example projects are saved as Microsoft® Visual Studio® 2003 projects.

6.2.2 Microsoft Visual Studio C#

The following guidelines help to setup the 5DT glove driver for use in a C# application.

1. Right Click on the References and Click on “Add Reference...”

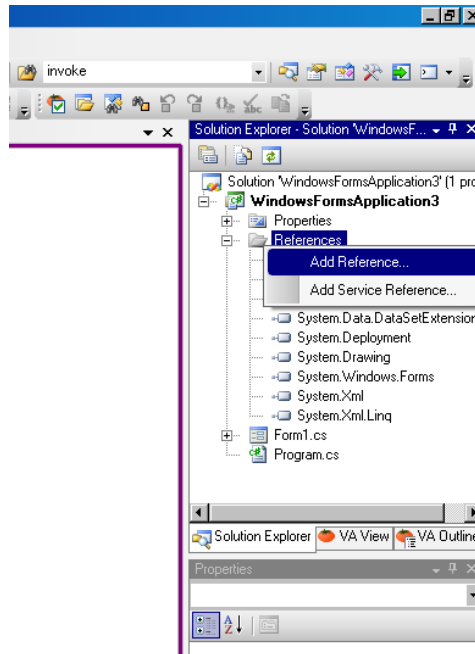


Figure 38 – Adding a Reference

2. Browse to the directory which contains 5DTGloveDriverCLI.dll. Select 5DTGloveDriverCLI.dll and click on OK.

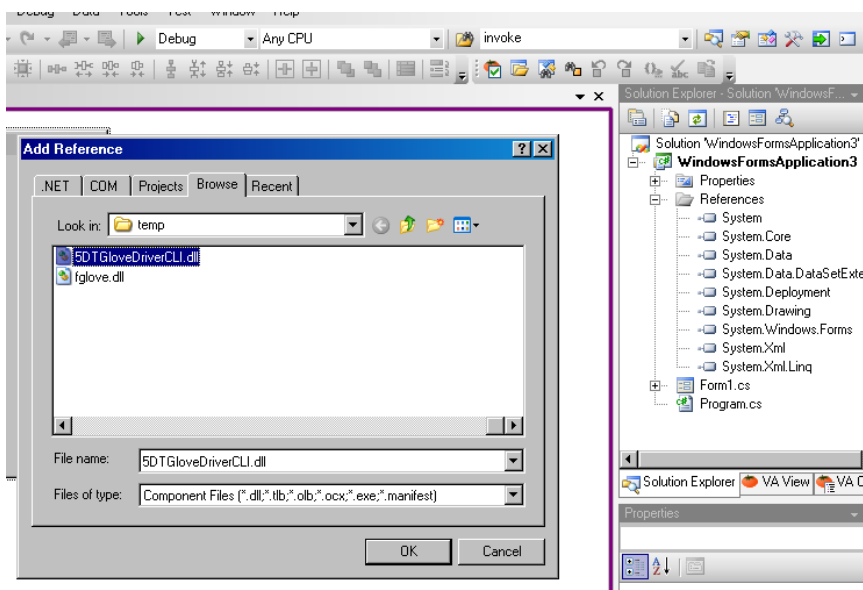


Figure 39 – Selecting the C# glove driver DLL

3. Confirm that the 5DTGloveDriverCLI is displayed under the References.

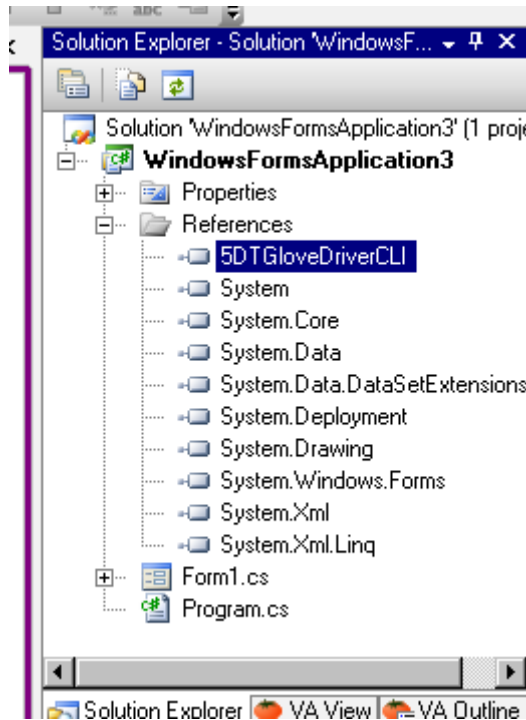


Figure 40 – C# glove driver added to References

4. Add the following using statement at the top of the *.cs file.

```
using FDTGlove;
```

5. You may now start using the glove

```
CfdGlove fdGlove; //Glove class  
float[] aSensorCaledVals = null; //Array of scaled sensor values  
...  
fdGlove = new CfdGlove(); //create a new glove  
fdGlove.GetSensorScaledAll(ref aSensorCaledVals); //read values  
fdGlove.Close(); //close the glove
```

See appendix D for more function names. You may also double-click on the “5DTGloveDriverCLI” displayed under the References (See Figure 40) to see all the types and functions that are available.

6.2.3 Linux C/C++

6.2.3.1 Getting started

Run install.sh in the “install” folder. This script will copy the shared objects and header files into the correct directories. If you do not have administrator privileges please see 6.2.3.2.

In the terminal type:

```
$ ./install.sh
```

Or double click on the script if the file is configured to be executed as a program.

Type "chmod +x install.sh" to allow executing file as a program, or right click on the file and find the "allow executing file as program" checkbox in the file properties.

Now that you have successfully installed the glove SDK you will be able to execute polling.o in "polling" directory. In the terminal navigate to the "polling" directory. If your glove has a USB connection type:

```
$ sudo ./polling /dev/usb/hiddev0
```

If your glove has a serial connection type:

```
$ sudo ./polling /dev/ttyS0 (replace ttyS0 with the correct serial port)
```

The "sudo" command is used here to make sure the application has access to the usb or serial device. The program will normally finish in less than 10 seconds. If you do not have root access please see the next section.

To find the serial ports that are available, type the following into the terminal:

```
$ setserial -g /dev/ttyS*
```

6.2.3.2 *Setting up the Linux environment for non-administrator users*

6.2.3.2.1 *Installing the library*

If you do not have root access to the system, you should place the library (libfglove.so) somewhere in your home directory, and then set the LD_LIBRARY_PATH environment variable to include the full path of the directory in which you have placed the library. This will indicate to the dynamic library loader where to find the file. For example:

```
$ mkdir /home/yourhomedir/libs  
$ cp libfglove.so /home/yourhomedir/libs  
$ export LD_LIBRARY_PATH="/home/yourhomedir/libs"
```

In the above example, replace "yourhomedir" with your own home directory name.

6.2.3.2.2 Setting up access to the serial port

The 5DT Data Glove accesses the serial port using the standard Unix/Linux device files in the /dev directory. Applications that use the glove should typically give the user the option of specifying which device to check, such as /dev/ttyS1. It is recommended that you create a symbolic link /dev/fglove to your device file, e.g:

```
# cd /dev
# ln -s ttyS1 fglove
```

This may make the setup of applications easier should you need to change the port that the glove is connected to.

It is also required that the user of the glove has read/write access to the serial port device file. By default, normally only the root user has these rights. If non-root users on the system will be using the glove, the root user must grant access rights to the device file.

For example, the following command will give everyone on the system full access to the serial port /dev/ttyS1:

```
# chmod 777 /dev/ttyS1
```

6.2.3.3 Example applications

6.2.3.3.1 Glove info example

This example is located in the “glove_info” directory.

In this example the glove driver is queried for info such as glove type and packet rate. Function calls `fdGetGloveHand()`, `fdGetGloveType()` and `fdGetPacketRate()` are used to get info from the glove driver.

6.2.3.3.2 Polling example

This example is located in “polling” directory.

In `polling.cpp` 5000 samples are read from the driver with a 1 millisecond delay between each sample. In the previous example the packet rate was printed out and will typically be around 75Hz. Take note that in this example more samples are taken per second (1000 samples/second) from the driver compared to what is updated from the glove hardware (75 samples/second).

6.2.3.3.3 Callback example

This example is located in “callback” directory.

Instead of polling the glove driver for new values like in the previous example, a function will be called by the driver every time a sensor value changes.

The following function is defined at the top of `callback.cpp`:

```
void call_back(void* param)
{
    iNumCallbackCalls++;

    float gloveA_scaled[18];
    fdGetSensorScaledAll(static_cast<fdGlove*>(param), gloveA_scaled);
    ...
    printf(" >> %d\n", fdGetGesture(static_cast<fdGlove*>(param)));
}
```

The `call_back()` function receives a `void*` parameter, which is casted to a `fdGlove*`.

The `call_back()` function needs to be registered as a callback in the glove driver:

```
fdSetCallback(pGloveA, (void*)&(call_back), static_cast<void*>(pGloveA))
```

The third parameter passed to `fdSetCallback()` is of type `void*`, and this is the parameter that should be passed to the callback function when it is called by the driver. In this case it is a pointer to the glove driver.

6.2.4 Mac OS X C/C++

The following guidelines help to setup the 5DT glove driver for use in Xcode running on Mac OS X.

1. Startup Xcode and create a new Xcode Project. Select “Application” and then “Command Line Tool” and click on “Choose...”.

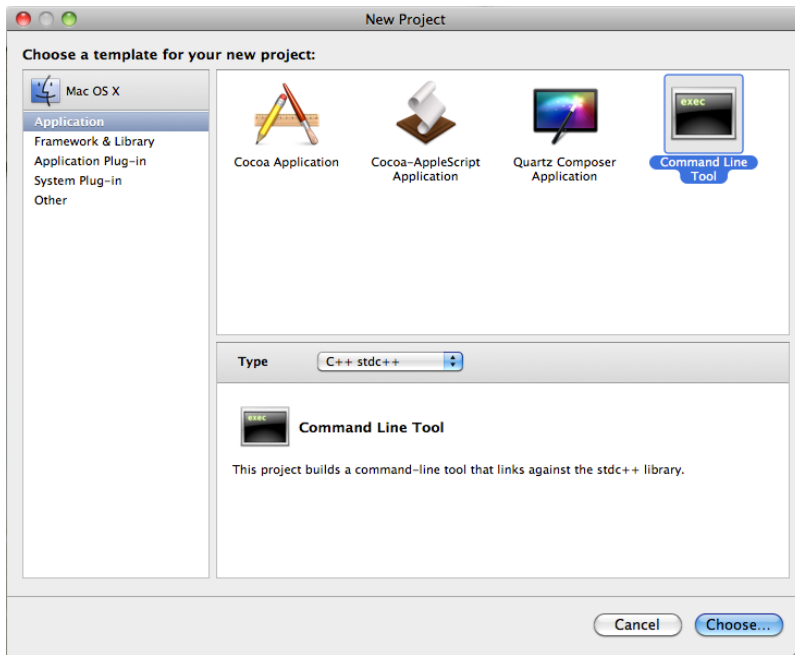


Figure 41 - Create new command line project

2. Paste fglove.dylib into your “build\Debug\” and “build\Release” directory of your project. If no such directory exists you have to build the application first: In the menu click on Build → Build or press CMD+B.
3. Paste fglove.h into the directory of your project.
4. Add the fglove.dylib file to your project by right-clicking on your project name in the “Groups & Files” tab and selecting “Add→Existing Frameworks...”.

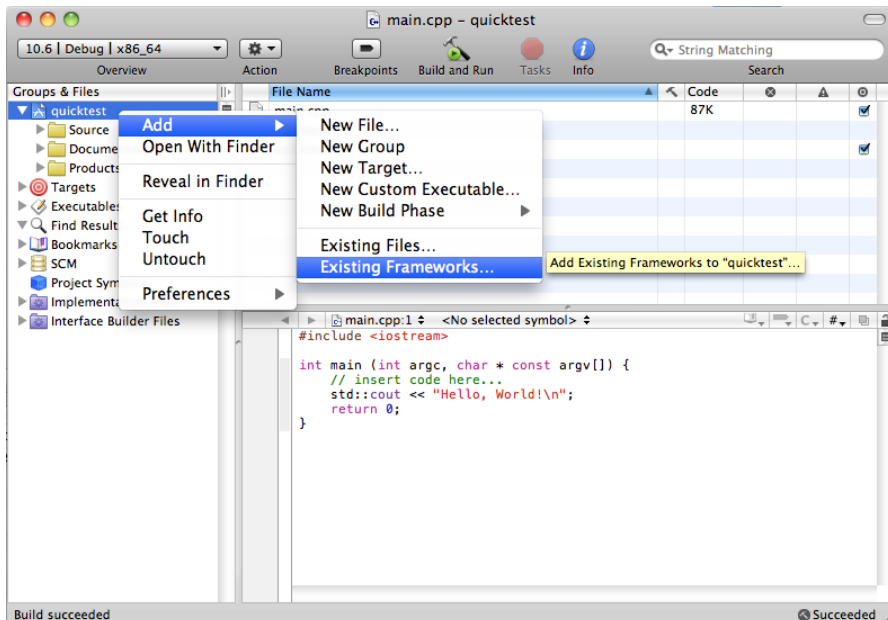


Figure 42 - Add fglove.dylib to your project

A new window will pop up. Click on “Add Other...” button at the bottom of this window. Browse to fglove.dylib, select it, and click on “Add”.

5. Add the following code to the “main.cpp” file of your project.

```

#include <iostream>
#include "fglove.h"

int main (int argc, char * const argv[])
{
    fdGlove *pGlove = NULL;
    pGlove = fdOpen(""); // connects to first glove found

    if (pGlove)
    {
        cout<<"found glove \n";
        cout<<"Glove Type:"<<fdGetGloveType(pGlove);
    }

    fdClose (pGlove);

    return 0;
}

```

6. Compile and run your application (CMD + Enter). Open the Debugger console to verify that your program gave the correct output (CMD + Shift + R).
7. When you connect to the glove using `fdOpen("")` the driver connects to the first glove it finds. If you want to connect to a specific glove you should supply the name to `fdOpen()`. For example `fdOpen("DG5U_L")` will connect to Glove 5 Ultra left-hand. Other valid name inputs to `fdOpen()` are "DG5U_R", "DG14U_L" and "DG14U_R".

6.3. Connecting to the second glove when using a wireless kit

In the situation where the wireless kit is used and there are 2 gloves connected simply call `fdOpen()` twice with the same port name as parameter to get hold of the second glove. Remember to store the returned glove pointer of the second call in a different pointer variable so that you can access both gloves individually. You may now use the second glove pointer in any way you would use any other glove pointer.

6.4. Supported Gloves

The glove SDK supports all 5DT Data Gloves. The current version implements 18 possible sensors, and includes the roll and pitch sensors of the 5DT Data Glove 5. The driver attempts to map values to all sensor outputs. If it is unable to do so the sensor value defaults to a sensible value. This value can be adjusted by forcing a specific value. To the application programmer the driver therefore appears the same regardless of the type of glove that is connected.

6.5. Sensor Mappings for the 5DT Data Glove 5 and 5 Ultra

The sensors on the 5DT Data Glove 5 are positioned as in Figure 43.

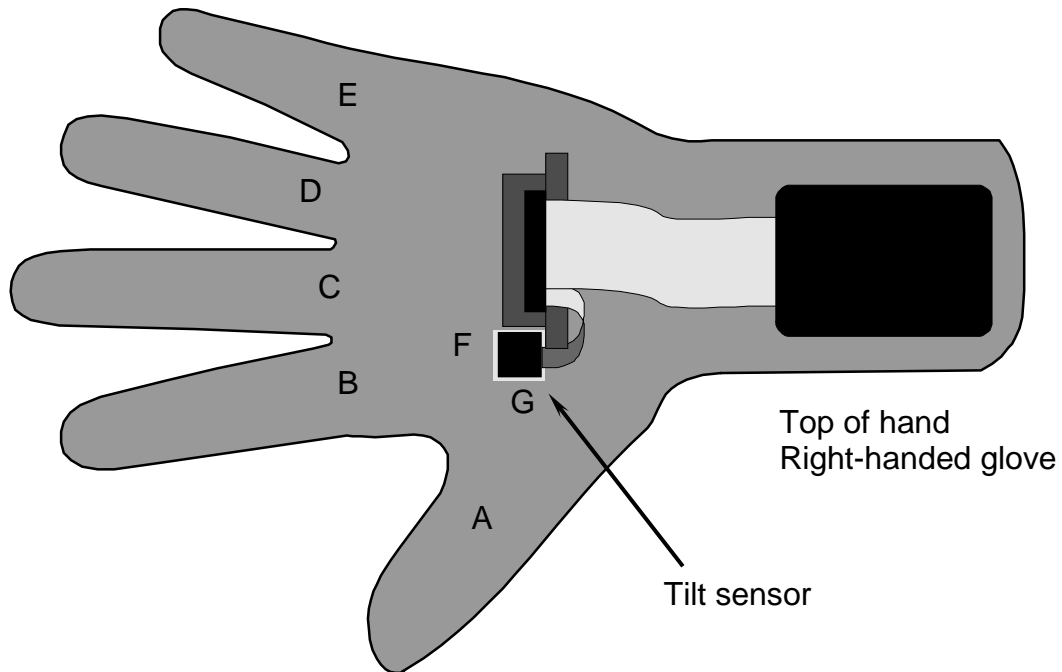


Figure 43 - Sensor Positions for the 5DT Data Glove 5

| <i>Sensor</i> | <i>Driver Sensor Index</i> | <i>Description</i> |
|---------------|----------------------------|---------------------------------------|
| A | 0,1* | Thumb flexure |
| B | 3,4* | Index finger flexure |
| C | 6,7* | Middle finger flexure |
| D | 9,10* | Ring finger flexure |
| E | 12,13* | Little finger flexure |
| F | 16 ⁺ | Pitch angle of tilt sensor (obsolete) |
| G | 17 ⁺ | Roll angle of tilt sensor (obsolete) |

Table 3 - Sensor Mappings for the 5DT Data Glove 5 and 5DT Data Glove 5 Ultra

* Both these driver sensor indices will return the same value when the 5DT Data Glove 5 or Data Glove 5 Ultra is used.

+ Not available on the Data Glove 5 Ultra.

6.6. Sensor Mappings for the 5DT Data Glove 14 Ultra

The sensors on the 5DT Data Glove 14 are positioned as in Figure 44.

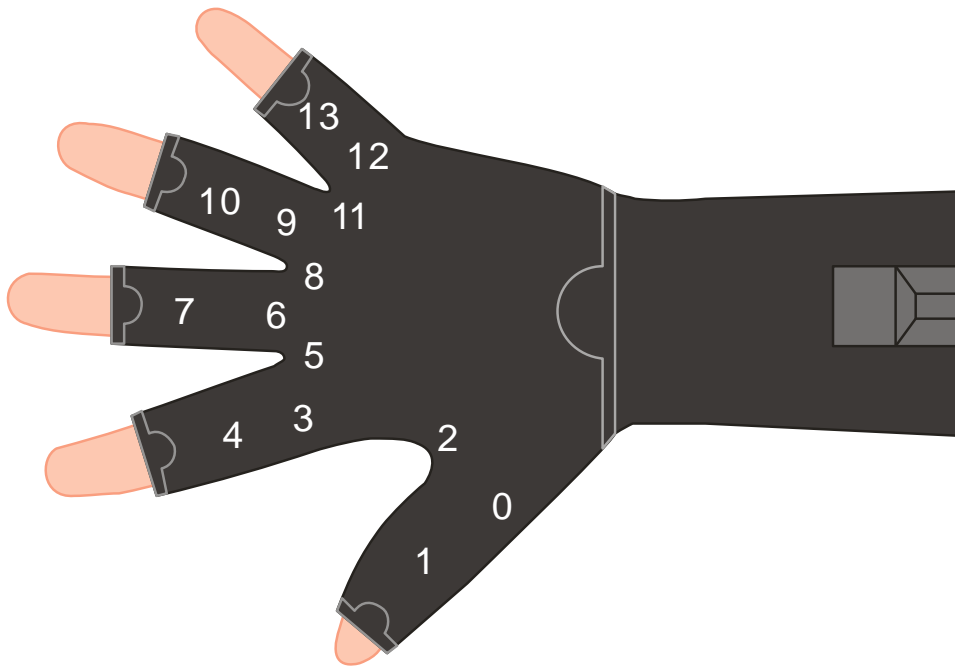


Figure 44 – Sensor Mappings for the 5DT Data Glove 14 Ultra

| <i>Sensor</i> | <i>Driver Sensor Index</i> | <i>Description</i> |
|---------------|----------------------------|--|
| 0 | 0 | Thumb flexure (lower joint) |
| 1 | 1 | Thumb flexure (second joint) |
| 2 | 2 | Thumb-index finger abduction |
| 3 | 3 | Index finger flexure (at knuckle) |
| 4 | 4 | Index finger flexure (second joint) |
| 5 | 5 | Index-middle finger abduction |
| 6 | 6 | Middle finger flexure (at knuckle) |
| 7 | 7 | Middle finger flexure (second joint) |
| 8 | 8 | Middle-ring finger abduction |
| 9 | 9 | Ring finger flexure (at knuckle) |
| 10 | 10 | Ring finger flexure (second joint) |
| 11 | 11 | Ring-little finger abduction |
| 12 | 12 | Little finger flexure (at knuckle) |
| 13 | 13 | Little finger flexure (second joint) |

Table 4 – Sensor Mappings for the 5DT Data Glove 14 Ultra

6.7. Gesture Definitions

The currently defined set of gestures is comprised of binary open/close configurations of the fingers excluding the thumb. There are $2^4 = 16$ such possible combinations. Gesture number 0 is defined as all the fingers (excluding the thumb) being closed, and gesture number 15 as all the fingers open. The index finger indicates the least significant bit. For example, the index finger point gesture will therefore be number 1, and the little finger point gesture number will be 8. An invalid (unrecognizable) gesture is defined as the value -1.

A scaled sensor value of higher than the upper threshold setting will indicate a closed finger, while a scaled sensor value of lower than the lower threshold setting will indicate an open finger. A value in-between is invalid and will result in an invalid gesture. In the case of multiple finger joint angle measurements, the maximum of the individual joint sensor values is taken to obtain a closed gesture and the minimum to obtain an open gesture. Closed gestures take precedence, in other words, bending only one joint of a finger will count as a closed gesture.

| Finger: | Little | Ring | Middle | Index |
|--------------------------------------|--------|-------|--------|-------|
| 5DT Data Glove 5 sensor: | E | D | C | B |
| 5DT Data Glove 16 & 14 Ultra sensor: | 12,13 | 9,10 | 6,7 | 3,4 |
| Driver sensor index: | 12,13# | 9,10# | 6,7# | 3,4# |

| Gesture Number | Flexure (0=flexed, 1=unflexed) | | | | Gesture Description | Fig. |
|----------------|--------------------------------|---|---|---|-------------------------------|-------|
| 0 | 0 | 0 | 0 | 0 | Fist | 16.0 |
| 1 | 0 | 0 | 0 | 1 | Index finger point | 16.1 |
| 2 | 0 | 0 | 1 | 0 | Middle finger point | 16.2 |
| 3 | 0 | 0 | 1 | 1 | Two finger point | 16.3 |
| 4 | 0 | 1 | 0 | 0 | Ring finger point | 16.4 |
| 5 | 0 | 1 | 0 | 1 | Ring index point | 16.5 |
| 6 | 0 | 1 | 1 | 0 | Ring middle point | 16.6 |
| 7 | 0 | 1 | 1 | 1 | Three finger point | 16.7 |
| 8 | 1 | 0 | 0 | 0 | Little finger point | 16.8 |
| 9 | 1 | 0 | 0 | 1 | Index and little finger point | 16.9 |
| 10 | 1 | 0 | 1 | 0 | Little middle point | 16.10 |
| 11 | 1 | 0 | 1 | 1 | Not ring finger point | 16.11 |
| 12 | 1 | 1 | 0 | 0 | Little ring point | 16.12 |
| 13 | 1 | 1 | 0 | 1 | Not middle finger point | 16.13 |
| 14 | 1 | 1 | 1 | 0 | Not index finger point | 16.14 |
| 15 | 1 | 1 | 1 | 1 | Flat hand | 16.15 |

Table 5 - Gesture Definition Scheme as Implemented for the 5DT Data Glove SDK

When the 5DT Data Glove 5 is used, both these driver sensor indices will return the same value.



Figure 45 - Gesture Illustrations

When the 5DT Data Glove 16 or 5DT Data Glove 14 Ultra is used, the driver sensor indices will return different values. The maximum of the two values will be used to test for a flexed (closed) gesture, and the minimum value of the two will be used to test for an unflexed (open) gesture.

6.8. Auto-Calibration

The driver can provide sensor outputs in an automatic, linearly calibrated fashion. During every update, the raw value read from the sensor is compared to the current minimum and maximum raw values (raw_{min} and raw_{max}) as set by the functions `fdSetCalibrationAll()`, `fdSetCalibration()` or `fdResetCalibration()`. If the current minimum and maximum values are exceeded, they are overwritten. The upper and lower calibration values are therefore continuously pushed "outwards". The normalized output is given by the first order equation

$$out = \frac{raw_{val} - raw_{min}}{raw_{max} - raw_{min}} \cdot Max, \quad (F-1)$$

which is in $[0...Max]$. The value of Max is set by the functions `fdSetSensorMaxAll()` and `fdSetSensorMax()`. Doing a few flexing movements with the hand quickly sets the operating values for raw_{min} and raw_{max} , and calibrates the glove.

The auto-calibration process can be ignored by simply regarding only the raw sensor outputs. It would be up to the application developer to provide a suitable calibration process.

Note that calibration is mandatory, especially with the high-end gloves which contain no hardware calibration possibilities.

7. Troubleshooting and Support

The following information is provided to help you diagnose any problems that may be encountered with your 5DT Data Glove Ultra or 5DT Data Glove Ultra Wireless Kit. Please consult this section before contacting your supplier.

7.1. General Troubleshooting

If the glove is not working at all, please work through the **physical connection problems** section of the troubleshooting procedure to identify the source of the problem.

7.1.1 Physical Connection Problems

1. Check that power is connected to the receiver as well as to the wireless data transmitter. There should be a green light on the transmitter when power is applied.
2. Check that the glove is connected to the right port. Try to connect the glove to another port and try again.
3. Check all the connections. Section 2.1.2 has information on how to correctly connect the system.
4. Check that the port is working. Try connecting another device, for example an external modem.

If the 5DT Data Glove is recognized by your computer but not connecting properly, proceed directly to the **software connection problems** section.

7.1.2 Software Connection Problems

Run **GloveManager** from the start menu. Check which COM ports are available by looking at the list in Figure 7. Remember that not all available ports will be physically connected to a socket on the back of your computer. Some ports may be unavailable because other programs have already taken control of the ports. Examples are mouse drivers and fax or communication programs.

7.2. Frequently Asked Questions (FAQ)

Q My gloves are connected to COM2, but Glove Manager reports that this port is not available.

A There may be a mouse driver, or other program that has already opened that port. Once a program or driver has opened a COM port, no other program will be able to access that port until the first one closes.

Q Is it possible to use the 5DT Data Glove Ultra in a LINUX environment?

A Yes, the Linux version of the SDK does support the 5DT Data Glove Ultra. Please check our website regularly for driver updates.

Q Can I use four (or more) USB gloves on one machine at the same time?

A 5DT just loves people like you! The number of gloves is limited by the number of USB ports on your computer. USB hubs and PCI cards are available that increase the number of USB ports on a computer.

Q When I flex a finger, the output does not change but stays at the maximum value. What is the problem?

A In this case it is possible that one of the fibers has failed. The glove will then have to be returned to the supplier (or 5DT) for repairs.

7.3. Support

If you experience problems or have complaints, suggestions or other comments, please feel free to contact your supplier (preferably via e-mail). Please have the following information available, or include it in your written correspondence:

1. Computer processor make and model (e.g. Pentium IV 3.2GHz)
2. Operating system and version
3. The symptoms of the problem and what was happening when the problem occurred
4. Can you reproduce the problem easily? Describe how
5. Actions taken by yourself to solve the problem

Please feel free to contact 5DT directly. Our contact details are available in Section 9.

Your feedback helps make this a truly great product.

8. Warranty Information

8.1. Thirty (30) Day Customer Satisfaction Guarantee

If, for any reason, you are disappointed with this product, please contact your supplier. If you are not completely satisfied, you will be granted a full refund once the product is returned undamaged. Unfortunately, shipping costs cannot be refunded.

8.2. 6 Months Product Warranty

5DT warrants to the original purchaser of the 5DT Data Glove 5/14 Ultra that it will be free of defects in materials and workmanship for a period of six months from the original date of purchase. During the warranty period, 5DT will repair or replace (with a reconditioned unit) components that are defective.

Warranty extension options are available when purchasing your glove.

Please note that the warranty is a carry-in warranty. i.e. you would be responsible for the costs of the getting the glove to us, but 5DT will cover the costs of returning the glove to you.

8.3. Exclusions

The above warranty is provided for private use only and shall not apply to any commercial use of the product, including (but not limited to):

- Location based entertainment (LBE) applications
- Trade show demonstrations

This warranty shall not apply to defects resulting from the following:

- Misuse
- Improper or inadequate maintenance
- Unauthorized modification

8.4. Warranty Claim Instructions

Please use the following procedure if you require warranty service:

1. Contact your supplier to determine if you need to return the product. If your supplier is unable to determine this, contact 5DT directly. Do not return a product to your supplier or to 5DT **without first contacting them**.
2. Your supplier (or 5DT) will issue you with a Return Material Authorization (RMA) number. Do not return a product to your supplier or to 5DT without an RMA number.
3. Pack the items to be returned securely using the original packaging material (if possible).
4. Please enclose the following:
 - Your original sales bill
 - Name, address, contact telephone/fax numbers
 - E-mail address
 - Reason for returning the item

- RMA number
- 5. Mark your shipping container with the RMA number to expedite handling at your supplier or at 5DT.
- 6. Ship prepaid to your supplier or to 5DT.
- 7. When completing customs or courier documentation, please clearly indicate: "Unit returned for repairs under warranty".

5DT is not responsible for any damage that may occur during shipping. Shipping charges to 5DT are your responsibility. COD shipments will not be accepted by 5DT.

9. About 5DT

5DT (Fifth Dimension Technologies) is a high technology company specializing in Training Solutions for the Mining Industry. For more information about our company, please visit our web site at:
www.5DT.com

5DT offers:

- Training Simulators (TS)
- Computer Based Training (CBT) Systems [e-Learning Solutions]
- Visualizers (V)
- Educational Games (EG)
- Maintenance Simulators (MS)
- Virtual Reality and Simulator Hardware (e.g. Data Gloves, Head Mounted Displays [HMDs], Consoles and Motion Bases)

5DT has their own Software Development Kits (SDKs) for Simulator Development and e-Learning Solution Development.

Contact Information:

California, USA (Serves United States and Americas)
15375 Barranca Pkwy, G-103
Irvine, CA 92618, United States of America
Tel: +1 949 450 9044
Fax: +1 949 450 9045
Web: www.5DT.com

Pretoria, South Africa (Serves Rest of World)
25 De Havilland Crescent, P.O. Box 5
Persequor Technopark, 0020, South Africa
Tel: +27 12 349 2690
Fax: +27 12 349 1404
Web: www.5DT.com

Appendix A – Hardware Specifications

- MATERIAL:** Black stretch lycra
- FLEXURE RESOLUTION:** 12-bit A/D for each sensor
Minimum dynamic range is 8-bits
- BEND SENSORS:** Data Glove 14 Ultra:
Proprietary fiber optic based flexor technology. 2 sensors per finger, 1 sensor between each finger
Data Glove 5 Ultra:
Proprietary fiber optic based flexor technology. 1 sensor per finger, measures average of knuckle and first joint.
- COMPUTER INTERFACE:** Full-speed USB *or*
RS 232 (3-wire)
GND, TX, RX
115kbps
8 data bits, 1 stop bit, no parity
- POWER SUPPLY:** Maximum 5 V DC
Center positive DC power connector
- SAMPLING RATE:** The full hand (all available sensors) may be sampled at least 60 times per second.

At 5DT we constantly try to improve our products. As a result, these specifications are subject to change without prior notice.

Appendix B – Serial Protocol

The data glove is connected to the RS-232 port of the computer. The 5DT Data Glove Ultra Wireless Kit is a transmit-only device, as it does not respond to any commands sent to it.

The 5DT Data Glove Ultra has the following serial settings:

- 115200 bits per second
- 8 data bits, 1 stop bit, no parity
- Hardware handshaking: none

The glove data arrives in packets. If two gloves' data is present on the data stream, then the packets will alternate from one glove then the other. It is possible to distinguish the two by identifying the glove type from the type byte.

The data packet sent by the glove is structured as follows:

| Byte No. | Byte | Byte No. | Byte |
|----------|-----------|----------|-------------|
| 1 | Start | 17 | 9LI 10HI |
| 2 | Type Byte | 18 | 10Lu 10LI |
| 3 | Version | 19 | 11HI 11Lu |
| 4 | 1HI 1Lh | 20 | 11LI 12HI |
| 5 | 1LI 2HI | 21 | 12Lu 12LI |
| 6 | 2Lu 2LI | 22 | 13HI 13Lu |
| 7 | 3HI 3Lu | 23 | 13LI 14HI |
| 8 | 3LI 4HI | 24 | 14Lu 14LI |
| 9 | 4Lu 4LI | 25 | 15HI 15Lu |
| 10 | 5HI 5Lu | 26 | 15LI 16HI |
| 11 | 5LI 6HI | 27 | 16Lu 16LI |
| 12 | 6Lu 6LI | 28 | Checksum |
| 13 | 7HI 7Lu | 29 | Footer |
| 14 | 7LI 8HI | | |
| 15 | 8Lu 8LI | | |
| 16 | 9HI 9Lu | | |

Table 6 - 5DT Data Glove 14 Ultra Packet Description

The bytes of the packet are defined as follows:

Header

Packet start byte defined as the '<' character. Hex value: **0x3c**

Glove Type

The glove type identifier byte. The following values are defined:

- 0x00** – Data Glove14 Ultra wired right hand glove
- 0x01** – Data Glove14 Ultra wired left hand glove
- 0x02** – Data Glove14 Ultra wireless right hand glove (Port A)
- 0x03** – Data Glove14 Ultra wireless left hand glove (Port A)
- 0x06** – Data Glove14 Ultra wireless right hand glove (Port B)
- 0x07** – Data Glove14 Ultra wireless left hand glove (Port B)
- 0x10** – Data Glove 5 Ultra wired right hand glove
- 0x11** – Data Glove 5 Ultra wired left hand glove
- 0x12** – Data Glove 5 Ultra wireless right hand glove (Port A)
- 0x13** – Data Glove 5 Ultra wireless left hand glove (Port A)
- 0x16** – Data Glove 5 Ultra wireless right hand glove (Port B)
- 0x17** – Data Glove 5 Ultra wireless left hand glove (Port B)

Version

The firmware version byte. The upper nibble (Vmaj) represents the version major, the lower nibble (Vmin) represents the version minor.

xHI

The upper four bits of the 12-bit sensor value, or the lower nibble of the MSB of the 12-bit sensor value.

xLh

The middle four bits of the 12-bit sensor value, or the upper nibble of the LSB of the 12-bit sensor value.

xLI

The lower four bits of the 12-bit sensor value, or the lower nibble of the LSB of the 12-bit sensor value.

Checksum

The 8-bit checksum is computed by adding all the data bytes.

Footer

Packet stop byte defined as the '>' character. Hex value: **0x3e**

Appendix C – SDK Function Descriptions

The following functions are available in the SDK:

```
fdGlove *fdOpen(char *pPort)
```

Initializes the glove device on the specified port.

Return value

Returns a pointer to the glove device (`fdGlove *`). NULL is returned if an error occurred.

Parameter

`pPort`

Windows and Linux - Pointer to a zero terminated ASCII string containing the name of the communication port. Valid values on Windows range from "COM1" to "COM8" and also "USB" followed by the USB number. Unix/Linux port names will differ, see section 6.2.3 for more details.

Mac - Pointer to a zero terminated ASCII string containing the name of the glove. Valid values are "DG5U_L", "DG5U_R", "DG14U_L" and "DG14U_R".

Remarks

Do not attempt to alter the contents of the returned pointer directly, use the functions provided instead.

If the wireless kit is used and there are 2 gloves connected, simply call `fdOpen()` twice with the same port name as parameter to get hold of the second glove. Remember to store the returned glove pointer of the second call in a different pointer variable so that you can access both gloves individually. You may now use the second glove pointer in any way you would use any other glove pointer.

```
int fdClose(fdGlove *pFG)
```

Frees the glove device and communications port.

Return value

Returns nonzero if successful, zero if an error occurred.

Parameters

`pFG`

Pointer to a glove device. This is the value returned by `fdOpen()`.

Remarks

It is important to call this function when you are finished using the glove.

```
int fdScanUSB(unsigend short *aPID, int &nNumMax)
```

Scans the USB for available gloves.

Return value

Returns the number of gloves found during the scan.

Parameters

aPID

Pointer to an unsigned short array of length nNumMax. The USB Product IDs (PIDs) of the gloves found are returned in this array. The following PIDs are defined:

DG14U_R – Data Glove 14 Ultra Right-hand

DG14U_L – Data Glove 14 Ultra Left-hand

DG5U_R – Data Glove 5 Ultra Right-hand

DG5U_L – Data Glove 5 Ultra Left-hand

nNumMax

The length of the aPID array. The number of gloves found is also returned in this parameter.

Remarks

None.

```
int fdGetGloveHand(fdGlove *pFG)
```

Obtains the handedness (left or right handed) of the glove.

Return value

Returns either FD_HAND_LEFT or FD_HAND_RIGHT, as defined by the enumerated type EfdGloveHand.

Parameters

pFG

Pointer to a glove device. This is the value returned by fdOpen().

Remarks

None.

```
int fdGetGloveType(fdGlove *pFG)
```

Obtains the type of the currently connected glove.

Return value

Returns one of FD_GLOVENONE, FD_GLOVE7, FD_GLOVE7W, FD_GLOVE16, FD_GLOVE16W, FD_GLOVE14U, FD_GLOVE14UW, or FD_GLOVE14U_USB, as defined by the enumerated type EfdGloveTypes.

Parameters

pFG

Pointer to a glove device. This is the value returned by fdOpen().

Remarks

FD_GLOVE7 and FD_GLOVE7W refer to the original 5+2 (tilt angles) sensor glove (5DT Data Glove 5). The W suffix indicates a wireless model. FD_GLOVE16 and FD_GLOVE16W refer to the Data Glove 16. FD_GLOVE14, FD_GLOVE14W, and FD_GLOVE14_USB refer to the Data Glove 14 Ultra. The USB suffix refers to the Universal Serial Bus interface. In order to accommodate all glove types the fdGetNumSensors() function currently returns 18 sensors. The additional two

sensors are defined as the original tilt angles that are not present in the 16-sensor glove. See the description of `fdGetNumSensors()` for more details.

```
int fdGetNumSensors(fdGlove *pFG)
```

Obtains the number of available sensors values the driver can make available.

Return value

Returns the number of sensors. Currently it is fixed at 18, but future driver releases may differ.

Parameters

`pFG`

Pointer to a glove device. This is the value returned by `fdOpen()`.

Remarks

Although the 5-sensor glove can measure only average flexure, the driver will attempt to fill in missing values. The number of sensors returned can therefore be of a higher dimension. The enumerated type `EfdSensors` defines the finger mapping for each sensor.

```
void fdGetSensorRawAll(fdGlove *pFG, unsigned short *pData)
```

Obtains the most recent raw sensor values from the currently connected glove.

Return value

None.

Parameters

`pFG`

Pointer to a glove device. This is the value returned by `fdOpen()`.

`pData`

Pointer to an array of 16-bit integers that will contain the raw sensor values. The size of the array must always match the value returned by `fdGetNumSensors()`.

Remarks

Currently the raw sensor samples are all 12-bit unsigned values. The range is therefore from 0 to 4095. Note that this is not the dynamic range of the sensors. There can be severe offset values associated with each sensor. The enumerated type `EfdSensors` defines the finger mapping for each sensor.

```
unsigned short fdGetSensorRaw(fdGlove *pFG, int nSensor)
```

Obtains the most recent raw sensor value for a specific sensor from the currently connected glove.

Return value

Returns a 16-bit integer. See `fdGetSensorRawAll()` for details.

Parameters

`pFG`

Pointer to a glove device. This is the value returned by `fdOpen()`.

`nSensor`

Index of the sensor that is being queried. The value must lie in the range given by the enumerated type `EfdSensors`, or alternatively from zero to the value returned by `fdGetNumSensors()` minus one.

Remarks

The enumerated type `EfdSensors` defines the finger mapping for each sensor.

```
void fdSetSensorRawAll(fdGlove *pFG, unsigned short *pData)
```

Forces the raw value for all the sensors.

Return value

None.

Parameters

`pFG`

Pointer to a glove device. This is the value returned by `fdOpen()`.

`pData`

Pointer to an array of 16-bit integers that will contain the raw sensor values. The size of the array must always match the value returned by `fdGetNumSensors()`.

Remarks

Currently the raw sensor samples are all 12-bit unsigned values. The range is therefore from 0 to 4095. The enumerated type `EfdSensors` defines the finger mapping for each sensor. Forcing a sensor value will result in a raw and scaled output other than the default zero. Values that can be mapped will be overwritten, rendering the forced value void.

```
void fdSetSensorRaw(fdGlove *pFG, int nSensor, unsigned short nRaw)
```

Forces the raw value for a specific sensor.

Return value

None.

Parameters

`pFG`

Pointer to a glove device. This is the value returned by `fdOpen()`.

`nSensor`

Index of the sensor that is being set. The value must lie in the range given by the enumerated type `EfdSensors`, or alternatively from zero to the value returned by `fdGetNumSensors()` minus one.

`nRaw`

16-bit raw value of the sensor. If the sensor is unmapped, the scaling calculations will proceed as normal.

Remarks

The enumerated type `EfdSensors` defines the finger mapping for each sensor. This function is only useful for sensors that cannot be mapped by a specific hardware device. Forcing a sensor value will result in a raw and scaled

output other than the default zero. Values that can be mapped will be overwritten, rendering the forced value void.

```
void fdGetSensorScaledAll(fdGlove *pFG, float *pData)
```

Obtains the most recent scaled (auto-calibrated) sensor values from the currently connected glove.

Return value

None.

Parameters

pFG

Pointer to a glove device. This is the value returned by `fdOpen()`.

pData

Pointer to an array of floating point numbers that will contain the scaled sensor values. The size of the array must always match the value returned by `fdGetNumSensors()`.

Remarks

The sensor range is a value from zero to the value defined by the `fdSetSensorMax()` and `fdSetSensorMaxAll()` functions. The glove driver defaults to a range of [0...1]. The automatic calibration process is described in section 6. The enumerated type `EfdSensors` defines the finger mapping for each sensor.

```
float fdGetSensorScaled(fdGlove *pFG, int nSensor)
```

Obtains the most recent scaled (auto-calibrated) value for a specific sensor from the currently connected glove.

Return value

Returns a floating point sensor value.

Parameters

pFG

Pointer to a glove device. This is the value returned by `fdOpen()`.

nSensor

Index of the sensor that is being queried. The value must lie in the range given by the enumerated type `EfdSensors`, or alternatively from zero to the value returned by `fdGetNumSensors()` minus one.

Remarks

The sensor range is a value from zero to the value defined by the `fdSetSensorMax()` and `fdSetSensorMaxAll()` functions. The glove driver defaults to a range of [0...1]. The automatic calibration process is described in section 6. The enumerated type `EfdSensors` defines the finger mapping for each sensor.

```
int fdGetNumGestures(fdGlove *pFG)
```

Obtains the number of available gestures that can be recognized by the glove driver.

Return value

Returns the number of available gestures. Currently 16 different gestures are supported. Refer to section 5 for details.

Parameters

`pFG`

Pointer to a glove device. This is the value returned by `fdOpen()`.

Remarks

None.

```
int fdGetGesture(fdGlove *pFG)
```

Obtains the current gesture being performed.

Return value

Returns the current gesture being performed. Refer to section 5 for details.

Parameters

`pFG`

Pointer to a glove device. This is the value returned by `fdOpen()`.

Remarks

None.

```
void fdGetCalibrationAll(fdGlove *pFG, unsigned short *pUpper, unsigned short *pLower)
```

Obtains the current auto-calibration settings of the driver.

Return value

None.

Parameters

`pFG`

Pointer to a glove device. This is the value returned by `fdOpen()`.

`pUpper` and `pLower`

Arrays of 16-bit unsigned integers that will contain the maximum and minimum raw sensor values. The size of each array must always match the value returned by `fdGetNumSensors()`. Refer to section 6 for details.

Remarks

None.

```
void fdGetCalibration(fdGlove *pFG, int nSensor, unsigned short *pUpper, unsigned short *pLower)
```

Obtains the current auto-calibration settings of the driver for a specific sensor.

Return value

None.

Parameters

`pFG`

Pointer to a glove device. This is the value returned by `fdOpen()`.

`nSensor`

Index of the sensor that is being queried. The value must lie in the range given by the enumerated type `EfdSensors`, or alternatively from zero to the value returned by `fdGetNumSensors()` minus one.

`pUpper` and `pLower`

Pointers to 16-bit unsigned integers containing the maximum and minimum raw sensor values. Refer to section 6 for details.

Remarks

None.

```
void fdSetCalibrationAll(fdGlove *pFG, unsigned short *pUpper, unsigned short *pLower)
```

Resets the current auto-calibration settings of the driver to user defined values.

Return value

None.

Parameters

`pFG`

Pointer to a glove device. This is the value returned by `fdOpen()`.

`pUpper` and `pLower`

Arrays of 16-bit unsigned integers containing the maximum and minimum raw sensor values. The size of each array must always match the value returned by `fdGetNumSensors()`. Refer to section 6 for details.

Remarks

For unmapped sensors it would be sensible to set the upper and lower calibration settings above and below the raw value forced with `fdSetSensorRaw()` and `fdSetSensorRawAll()`.

```
void fdSetCalibration(fdGlove *pFG, int nSensor, unsigned short nUpper, unsigned short nLower)
```

Resets the current auto-calibration settings of the driver for a specific sensor to user defined values.

Return value

None.

Parameters

`pFG`

Pointer to a glove device. This is the value returned by `fdOpen()`.

`nSensor`

Index of the sensor that is being set. The value must lie in the range given by the enumerated type `EfdSensors`, or alternatively from zero to the value returned by `fdGetNumSensors()` minus one.

`nUpper` and `nLower`

16-bit unsigned integers containing the maximum and minimum raw sensor values. Refer to section 6 for details.

Remarks

For unmapped sensors it would be sensible to set the upper and lower calibration settings above and below the raw value forced with `fdSetSensorRaw()` and `fdSetSensorRawAll()`.

```
void fdResetCalibration(fdGlove *pFG)
```

Resets the internal auto-calibration settings of the driver to appropriate default values (for all the sensors).

Return value

None.

Parameters

`pFG`

Pointer to a glove device. This is the value returned by `fdOpen()`.

Remarks

This function is similar to the `fdSetCalibrationAll()` function with each of the upper and lower calibration array values set to 0 and 4095 respectively. This function, or any of the other calibration functions, should be called whenever the application starts up or the glove changes users during run-time. For unmapped sensors the upper and lower calibration values are set to 4095 and 0 respectively, which is the *inverse* of the auto-calibration settings. If auto calibration was turned off, it is turned on again.

```
void fdResetCalibration(fdGlove *pFG, int nSensor)
```

Resets the internal auto-calibration settings of the driver to appropriate default values (for a specific sensor).

Return value

None.

Parameters

`pFG`

Pointer to a glove device. This is the value returned by `fdOpen()`.

`nSensor`

Index of the sensor whose calibration values are to be reset.

Remarks

This function is similar to the `fdSetCalibrationAll()` function with each of the upper and lower calibration array values set to 0 and 4095 respectively. This function, or any of the other calibration functions, should be called whenever the application starts up or the glove changes users during run-time. For unmapped sensors the upper and lower calibration values are set to 4095 and 0 respectively, which is the *inverse* of the auto-calibration settings. If auto calibration was turned off, it is turned on again.

```
void fdGetSensorMaxAll(fdGlove *pFG, float *pMax)
```

Obtains the maximum scaled value for each sensor.

Return value

None.

Parameters

64

pFG

Pointer to a glove device. This is the value returned by `fdOpen()`.

pMax

Array of floating point values that will contain the maximum scaled sensor values. The size of the array must always match the value returned by `fdGetNumSensors()`.

Remarks

The glove driver defaults to a maximum scaled value of 1 for each sensor.

```
float fdGetSensorMax(fdGlove *pFG, int nSensor)
```

Obtains the maximum scaled value for a specific sensor.

Return value

Returns the maximum scaled values of the sensor.

Parameters

pFG

Pointer to a glove device. This is the value returned by `fdOpen()`.

nSensor

Index of the sensor that is being queried. The value must lie in the range given by the enumerated type `EfdSensors`, or alternatively from zero to the value returned by `fdGetNumSensors()` minus one.

Remarks

The glove driver defaults to a maximum scaled value of 1 for each sensor.

```
void fdSetSensorMaxAll(fdGlove *pFG, float *pMax)
```

Sets the maximum scaled value for each sensor.

Return value

None.

Parameters

pFG

Pointer to a glove device. This is the value returned by `fdOpen()`.

pMax

Array of floating point values that contains the maximum scaled sensor values. The size of the array must always match the value returned by `fdGetNumSensors()`.

Remarks

The glove driver defaults to a maximum scaled value of 1 for each sensor.

```
void fdSetSensorMax(fdGlove *pFG, int nSensor, float fMax)
```

Sets the maximum scaled value for a specific sensor.

Return value

None.

Parameters

pFG

Pointer to a glove device. This is the value returned by `fdOpen()`.

`nSensor`

Index of the sensor that is being set. The value must lie in the range given by the enumerated type `EfdSensors`, or alternatively from zero to the value returned by `fdGetNumSensors()` minus one.

`fMax`

A floating point value that contains the maximum scaled sensor value.

Remarks

The glove driver defaults to a maximum scaled value of 1 for each sensor.

```
void fdGetThresholdAll(fdGlove *pFG, float *pUpper, float *pLower)
```

Obtains the current gesture recognition threshold settings of the driver.

Return value

None.

Parameters

`pFG`

Pointer to a glove device. This is the value returned by `fdOpen()`.

`pUpper` and `pLower`

Arrays of floating point numbers that will contain the maximum and minimum threshold values. The size of each array must always match the value returned by `fdGetNumSensors()`. Refer to section 6 for details.

Remarks

None.

```
void fdGetThreshold(fdGlove *pFG, int nSensor, float *pUpper, float *pLower)
```

Obtains the current gesture recognition threshold settings of the driver for a specific sensor.

Return value

None.

Parameters

`pFG`

Pointer to a glove device. This is the value returned by `fdOpen()`.

`nSensor`

Index of the sensor that is being queried. The value must lie in the range given by the enumerated type `EfdSensors`, or alternatively from zero to the value returned by `fdGetNumSensors()` minus one.

`pUpper` and `pLower`

Pointers to floating point numbers that will contain the maximum and minimum threshold values. Refer to section 6 for details.

Remarks

None.

```
void fdSetThresholdAll(fdGlove *pFG, float *pUpper, float *pLower)
```

Sets the current gesture recognition threshold settings of the driver.

Return value

None.

Parameters

pFG

Pointer to a glove device. This is the value returned by `fdOpen()`.

pUpper and *pLower*

Arrays of floating point numbers that contains the maximum and minimum threshold values. The size of each array must always match the value returned by `fdGetNumSensors()`. Refer to section 6 for details.

Remarks

None.

```
void fdSetThreshold(fdGlove *pFG, int nSensor, float fUpper, float fLower)
```

Sets the current gesture recognition threshold settings of the driver for a specific sensor.

Return value

None.

Parameters

pFG

Pointer to a glove device. This is the value returned by `fdOpen()`.

nSensor

Index of the sensor that is being set. The value must lie in the range given by the enumerated type `EfdSensors`, or alternatively from zero to the value returned by `fdGetNumSensors()` minus one.

fUpper and *fLower*

Floating point numbers that contain the maximum and minimum threshold values. Refer to section 6 for details.

Remarks

None.

```
void fdGetGloveInfo(fdGlove *pFG, unsigned char *pData)
```

Obtains the information data block of the currently connected glove.

Return value

None.

Parameters

pFG

Pointer to a glove device. This is the value returned by `fdOpen()`.

pData

Array of 32 bytes that will contain the information data.

Remarks

The information data is specified in the glove user's manual. The size of the information block is always 32 bytes.

```
void fdGetDriverInfo(fdGlove *pFG, unsigned char *pData)
```

Obtains the information data block of the driver.

Return value

None.

Parameters

pFG

Pointer to a glove device. This is the value returned by `fdOpen()`.

pData

Array of 32 bytes that will contain the information data.

Remarks

The information data is a zero terminated string that contains driver information. The size of the information block is always 32 bytes.

```
void fdSetCallback(fdGlove *pFG, void *pFunc, LPVOID param)
```

Sets the Callback function and associated parameters.

Return value

None.

Parameters

pFG

Pointer to a glove device. This is the value returned by `fdOpen()`.

pFunc

Pointer to the callback function. This should be cast to void.

param

The parameter to be given to the callback function.

Remarks

This callback function is called every time a new packet is received by the driver. Example code snippet to register a callback function:

```
fdSetCallback(pGlove, (void *) &(DriverUpdate), this);
```

```
int fdGetPacketRate(fdGlove *pFG)
```

Obtains the latest packet rate.

Return value

Returns the latest available packet rate as an integer.

Parameters

pFG

Pointer to a glove device. This is the value returned by `fdOpen()`.

Remarks

None.

```
bool fdNewData(fdGlove *pFG)
```

Indicates if the driver has received new data since this function was last called.

Return value

Returns `true` if new data is available.

Parameters

`pFG`

Pointer to a glove device. This is the value returned by `fdOpen()`.

Remarks

This function may be used as an alternative to using the callback function to trigger events upon the arrival of new glove data.

```
int fdGetFWVersionMajor(fdGlove *pFG)
```

Obtains the major version of the glove's firmware.

Return value

Returns the major version of the glove's firmware as an integer.

Parameters

`pFG`

Pointer to a glove device. This is the value returned by `fdOpen()`.

Remarks

This function has currently only been implemented for the Data Glove 14 Ultra. It will return 0 if the glove type isn't a Data Glove 14 Ultra variant. Example: if the firmware version is 3.2 then 3 is returned.

```
int fdGetFWVersionMinor(fdGlove *pFG)
```

Obtains the minor version of the glove's firmware.

Return value

Returns the minor version of the glove's firmware as an integer.

Parameters

`pFG`

Pointer to a glove device. This is the value returned by `fdOpen()`.

Remarks

This function has currently only been implemented for the Data Glove 14 Ultra. It will return 0 if the glove type isn't a Data Glove 14 Ultra variant. Example: if the firmware version is 3.2 then 2 is returned.

```
bool fdGetAutoCalibrate(fdGlove *pFG)
```

Indicates if the driver is currently auto calibrating.

Return value

Returns `true` if the driver is currently auto calibrating.

Parameters

`pFG`

Pointer to a glove device. This is the value returned by `fdOpen()`.

Remarks

None.

```
bool fdSetAutoCalibrate(fdGlove *pFG, bool bAutoCalibrate)
```

Turns auto calibration **on** or **off**.

Return value

Returns **true** if the driver is currently auto calibrating.

Parameters

`pFG`

Pointer to a glove device. This is the value returned by `fdOpen()`.

`bAutoCalibrate`

Boolean value indicating if auto calibration should be turned **on** (**true**) or **off** (**false**).

Remarks

Turn auto calibration off after calibrating to prevent the calibration values from being too extreme, thus causing scaled values of poor quality.

```
bool fdSaveCalibration(fdGlove *pFG, const char *pFileName)
```

Saves the current calibration values to file.

Return value

Returns **true** if the save operation completed successfully.

Parameters

`pFG`

Pointer to a glove device. This is the value returned by `fdOpen()`.

`pFileName`

A null-terminated string containing the path and filename.

Remarks

Save your calibration values to file to prevent you from having to calibrate the glove every time you start your application.

```
bool fdLoadCalibration(fdGlove *pFG, const char *pFileName)
```

Loads calibration values from file. Current calibration values are discarded.

Return value

Returns **true** if the load operation completed successfully.

Parameters

`pFG`

Pointer to a glove device. This is the value returned by `fdOpen()`.

`pFileName`

A null-terminated string containing the path and filename.

Remarks

None.

Appendix D – C# SDK Function Descriptions

The functions described here are member functions of the class `CfdGlove`. To setup your application to use the C# driver see section 6.2.2.

```
void Open(string sPort)
```

Initializes the glove device on the specified port.

Return value

None.

Parameters

`sPort`

String containing the name of the communication port. If the glove is connected to a USB port, `sPort` should contain "USB" followed by a number, for example: "USB0". If the glove is connected to a serial port, `sPort` should contain "COM" followed by the COM port number the glove is connected to, for example: "COM5".

Remarks

None.

```
int Close()
```

Frees the glove device and communications port.

Return value

Returns nonzero if successful, zero if an error occurred.

Parameters

None.

Remarks

It is important to call this function when you are finished using the glove.

```
int GetGloveHand()
```

Obtains the handedness (left or right handed) of the glove.

Return value

Returns either `FD_HAND_RIGHT` or `FD_HAND_LEFT`, as defined by the enumerated type `FD_HAND`.

Parameters

None.

Remarks

None.

```
int GetGloveType()
```

Obtains the type of the currently connected glove.

Return value

Returns one of `FD_GLOVENONE`, `FD_GLOVE7`, `FD_GLOVE7W`, `FD_GLOVE16`, `FD_GLOVE16W`, `FD_GLOVE14U`, `FD_GLOVE14UW`, or `FD_GLOVE14U_USB`, as defined by the enumerated type `FD_GLOVE`.

Parameters

None.

Remarks

`FD_GLOVE7` and `FD_GLOVE7W` refer to the original 5+2 (tilt angles) sensor glove (5DT Data Glove 5). The `W` suffix indicates a wireless model. `FD_GLOVE16` and `FD_GLOVE16W` refer to the Data Glove 16. `FD_GLOVE14`, `FD_GLOVE14W`, and `FD_GLOVE14_USB` refer to the Data Glove 14 Ultra. The `USB` suffix refers to the Universal Serial Bus interface. In order to accommodate all glove types the `GetNumSensors()` function currently returns 18 sensors. The additional two sensors are defined as the original tilt angles that are not present in the 16-sensor glove. See the description of `GetNumSensors()` for more details.

```
int GetNumSensors()
```

Obtains the number of available sensors values the driver can make available.

Return value

Returns the number of sensors. Currently it is fixed at 18, but future driver releases may differ.

Parameters

None.

Remarks

Although the 5-sensor glove can measure only average flexure, the driver will attempt to fill in missing values. The number of sensors returned can therefore be of a higher dimension. The enumerated type `EfdSensors` defines the finger mapping for each sensor.

```
void GetSensorRawAll(ref ushort[] aSensorRawVals)
```

Obtains the most recent raw sensor values from the currently connected glove.

Return value

None.

Parameters

`aSensorRawVals`

Array of 16-bit integers that will contain the raw sensor values. If the size of `aSensorRawVals` is not equal to the value returned by `GetNumSensors()`, `aSensorRawVals` will be reinitialized inside the function.

Remarks

Currently the raw sensor samples are all 12-bit unsigned values. The range is therefore from 0 to 4095. Note that this is not the dynamic range of the sensors. There can be severe offset values associated with each sensor. The enumerated type `EfdSensors` defines the finger mapping for each sensor.

```
unsigned short GetSensorRaw(int nSensor)
```

Obtains the most recent raw sensor value for a specific sensor from the currently connected glove.

Return value

Returns a 16-bit integer. See `GetSensorRawAll()` for details.

Parameters

`nSensor`

Index of the sensor that is being queried. The value must lie in the range given by the enumerated type `EfdSensors`, or alternatively from zero to the value returned by `GetNumSensors()` minus one.

Remarks

The enumerated type `EfdSensors` defines the finger mapping for each sensor.

```
void GetSensorScaledAll(ref float[] aSensorScaledVals)
```

Obtains the most recent scaled (auto-calibrated) sensor values from the currently connected glove.

Return value

None.

Parameters

`aSensorScaledVals`

Pointer to an array of floating point numbers that will contain the scaled sensor values. If the size of `aSensorScaledVals` is not the equal to the value returned by `GetNumSensors()`, `aSensorScaledVals` will be reinitialized inside the function.

Remarks

The glove driver defaults to a range of [0...1]. The enumerated type `EfdSensors` defines the finger mapping for each sensor.

```
float GetSensorScaled(int nSensor)
```

Obtains the most recent scaled (auto-calibrated) value for a specific sensor from the currently connected glove.

Return value

Returns a floating point sensor value.

Parameters

`nSensor`

Index of the sensor that is being queried. The value must lie in the range given by the enumerated type `EfdSensors`, or alternatively from zero to the value returned by `GetNumSensors()` minus one.

Remarks

The glove driver defaults to a range of [0...1]. The enumerated type `EfdSensors` defines the finger mapping for each sensor.

```
int GetNumGestures()
```

Obtains the number of available gestures that can be recognized by the glove driver.

Return value

Returns the number of available gestures. Currently 16 different gestures are supported.

Parameters

None.

Remarks

None.

```
int GetGesture()
```

Obtains the current gesture being performed.

Return value

Returns the current gesture being performed.

Parameters

None.

Remarks

None.

```
void GetCalibrationAll(ref ushort[] aUpperVals, ref  
ushort[] aLowerVals)
```

Obtains the current auto-calibration settings of the driver.

Return value

None.

Parameters

`aUpperVals` and `aLowerVals`

Arrays of 16-bit unsigned integers that will contain the maximum and minimum raw sensor values. If the size of `aUpperVals` or `aLowerVals` is not equal to the value returned by `GetNumSensors()`, it will be reinitialized inside the function.

Remarks

None.

```
void GetCalibration(int nSensor, ref ushort uUpper, ref  
ushort uLower)
```

Obtains the current auto-calibration settings of the driver for a specific sensor.

Return value

None.

Parameters

`nSensor`

Index of the sensor that is being queried. The value must lie in the range given by the enumerated type `EfdSensors`, or alternatively from zero to the value returned by `GetNumSensors()` minus one.

`uUpper` and `uLower`

References to 16-bit unsigned integers containing the maximum and minimum raw sensor values.

Remarks

None.

```
void SetCalibrationAll(ushort[] aUpperVals, ushort[]  
aLowerVals)
```

Resets the current auto-calibration settings of the driver to user defined values.

Return value

None.

Parameters

`aUpperVals` and `aLowerVals`

Arrays of 16-bit unsigned integers containing the maximum and minimum raw sensor values. The size of `aUpperVals` or `aLowerVals` must be equal to the value returned by `GetNumSensors()`.

Remarks

None.

```
void SetCalibration(int nSensor, ushort uUpper, ushort  
uLower)
```

Resets the current auto-calibration settings of the driver for a specific sensor to user defined values.

Return value

None.

Parameters

`nSensor`

Index of the sensor that is being set. The value must lie in the range given by the enumerated type `EfdSensors`, or alternatively from zero to the value returned by `GetNumSensors()` minus one.

`uUpper` and `uLower`

16-bit unsigned integers containing the maximum and minimum raw sensor values.

Remarks

None.

```
void ResetCalibration()
```

Resets the internal auto-calibration settings of the driver to appropriate default values (for all the sensors).

Return value

None.

Parameters

None.

Remarks

This function is similar to the `SetCalibrationAll()` function with each of the upper and lower calibration array values set to 0 and 4095 respectively. This function, or any of the other calibration functions, should be called whenever the

application starts up or the glove changes users during run-time. For unmapped sensors the upper and lower calibration values are set to 4095 and 0 respectively, which is the *inverse* of the auto-calibration settings. If auto calibration was turned off, it is turned on again.

```
void ResetCalibration(int nSensor)
```

Resets the internal auto-calibration settings of the driver to appropriate default values (for a specific sensor).

Return value

None.

Parameters

`nSensor`

Index of the sensor whose calibration values are to be reset.

Remarks

This function is similar to the `SetCalibrationAll()` function with each of the upper and lower calibration array values set to 0 and 4095 respectively. This function, or any of the other calibration functions, should be called whenever the application starts up or the glove changes users during run-time. For unmapped sensors the upper and lower calibration values are set to 4095 and 0 respectively, which is the *inverse* of the auto-calibration settings. If auto calibration was turned off, it is turned on again.

```
void GetThresholdAll(ref float[] fUpperVals, ref[] float fLowerVals)
```

Obtains the current gesture recognition threshold settings of the driver.

Return value

None.

Parameters

`fUpperVal` and `fLowerVal`

Arrays of floating point numbers that will contain the maximum and minimum threshold values. If the size of `aUpperVals` or `aLowerVals` is not equal to the value returned by `GetNumSensors()`, it will be reinitialized inside the function.

Remarks

None.

```
void GetThreshold(int nSensor, ref float fUpperVal, ref float fLowerVal)
```

Obtains the current gesture recognition threshold settings of the driver for a specific sensor.

Return value

None.

Parameters

`nSensor`

Index of the sensor that is being queried. The value must lie in the range given by the enumerated type `EfdSensors`, or alternatively from zero to the value returned by `GetNumSensors()` minus one.

`fUpperVal` and `fLowerVal`

Pointers to floating point numbers that will contain the maximum and minimum threshold values.

Remarks

None.

```
void SetThresholdAll(float[] aUpperVals, float[] aLowerVals)
```

Sets the current gesture recognition threshold settings of the driver.

Return value

None.

Parameters

`aUpperVals` and `aLowerVals`

Arrays of floating point numbers that contains the maximum and minimum threshold values. The size of each array must always match the value returned by `GetNumSensors()`.

Remarks

None.

```
void SetThreshold(int nSensor, float fUpperVal, float fLowerVal)
```

Sets the current gesture recognition threshold settings of the driver for a specific sensor.

Return value

None.

Parameters

`nSensor`

Index of the sensor that is being set. The value must lie in the range given by the enumerated type `EfdSensors`, or alternatively from zero to the value returned by `GetNumSensors()` minus one.

`fUpperVal` and `fLowerVal`

Floating point numbers that contain the maximum and minimum threshold values.

Remarks

None.

```
string GetGloveInfo()
```

Obtains the information data block of the currently connected glove.

Return value

String of 32 characters that will contain the information data.

Parameters

None.

Remarks

The information data is specified in the glove user's manual.

```
string GetDriverInfo()
```

Obtains the information data block of the driver.

Return value

String of 32 characters that will contain the information data.

Parameters

None.

Remarks

None.

```
void SetCallBack(EventHandler Event)
```

Sets the Callback function.

Return value

None.

Parameters

Event

Event that should be called when a new packet is received from the driver.

Remarks

This callback function is called every time a new packet is received by the driver.

Example:

```
fdGlove.SetCallBack(new EventHandler(Update));
```

with Update function defined as:

```
void Update(object sender, EventArgs e);
```

```
int GetPacketRate()
```

Obtains the latest packet rate.

Return value

Returns the latest available packet rate as an integer.

Parameters

None.

Remarks

None.

```
bool NewData()
```

Indicates if the driver has received new data since this function was last called.

Return value

Returns `true` if new data is available.

Parameters

None.

Remarks

This function may be used as an alternative to using the callback function to trigger events upon the arrival of new glove data.

```
int GetFWVersionMajor()
```

Obtains the major version of the glove's firmware.

Return value

Returns the major version of the glove's firmware as an integer.

Parameters

None.

Remarks

This function has currently only been implemented for the Data Glove 14 Ultra. It will return 0 if the glove type isn't a Data Glove 14 Ultra variant. For example if the firmware version is 3.2 then 3 is returned.

```
int GetFWVersionMinor()
```

Obtains the minor version of the glove's firmware.

Return value

Returns the minor version of the glove's firmware as an integer.

Parameters

None.

Remarks

This function has currently only been implemented for the Data Glove 14 Ultra. It will return 0 if the glove type isn't a Data Glove 14 Ultra variant. For example if the firmware version is 3.2 then 2 is returned.

```
bool GetAutoCalibrate()
```

Indicates if the driver is currently auto calibrating.

Return value

Returns `true` if the driver is currently auto calibrating.

Parameters

None.

Remarks

None.

```
bool SetAutoCalibrate(bool bAutoCalibrate)
```

Turns auto calibration **on** or **off**.

Return value

Returns **true** if the driver is currently auto calibrating.

Parameters

bAutoCalibrate

Boolean value indicating if auto calibration should be turned **on** (**true**) or **off** (**false**).

Remarks

Turn auto calibration off after calibrating to prevent the calibration values from being too extreme, thus causing scaled values of poor quality.

```
bool SaveCalibration(string sFileName)
```

Saves the current calibration values to file.

Return value

Returns **true** if the save operation completed successfully.

Parameters

sFileName

A string containing the path and filename.

Remarks

Save your calibration values to file to prevent you from having to calibrate the glove every time you start your application.

```
bool LoadCalibration(string sFileName)
```

Loads calibration values from file. Current calibration values are discarded.

Return value

Returns **true** if the load operation completed successfully.

Parameters

sFileName

A string containing the path and filename.

Remarks

None.