Emotiv

Software Development Kit

User Manual for Release 2.0.0.20

TABLE OF CONTENTS

DIRECTORY OF FIGURES

DIRECTORY OF TABLES

DIRECTORY OF LISTINGS

1. Introduction

This document is intended as a guide for Emotiv SDK and SDKLite developers. It describes different aspects of the Emotiv Software Development Kit (SDK), including:

- **Getting Started**: Basic information about installing the Emotiv SDK hardware and software.
- **Emotiv Control Panel™**: Introduction to Emotiv Control Panel, an application that configures and demonstrates the Emotiv detection suites.
- **Emotiv SDK Tools**: Usage guide for EmoKey™ and EmoComposer™, tools that help you develop applications with the Emotiv SDK.
- **Emotiv API Introduction**: Introduction to programming with the Emotiv API and an explanation of the code examples included with the SDK.

If you have any queries beyond the scope of this document, please contact the Emotiv SDK support team.

1.1 Glossary

Affectiv™	The detection suite that deciphers a user's emotional state.
SDK Neuroheadset	The headset worn by the user, which interprets brain signals and sends the information to Emotiv EmoEngine™.
Cognitiv™	The detection suite that recognizes a user's conscious thoughts.
Default Profile	A generic profile template that contains default settings for a new user. See Profile.
Detection	A high-level concept that refers to the proprietary algorithms running on the neuroheadset and in Emotiv EmoEngine which, working together, recognize a specific type of facial expression, emotion, or mental state. Detections are organized into three different suites: Expressiv, Affectiv, and Cognitiv.
EML	EmoComposer™ Markup Language – an XML-based syntax that can be interpreted by EmoComposer to playback predefined EmoState values.
Emotiv API	Emotiv Application Programming Interface: a library of functions, provided by Emotiv to application developers, which enables them to write software applications that work with Emotiv neuroheadsets and the Emotiv detection suites.
Emotiv EPOC™	The neuroheadset that will be available with Emotiv's consumer product.
Emotiv SDK	The Emotiv Software Development Kit: a toolset that allows development of applications and games to interact with Emotiv EmoEngine™ and Emotiv neuroheadsets.
Emotiv SDKLite™	A version of the Emotiv SDK that uses neuroheadset emulation to allow integration with new and existing software. Software developed with the SDKLite will be compatible with the Emotiv EPOC™ headset.
EmoComposer™	An Emotiv EmoEngine™ emulator designed to speed-up the

development of Emotiv-compatible software applications.

Emotiv EmoEngine™	A logical abstraction exposed by the Emotiv API. EmoEngine communicates with the Emotiv neuroheadset, manages user-specific and application-specific settings, and translates the Emotiv detection results into an EmoState.
EmoKey™	Tool to translate EmoStates™ into signals that emulate traditional input devices (such as keyboard).
EmoScript™	A text file containing EML, which can be interpreted by EmoComposer to automate the generation of predefined EmoStates. Also refers to the operational mode of EmoComposer in which this playback occurs.
EmoState™	A data structure containing information about the current state of all activated Emotiv detections. This data structure is generated by Emotiv EmoEngine and reported to applications that use the Emotiv API.
Expressiv™	The detection suite that identifies a user's facial expressions.
Player	Synonym for User.
Profile	A user profile contains user-specific data created and used by the EmoEngine to assist in personalizing Emotiv detection results. When created with Emotiv Control Panel, all users' profiles are saved to the profile.bin file in the Emotiv program files directory.
User	A person who is wearing a neuroheadset and interacting with Emotiv-enabled software. Each user should have a unique profile.

1.2 Trademarks

The following are trademarks of Emotiv.

The absence of a product or service name or logo from this list does not constitute a waiver of Emotiv's trademark or other intellectual property rights concerning that name or logo.

Affectiv™

Cognitiv™

EmoComposer™

EmoKey™

EmoScript™

EmoState™

Emotiv Control Panel™

Emotiv EmoEngine™

Emotiv EPOC™

Emotiv SDKLite™

Expressiv™

1.3 Regulatory Information:

USA - FCC Requirements Part 15 CFR47

This device complies with Part 15 of the FCC Rules. Operation is subject to the following two conditions:

1. This device may not cause harmful interference, and

2. This device must accept any interference received, including interference that may cause undesired operation.

This equipment has been tested and found to comply with the limits for a Class B digital device, pursuant to Part 15 of the FCC Rules. These limits are designed to provide reasonable protection against harmful interference in a residential installation. This equipment generates, uses, and can radiate radio frequency energy and, if not installed and used in accordance with the instructions, may cause harmful interference to radio communications. However, there is no guarantee that interference will not occur in a particular installation. If this equipment does cause harmful interference to radio or television reception which can be determined by turning the radio or television off and on, the user is encouraged to try to correct interference by one or more of the following measures:

1. Reorient or relocate the receiving antenna.

2. Increase the separation between the equipment and receiver.

3. Connect the equipment into an outlet on another circuit.

4. Consult the dealer or an experienced radio/TV technician for help.

NOTE: Modifications not expressly approved by Emotiv could void the user's authority to operate the equipment

Canada

This device complies with RSS-210 of Industry Canada (IC). Operation is subject to the following two conditions:

1. This device may not cause interference, and

2. This device must accept any interference received, including interference that may cause undesired operation of this device.

This Class B digital apparatus complies with Canadian ICES-003. Cet appareil numérique de la Classe B est conforme à la norme NMB-003 du Canada.

Europe, Australia, New Zealand

Product Name and Model: Emotiv EPOC Model 1.0

Product description: EPOC Neuroheadset, USB-01 Transceiver, Hydrator Pack + charger or charge cable

conforms to the following Product Specifications and Regulations:

EMC and Telecom: Class B

ETSI EN 300 440-2 V1.4.1

EN 301 489-1

EN 301 489-3

AS/NZS CISPR22 :2009

AS/NZS 4268 :2008

FCC CFR 47 Part 15C (identifiers XUE-EPOC01, XUE-USBD01)

Safety:

EN 60950-1:2006

IEC 60950-1:2005 (2nd Edition)

AS/NZS 60950.1:2003 including amendments 1, 2 & 3

CB Certificate JPTUV-029914 (TUV Rheinland)

The product herewith complies with the requirements of the Low Voltage Directive 2006/95/EC, the EMC Directive 2004/108/EC, the R&TTE Directive 1999/5/EC, and carries the CE and C-Tick marks accordingly.

Quick Start Guide

Items in the EPOC Headset Kit

Make sure all items are present in your kit before starting.

- Headset Assembly with Rechargeable Lithium battery already installed
- USB Transceiver Dongle
- Hydration Sensor Pack with 16 Sensor Units
- Saline solution
- 50/60Hz 100-250 VAC Battery Charger (US customers) or USB charger (non-US customers)
- CD Installation Disk for Windows XP or Vista (only for EPOC consumer headset. SDKs are delivered electronically)
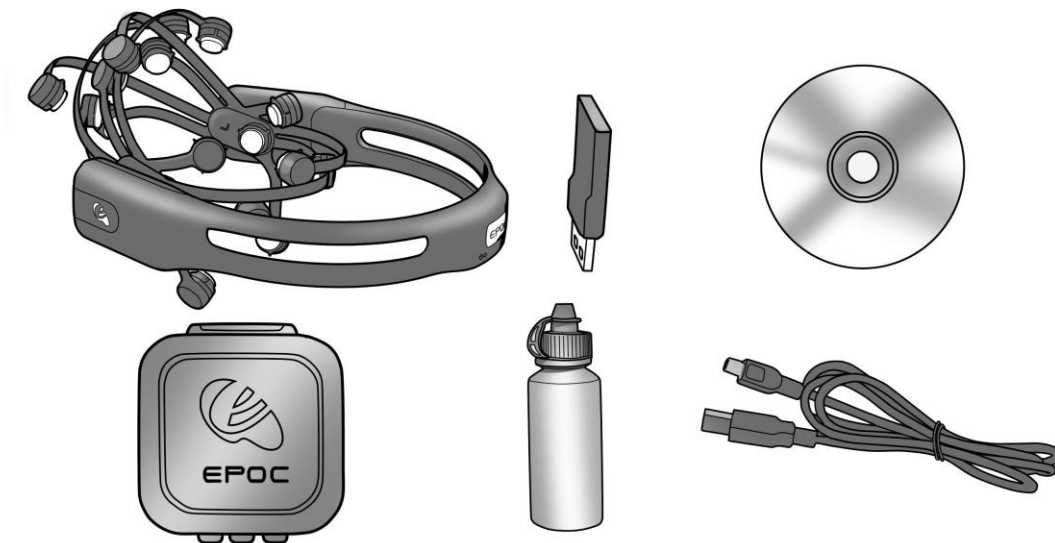
Initial charging of headset

Make sure the small switch on the rear underside of the headset is set to the "Off" position before starting.

Plug the mini USB cable attached to the supplied battery charger into the slot at the top of the headset and to the USB port on your PC or the power cord into a 50 or 60 Hz 100-250 V electrical outlet.
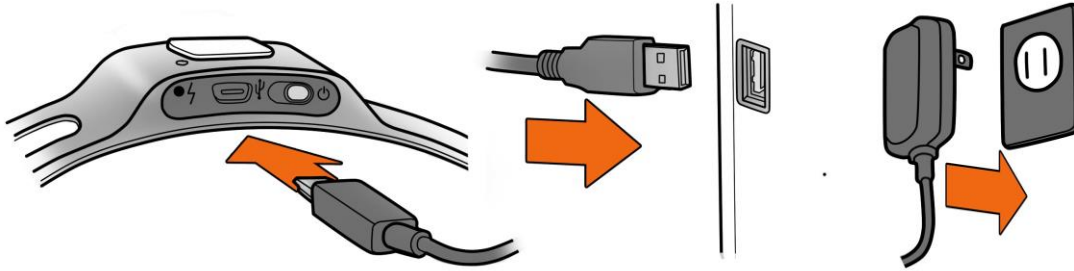
The Lithium battery can be recharged to 100% capacity in approximately 4 hours depending on the initial state of charge. Charging for 30 minutes usually yields about a 10% increase in charge.

The EPOC Headset contains two status LEDs located at the rear and next to the power switch at the back of the headband. When the power switch is set to the "on" position, the rear LED will illuminate and appear blue if there is sufficient charge for correct operation, unless charging is in progress. The charging LED will appear red during battery charging; when the battery is fully-charged, the charging LED will display green.

NOTE: The Headset should not be charged when still on the head.

Software Installation

Insert the supplied EPOC CD setup disk into your computer's CD/DVD drive and follow the step-by-step installation instructions.
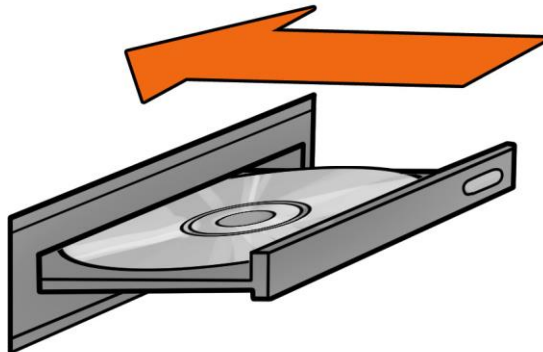
After software installation, start-up the EPOC Control Panel program, loading the Headset Setup screen.

Hydrating the Sensors

Open the Saline Hydration Sensor Pack with the white felt inserts inside. The inserts will eventually be mounted in the headset arms but must be properly wetted with saline solution first. Begin wetting each of the felt inserts with the supplied saline solution. The felts should be wet to the touch, but not soaking wet!

Note: This is standard multipurpose contact lens saline solution and is available from any local drug store in case you run out of solution. However, the bottle supplied with the kit should be sufficient initially. See the User Manual on the EPOC CD setup disk for recommendations.

Add a few drops of saline to saturate the large white hydrator pad attached to the top cover of the hydrator, then close the cover and gently shake the hydrator pack. This will maintain the moisture of the felt pads when they are not in use. Open the pack and check that each of the pads had been wetted. If not fully wetted, then add a drop or two of saline to any pads not sufficiently wet using the dropper bottle. Be careful not to over-wet the pads. If you have connection problems, add more saline to each felt pad.
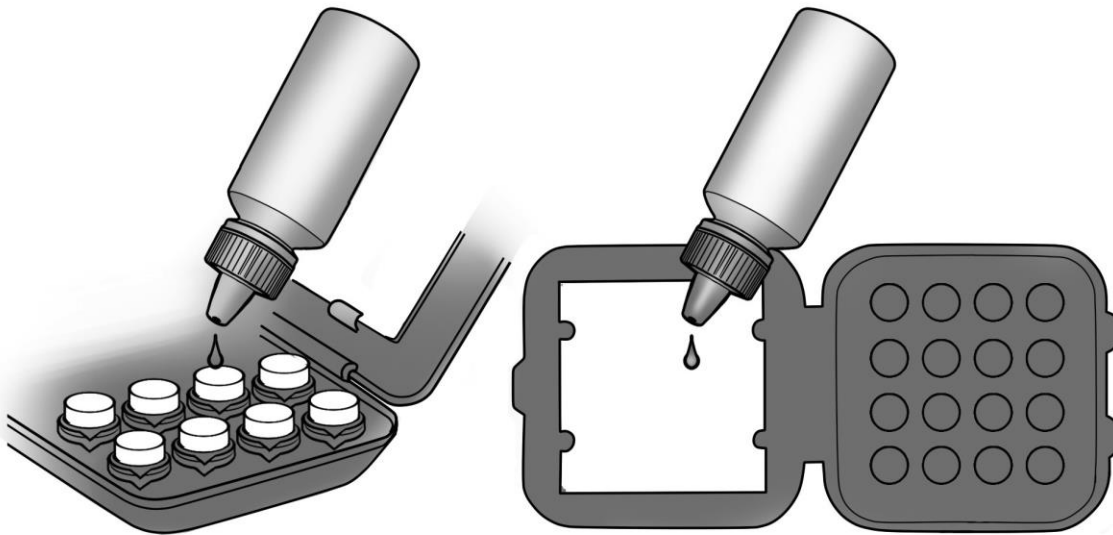
Sensor Assembly

After the wetting process, remove the sensor units with their felt pads from the hydrator pack and insert each one into the black plastic headset arms, turning each one clockwise one-quarter turn until you feel a definite "click". The "click" indicates each sensor is correctly installed in a headset arm. If you have difficulty with this step, apply a little more force until you feel the "click" but be careful not to exert excessive force as damage might occur. Please see the Troubleshooting section if the sensors do not click in place easily.
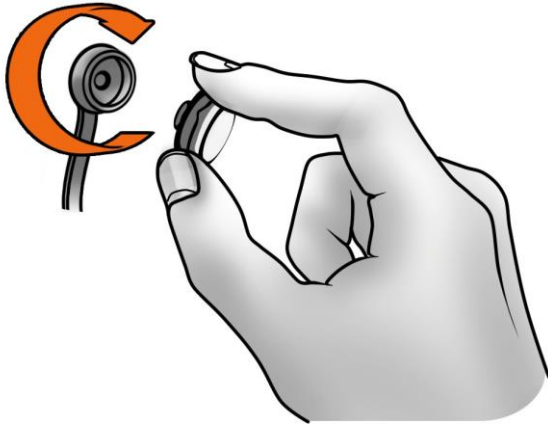
NOTE: When not in use, the sensor units should be removed from the headset arms and stored in the hydrator pack for subsequent use.
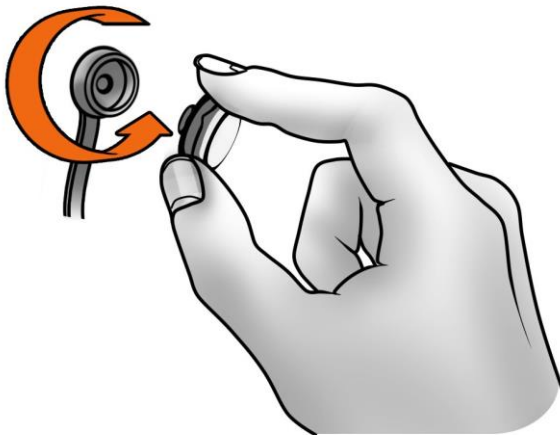
Pairing the Neuroheadset

Insert the supplied USB Transceiver Dongle into one of your computer's USB slots. Use a USB extension cable and position the Transceiver in a prominent location away from your monitor and PC to improve poor reception.
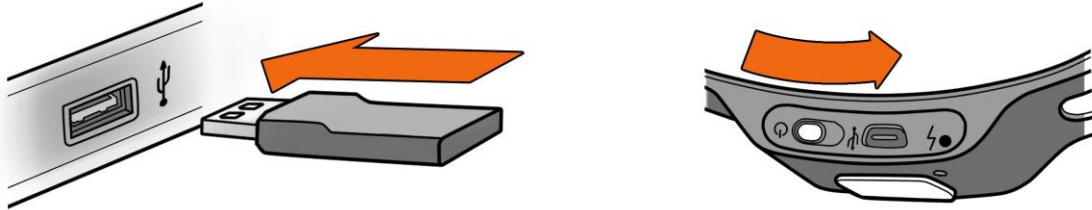
Then turn-on the headset using the switch at the bottom end of the headset, holding it close to the Transceiver.

Headset Placement

You are now ready to put the EPOC headset on your head. Using both hands, slide the headset down from the top of your head. Place the arms approximately as depicted, being careful to place the sensors with the black rubber insert on the bone just behind each ear lobe. Correct placement of the rubber sensor is critical for correct operation.
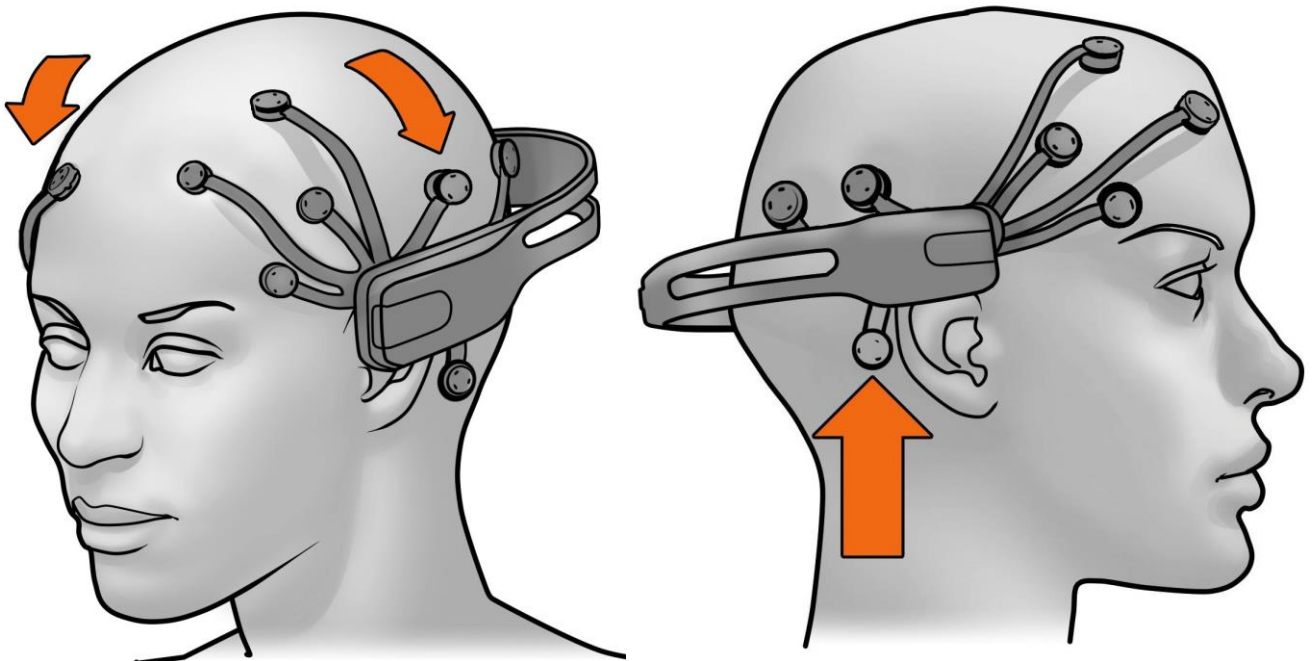
Notice the 2 front sensors should be approximately at the hairline or about the width of 3 fingers above your eyebrows.
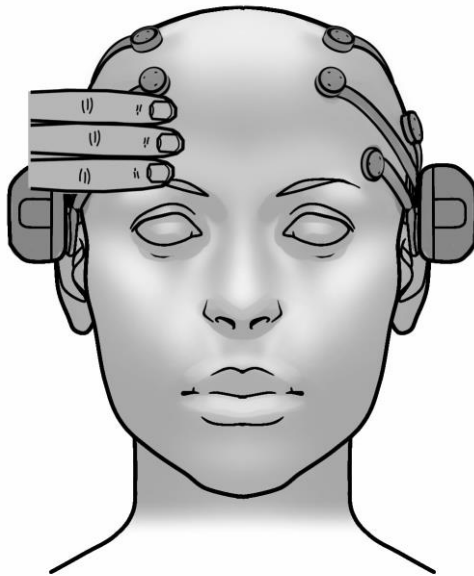
After the headset is in position, press and hold the 2 reference sensors (located just above and behind your ears) for about 5-10 seconds. **Good contact of reference sensors is the key for a good signal.** Check that the lights corresponding to these 2 reference sensors turn from red to green in the EPOC Control Panel Headset Setup screen.
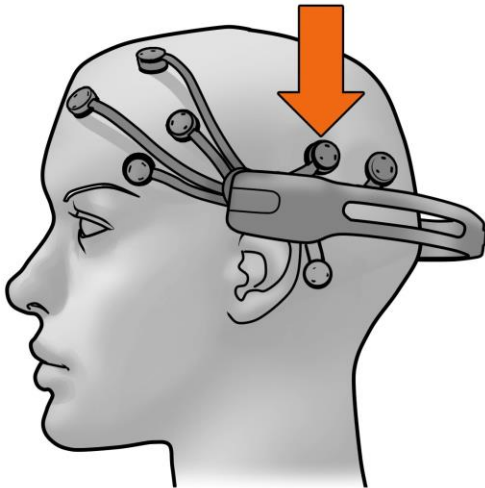
Gently press and hold each remaining sensor against your scalp until all the lights corresponding to those sensors turn to green in the EPOC Control Panel.

If you are unable to get anything except 2 red sensors, add saline to the reference and other sensors, or try the alternate reference locations – swap the reference sensors with the rubber comfort pads located directly behind the ears, making sure the reference sensors contact directly onto the bare skin on the bony bump.

2. Getting Started

2.1 Hardware Components

The Emotiv SDK consists of one or two SDK neuroheadsets, one or two USB wireless receivers, and an installation CD. Emotiv SDKLite does not ship with any hardware components. The neuroheadsets capture users' brainwave (EEG) signals. After being converted to digital form, the brainwaves are processed, and the results are wirelessly transmitted to the USB receivers. A post-processing software component called Emotiv EmoEngine™ runs on the PC and exposes Emotiv detection results to applications via the Emotiv Application Programming Interface (Emotiv API).

Windows PC
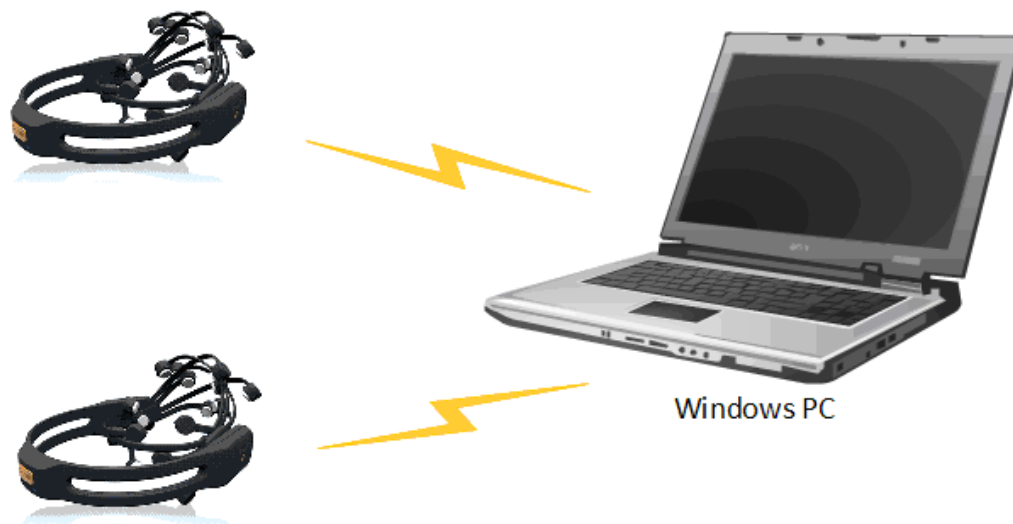
Figure 1 *Emotiv SDK Setup*

For more detailed hardware setup and neuroheadset fitting instructions, please see the "Emotiv SDK Hardware Setup.pdf" file shipped to SDK customers.

2.1.1 Charging the Neuroheadset Battery

The neuroheadset contains a built-in battery which is designed to run for approximately 12 hours when fully charged. To charge the neuroheadset battery, set the power switch to the "off" position, and plug the neuroheadset into the Emotiv battery charger using the mini-USB cable provided with the neuroheadset. Using the battery charger, a fully-drained battery can be recharged to 100% capacity in approximately 6 hours; charging for 30 minutes usually yields about a 10% increase in charge.

Alternatively, you may recharge the neuroheadset by connecting it directly to a USB port on your computer. Please note that this method takes the same amount of time to charge the battery.

The neuroheadset contains a status LED located next to the power switch at the back of the headband. When the power switch is set to the "on" position, the LED will illuminate and appear blue if there is sufficient charge for correct operation. The LED will appear red during battery charging; when the battery is fully-charged, the LED will display green.

2.2 Emotiv SDK Installation

This section guides you through the process of installing the Emotiv Software Development Kit on a Windows PC.

2.2.1 Minimum Hardware and Software requirements

- 2.4 GHz Intel Pentium 4 processor (or equivalent).
- Microsoft Windows XP with Service Pack 2 or Microsoft Windows Vista.
- 1GB RAM.
- 50 MB available disk space.
- One or two unused USB 2.0 ports (depending on the number of neuroheadsets you wish to use simultaneously)

The program displays best when the DPI setting in Windows is 100%

2.2.2 Included Emotiv SDK software

SDKLite developers will download the compressed file Emotiv_SDKLite_v**Error! Reference source not found.**.exe, which contains both the SDKLite software and this User Manual.

SDK developers will download the relevant Edition of the SDK that has all software needed for Emotiv SDK installation. Log in to your account at www.emotiv.com and navigate to My Emotiv ~> Purchases. Your SDK Edition should be available for download. Please also note the installation keys available from the KEY icon next to the DOWNLOAD button.

2.2.3 USB Receiver Installation

(This section is not relevant for SDKLite developers).

Plug the provided Emotiv USB receiver(s) into an unused USB port on your computer. Each receiver should be recognized and installed automatically by your computer as a USB Human Interface Device. The receivers follow the USB Human Interface Device standard so no additional hardware drivers are required to be installed. Please wait for a moment until Windows indicates that the new hardware is installed and ready to use.

2.2.4 Emotiv SDK Installation

This section explains the steps involved in installing the Emotiv SDK software. If an older version of the Emotiv SDK is present in your computer, we recommend that you uninstall it before proceeding.

Step 1 Using Windows Explorer, access the Emotiv SDK installer downloaded from the website.

Step 2 Run the Emotiv Developer Edition SDK v2.0.0.20 Installer file. An **Emotiv Developer Edition SDK v2.0.0.20 Setup** window will appear after a few seconds.
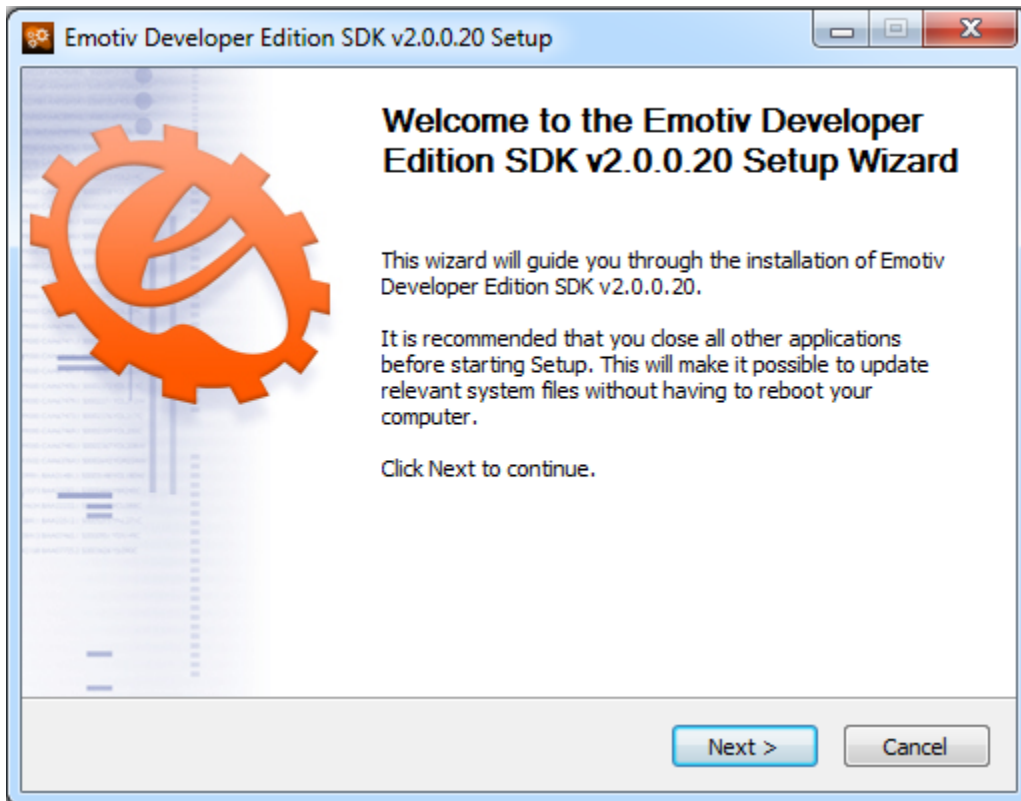
Figure 2 *Emotiv SDK Setup wizard*

Step 3 Click **Next** to start the installation process. You will be asked to enter Order and Serial Number. These numbers are available from the KEY icon next to the DOWNLOAD button at My Emotiv => Purchase. Enter these numbers and click **Next**.

Note: when you enter the correct serial key and order number, a pop-up box will appear indicating the "Serial number is valid". Click OK and proceed to the next step.
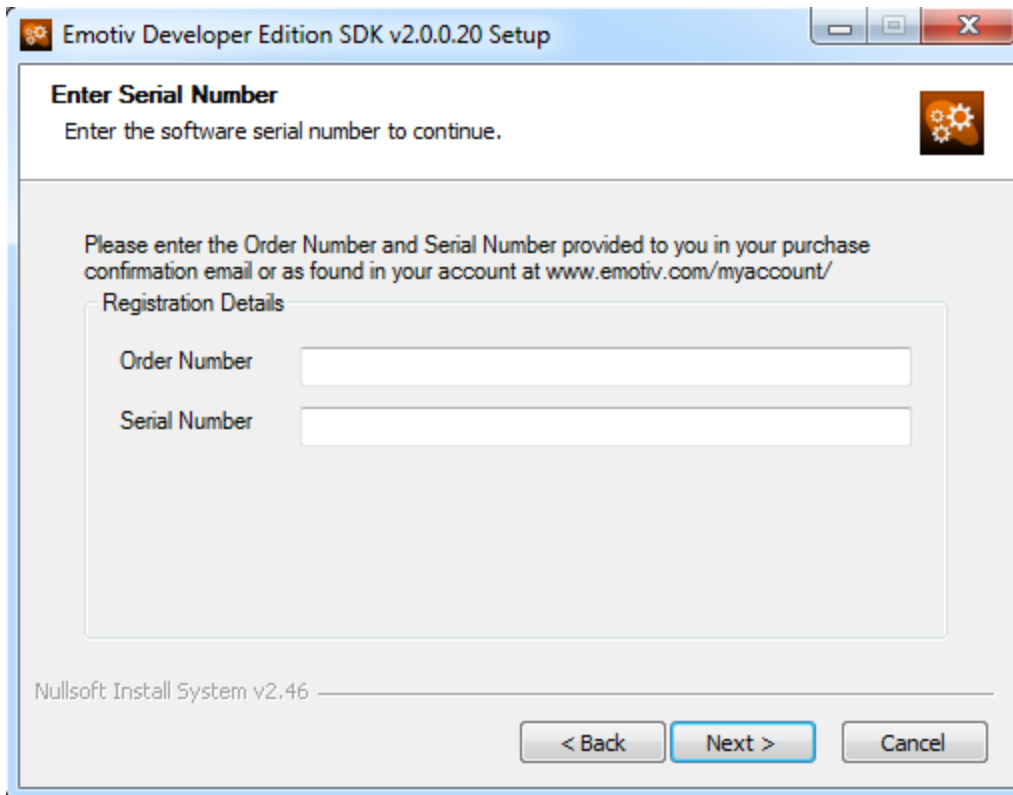
Figure 3   *Enter Order and Serial Number.*

Step 4 If you haven't uninstalled and older version of the Emotiv SDK, you may be asked if you wish to uninstall the older copy before proceeding. Multiple copies of the SDK can coexist on the same machine but you must be careful not to "mix and match" components from multiple installations.

Step 5 After a few seconds, an **Installation Complete** dialog will appear.
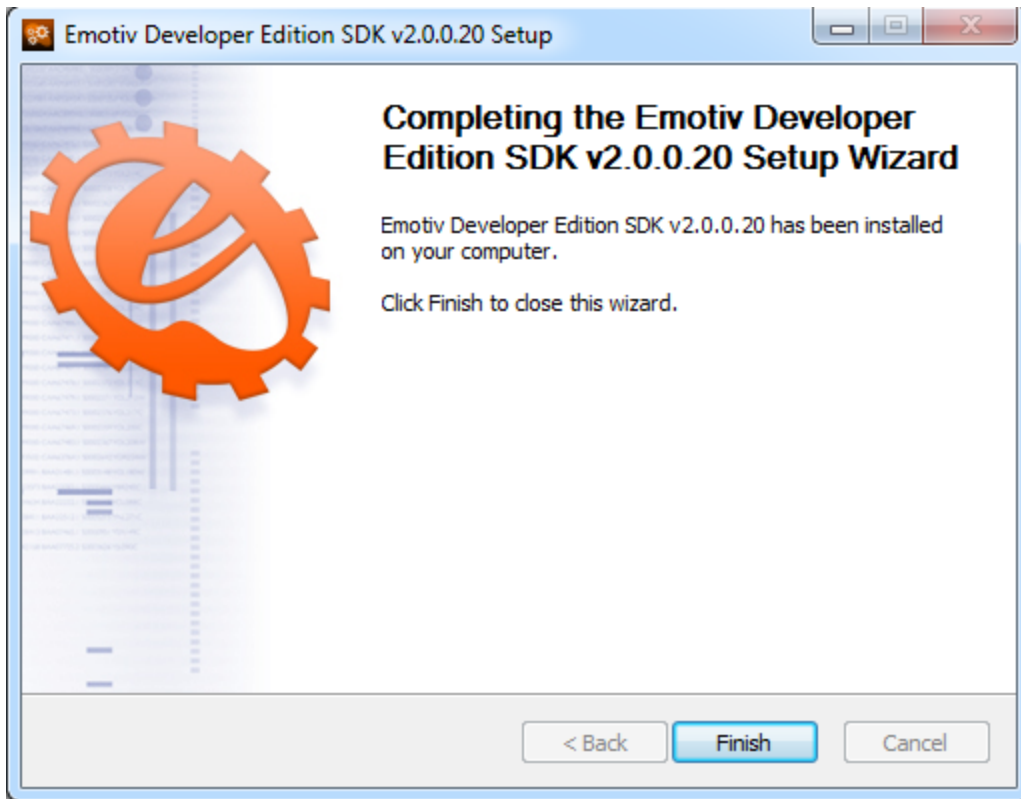
Figure 4 *Installation Complete dialog*

Step 6 Click **Finish** to complete the installation.

2.3 Start Menu Options

Once you have installed the SDK, you will find the following in **Start > Programs**:

- Emotiv Developer Edition SDK v2.0.0.20

  - Control Panel

    - Control Panel   A program to test and tune detection suites available in the Emotiv SDK.

  - Documentation

    - API Reference   Emotiv API programmer's reference guide
    - User Manual   This document

  - Tools

    - EmoComposer   An EmoEngine emulator
    - EmoKey   Tool to map EmoStates to keyboard input to other programs.

  - Uninstall   To uninstall the Emotiv SDK

3. Emotiv Control Panel™

This section explains how to use Emotiv Control Panel to explore the Emotiv detection suites. Refer to Section 5, **Programming with the Emotiv SDK**, for information about using the Control Panel to assist application development with the Emotiv SDK.

Launch Emotiv Control Panel by selecting **Windows Start → Programs → Emotiv Developer Edition SDK v2.0.0.20 → Control Panel → Control Panel**. When the Control Panel is launched for the first time, your firewall software (if installed on your computer) may notify you that the Control Panel is trying to accept connections from the network (port 3008). The notification message may be similar to the dialog shown in Figure 5. For proper operation, you must allow Emotiv Control Panel to use this port by selecting **Unblock** (or a similar option, depending on your firewall software).

Figure 5   *Windows Firewall warning about Emotiv Control Panel – select Unblock*

Emotiv delivers the Emotiv API in the form of a dynamically linked library named edk.dll. Emotiv Control Panel provides a GUI (graphical user interface)that interfaces with Emotiv EmoEngine through the Emotiv API. The Control Panel user interface showcases the EmoEngine's capabilities to decipher brain signals and present them in useful forms using Emotiv's detection suites.

3.1 EMOENGINE STATUS PANE

The top pane of Emotiv Control Panel is known as the EmoEngine Status Pane. This pane displays indicators that provide real-time information about EmoEngine status and neuroheadset sensor contact quality. It also exposes user profile management controls.
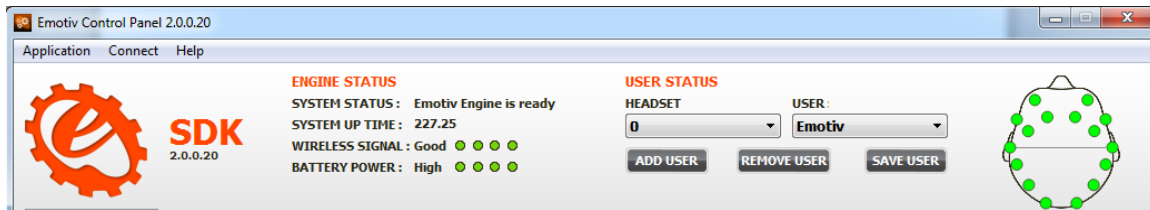
Figure 6   *EmoEngine Status Pane*

3.1.1 ENGINE STATUS

By default, the Control Panel will automatically connect to the EmoEngine when launched. In this mode, it will automatically discover attached USB receivers and Emotiv neuroheadsets. Alternatively, you may choose to connect to EmoComposer, Emotiv's EmoEngine emulator tool, from the **Connect** menu. **SDKLite Developers**: you will need to change this menu setting and connect to EmoComposer. **Please note that EmoComposer should be launched prior to selecting this option in Control Panel.**

There are four status indicators:
- *SYSTEM STATUS*: A summary of the general EmoEngine status.
- *SYSTEM UP TIME*: The timestamp (in seconds) attached to the most recently received EmoState event. Generally, this corresponds to the length of time that the EmoEngine has been running with a neuroheadset connected to the USB receiver
- *WIRELESS SIGNAL*: This displays the quality of the connection between the neuroheadset and the Emotiv wireless USB receiver connected to your machine. If you have not yet connected, the display will show "No Signal". If the wireless signal strength drops too low (displayed as "Bad" or "No Signal") then no detection results will be transmitted and the Control Panel will disable its detection suite UI controls.
- *BATTERY POWER*: Displays an approximation of the remaining charge in the neuroheadset's built-in battery. Not yet supported by all SDK neuroheadsets.

3.1.2 USER STATUS

Use the controls in this section to manage user profiles and assign a specific user (via their profile) to a specific attached neuroheadset. Although the EmoEngine supports up to two simultaneously connected neuroheadsets, Emotiv Control Panel only displays status information and detection results for a single neuroheadset at a time. The **HEADSET** combo box allows you to specify the neuroheadset that has the current "focus." In Figure 6, the **USER STATUS** controls tell us that the Control Panel is currently displaying information for the user with profile "ec", wearing neuroheadset "0." Note: headset numbering begins with 0 and not 1 as you might expect.

Other operations that are supported include adding, saving, removing, and switching between user profiles. Note: Emotiv Control Panel will automatically save user profile data to disk when it exits so it is generally not necessary to use the Save Profile button.

3.1.3 Sensor Contact Quality Display

Accurate detection results depend on good sensor contact and EEG signal quality. This display is a visual representation of the current contact quality of the individual neuroheadset sensors. The display is a smaller copy of the contact quality visualization found on the Control Panel's **HEADSET SETUP** tab. Please see Section **Error! Reference source not found.** for more information about fitting the neuroheadset and achieving good EEG signal quality.

3.2 HEADSET SETUP

The **HEADSET SETUP** panel is displayed by default when starting Emotiv Control Panel. The main function of this panel is to display contact quality feedback for the neuroheadset's EEG sensors and provide guidance to the user in fitting the neuroheadset correctly. It is extremely important for the user to achieve the best possible contact quality before proceeding to the other Emotiv Control Panel tabs. Poor contact quality will result in poor Emotiv detection results.
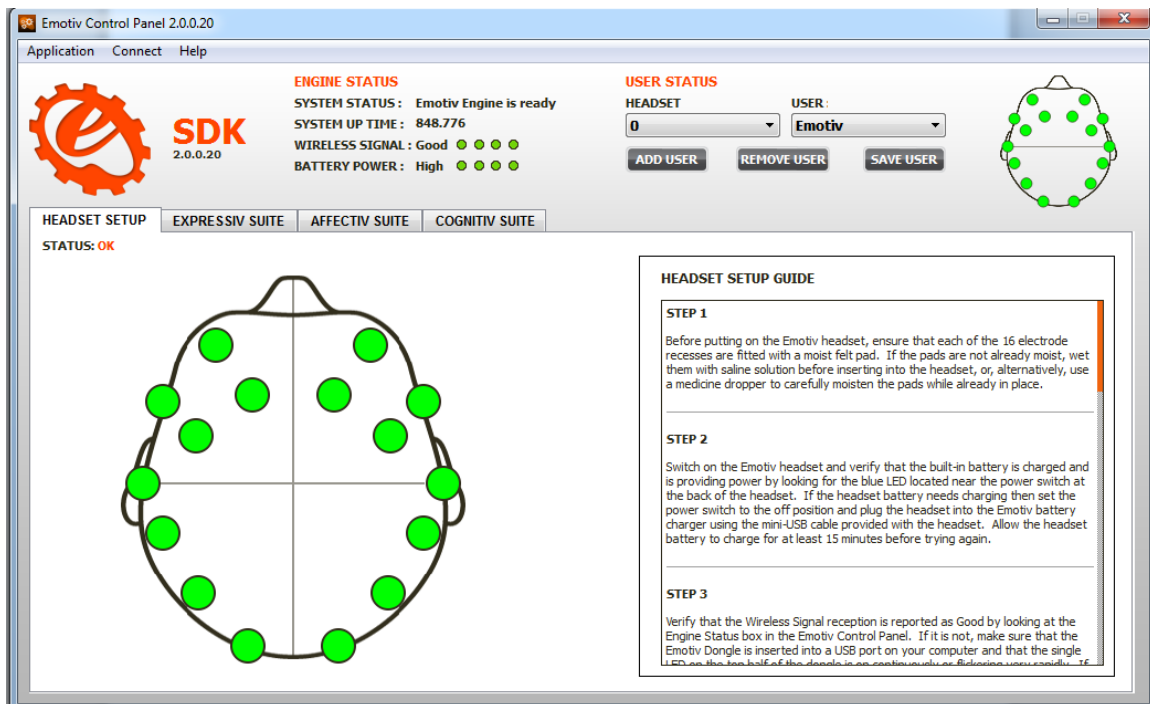
Figure 7   HEADSET SETUP *Panel*

The image on the left is a representation of the sensor locations when looking down from above onto the user's head. Each circle represents one sensor and its approximate location when wearing the SDK headset. The color of the sensor circle is a representation of the contact quality. To achieve the best possible contact quality, all of the sensors should show as green. Other sensor colors indicate:

Black   No signal

Red   Very poor signal

Orange   Poor signal

Yellow   Fair signal

Green   Good signal

The setup procedure used to achieve good contact quality is outlined below. Only after the neuroheadset sensor contact quality has been verified, should you move on to other Emotiv Control Panel tabs.

3.2.1 Achieving Good Signal Quality

Note to SDKLite Developers: This section is not required, but it may be useful to understand how contact quality information may need to be conveyed to the user for standalone, Emotiv-enabled applications.

Step 1 Before putting on the SDK neuroheadset, ensure that each of the 16 electrode recesses are fitted with a moist felt pad. If the pads are not already moist, wet them with saline solution before inserting into the headset, or, alternatively, use a medicine dropper to carefully moisten the pads while already in place.

Step 2 Switch on the neuroheadset, and verify that the built-in battery is charged and is providing power by looking for the blue LED located near the power switch at the back of the headset. If the headset battery needs charging, then set the power switch to the off position, and plug the headset into the Emotiv battery charger using the mini-USB cable provided with the neuroheadset. Allow the neuroheadset battery to charge for at least 15 minutes before trying again.

Step 3 Verify that the *WIRELESS SIGNAL* reception is reported as "Good" by looking at the *ENGINE STATUS* area in the EmoEngine Status Pane (described in Section **Error! Reference source not found.**). If the *WIRELESS SIGNAL* status is reported as "Bad" or "No Signal", then make sure that the Emotiv Wireless USB Receiver is inserted into a USB port on your computer and that the single LED on the top half of the receiver is on continuously or flickering very rapidly. If the LED is blinking slowly or is not illuminated, then remove the receiver from the computer, reinsert it, and try again. Remove any metallic or dense physical obstructions located near the receiver or the neuroheadset, and move away from any powerful sources of electromagnetic interference, such as microwave ovens, large motors, or high-powered radio transmitters.

Step 4 Put on the neuroheadset by gently pulling apart the headband and lowering the sensor arms onto your head from the top down, near the rear of the skull. Next, slide the headset forward until the sensors closest to the headset pivot points are located directly above your ears and as close to your hairline as possible. Adjust the fit so that the rectangular compartments at the front ends of the headband sit comfortably just above and behind your ears. Tilt the headset so that the two lowest, frontmost sensors are symmetric on your forehead and positioned 2 to 2.5 inches above your eyebrows. Finally, check that all sensors are touching your head and, if not, then fine tune the headset fit by gently sliding the headset in small increments until an ideal fit has been achieved.

Step 5 Starting with the two sensors just above and behind your ears (these are reference sensors for which good contact with your scalp is essential), adjust the sensors so they make proper contact with your scalp (i.e. show green on the contact quality display). If the indicators are:

Black: Check that the sensor has a felt pad fitted. Check that the felt pad is pressing firmly against your scalp. Then try re-moistening the felt pad. If problems persist, this may indicate a problem with the neuroheadset.

Yellow, **Orange**, or **Red**: The sensor has not established a good conductive path with your scalp. Check that the felt pad is making comfortable, yet firm, contact with your scalp. Try shifting the headset slightly back and forth on your head, or press gently on the troublesome sensor to improve contact. If the contact is adequate, ensure that the felt pad is moist. If the sensor's indicator color becomes lighter, the signal quality is

improving. If the sensor's indicator color is getting darker the signal quality is deteriorating. If problems still persist, try parting the hair in the vicinity of the electrode so the felt pad touches your scalp.

Step 6 Repeat for each of the remaining sensors until all of the sensors have adequate contact quality (i.e. are predominantly showing green).

If at any time the reference sensors (located just above and behind your ears) no longer have a good connection (i.e. are not showing green), immediately restore these sensors to green before proceeding further.

3.3 EXPRESSIV™ SUITE
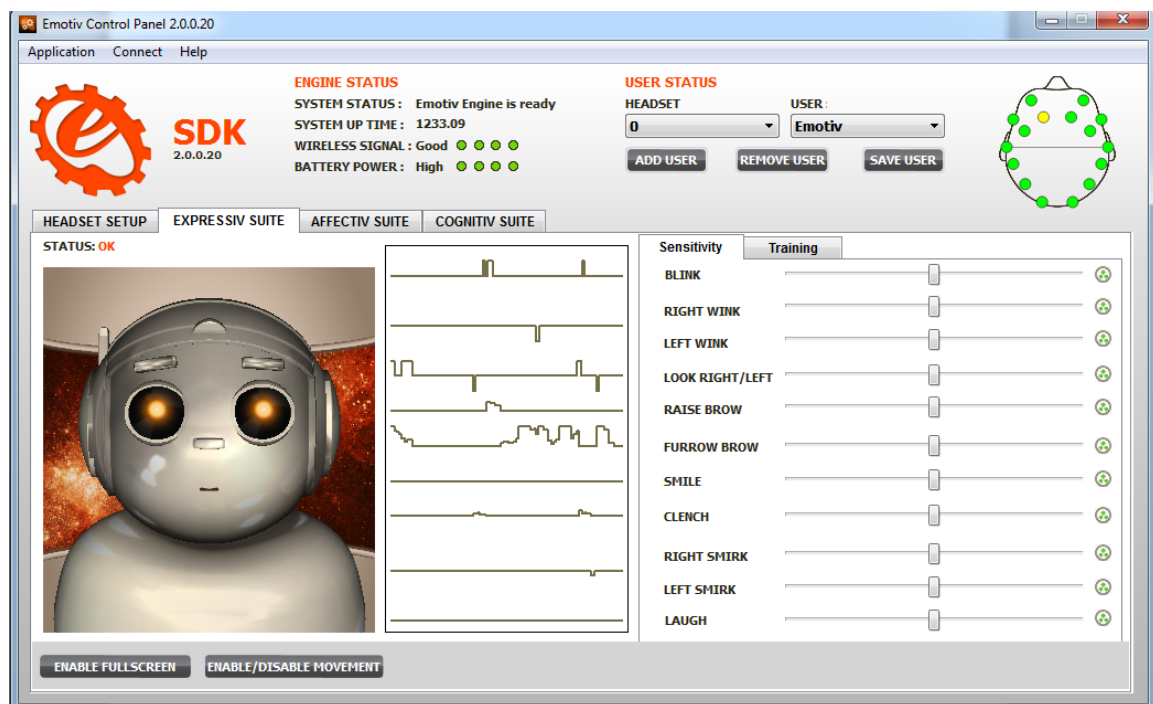
Figure 8   *EXPRESSIV SUITE Sensitivity Adjustment Panel*

3.3.1 Understanding the EXPRESSIV SUITE Panel Display

On the left-hand side of the **EXPRESSIV SUITE** panel is a simple avatar. The avatar will mimic your facial expressions, in camera view (i.e. not mirrored).

In the center of the panel is a series of graphs, indicating the various expression detection event signals. These graphs show a short history of the detections listed. The graphs should be interpreted as follows:
- *BLINK*: low level indicates a non-blink state, while a high level indicates a blink.
- *RIGHT WINK / LEFT WINK*: these two detections share a common graph line. A center level indicates no wink, low level indicates a left wink and high level indicates a right wink.
- *LOOK RIGHT / LEFT*: these two detections share a common graph line and a single sensitivity slider control. A center level indicates eyes looking straight ahead, while a low level indicates eyes looking left, and a high level indicates eyes looking right.

- **RAISE BROW:** low level indicates no expression has been detected, high level indicates a maximum level of expression detected. The graph level will increase or decrease depending on the level of expression detected.
- **FURROW BROW**: low level indicates no expression has been detected, high level indicates a maximum level of expression detected. The graph level will increase or decrease depending on the level of expression detected.
- **SMILE**: low level indicates no expression has been detected, high level indicates a maximum level of expression detected. The graph level will increase or decrease depending on the level of expression detected.
- **CLENCH**: low level indicates no expression has been detected, high level indicates a maximum level of expression detected. The graph level will increase or decrease depending on the level of expression detected.
- **RIGHT SMIRK / LEFT SMIRK**: these two detections share a common graph line. A center level indicates no smirk, low level indicates a left smirk and high level indicates a right smirk.
- **LAUGH**: low level indicates no expression has been detected, high level indicates a maximum level of expression detected. The graph level will increase or decrease depending on the level of expression detected.

On the right-hand side of the panel are two additional panels: **Sensitivity** and **Training**. These panels are explained further in the following two sections.

3.3.2 Sensitivity Adjustment Panel

The Control Panel offers sensitivity adjustments for the Expressiv Suite detections. This is controlled through sliders to the right of corresponding graph.

For each facial expression, check the performance of the detection. If you feel that the Expressiv detection is not responding readily to a particular expression, then increase the sensitivity for that expression. If you feel that it is too easy to trigger a particular expression, or you are seeing "false positive" expressions, then decrease the sensitivity for that expression. Sensitivity can be increased or decreased by moving the sensitivity slider to the right or left, respectively.

Expressiv supports two types of "signatures" that are used to classify input from the neuroheadset as indicating a particular facial expression. The icon to the right of the sliders is an indicator of whether the *Universal Signature* or *Trained Signature* is being used. A circle with three dots is shown when the Universal Signature is active, while a circle with one dot inside indicates that a Trained Signature is active. An empty circle indicates that a Trained Signature has been selected, but that no training data has been collected for this action, and so it is currently disabled. The default signature, the Universal Signature, is designed to work well for a large population of users for the supported facial expressions.

If the application or user requires more accuracy or customization, then you may decide to use a Trained Signature as described below.

3.3.3 Training Panel
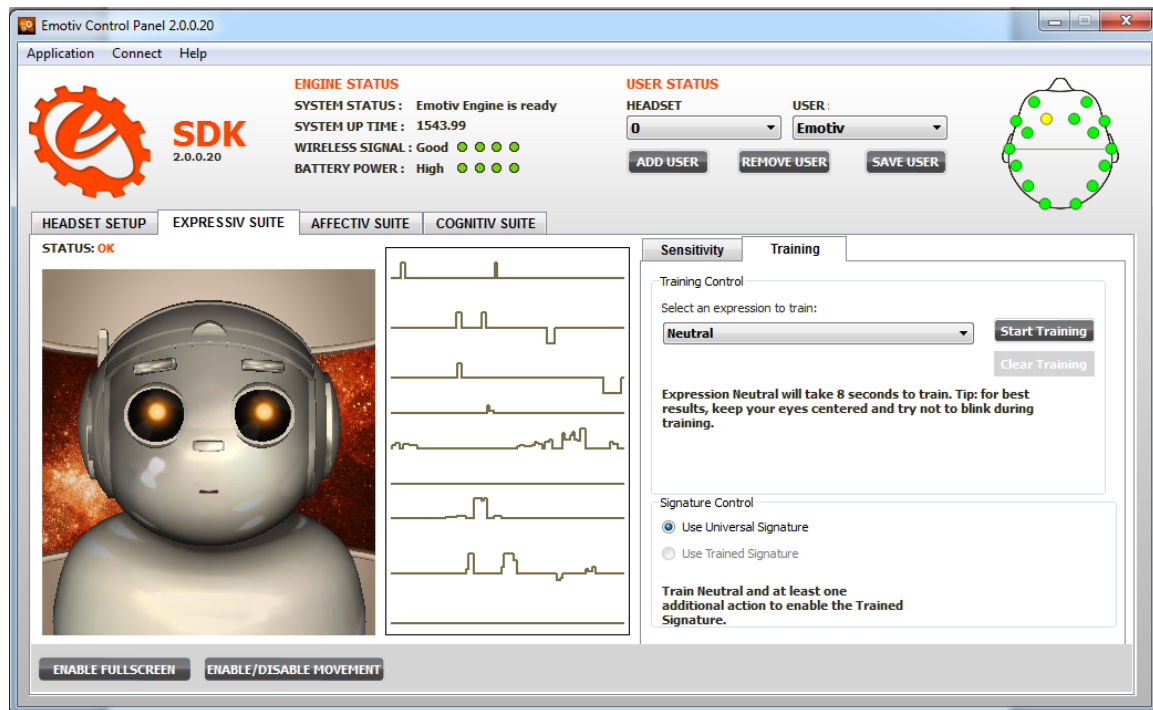
Figure 9   EXPRESSIV SUITE Training Panel

In this mode, Expressiv requires the user to train the system by performing the desired action before it can be detected. As the user supplies more training data, the accuracy of the Expressiv detection typically improves. If you elect to use a Trained Signature, the system will only detect actions for which the user has supplied training data. The user must provide training data for a neutral expression and at least one other supported expression before the Trained Signature can be activated with the *Use Trained Signature* checkbox. Important note: not all Expressiv expressions can be trained. In particular, eye and eyelid-related expressions (i.e. "blink", "wink", "look left", and "look right") can not be trained and always rely on the Universal Signature.

3.4 AFFECTIV™ SUITE
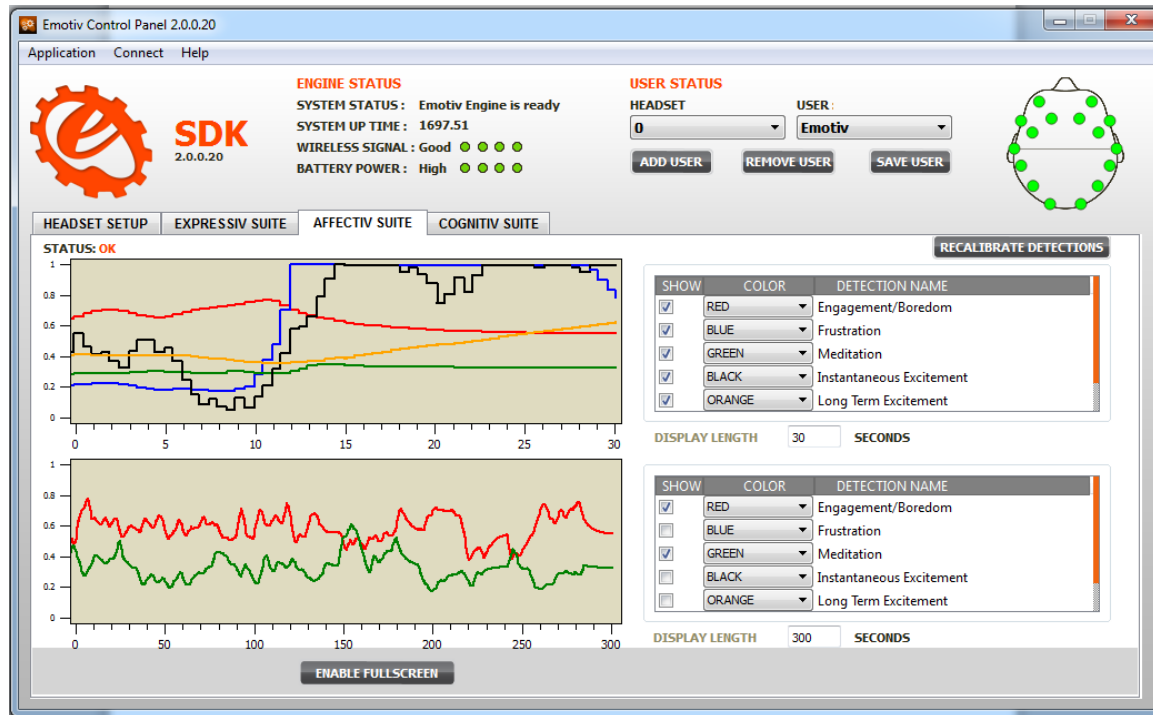
Figure 10   *AFFECTIV SUITE Panel*

3.4.1 AFFECTIV SUITE Introduction

The Affectiv Suite reports real time changes in the subjective emotions experienced by the user. Emotiv currently offers three distinct Affectiv detections: Engagement, Instantaneous Excitement, and Long-Term Excitement. See Section 3.4.3 below for a description of these detections. The Affectiv detections look for brainwave characteristics that are universal in nature and don't require an explicit training or signature-building step on the part of the user. However, individual data is collected for each user and is saved in the user's profile while the Affectiv Suite runs. This data is used to rescale the Affectiv Suite results and improve the detection accuracy over time. For this reason, it is very important that a new user profile is selected when a new user puts on the neuroheadset.

3.4.2 Understanding the Affectiv Panel Display

The **AFFECTIV SUITE** panel contains two graphs which can be customized to display different combinations of detections and time scales. By default, the top chart is configured to plot 30 seconds of data for the **Engagement** and **Instantaneous Excitement** detections. The bottom chart defaults to display 5 minutes worth of data for the **Long-Term Excitement** detection. The values that are plotted on the graphs are the output scores returned by the Affectiv detections.

The controls to the right of the charts can be used to select the detection output to be plotted and to customize the color of each plot. The **DISPLAY LENGTH** edit box allows you to customize the time scale for the associated chart.

3.4.3 AFFECTIV SUITE Detection Details

Instantaneous Excitement is experienced as an awareness or feeling of physiological arousal with a positive value. Excitement is characterized by activation in the sympathetic nervous system which results in a range of physiological responses including pupil dilation, eye widening, sweat gland stimulation, heart rate and muscle tension increases, blood diversion, and digestive inhibition.

Related emotions: titillation, nervousness, agitation

Scoring behavior: In general, the greater the increase in physiological arousal the greater the output score for the detection. The Instantaneous Excitement detection is tuned to provide output scores that more accurately reflect short-term changes in excitement over time periods as short as several seconds.

Long-Term Excitement is experienced and defined in the same way as Instantaneous Excitement, but the detection is designed and tuned to be more accurate when measuring changes in excitement over longer time periods, typically measured in minutes.

Engagement is experienced as alertness and the conscious direction of attention towards task-relevant stimuli. It is characterized by increased physiological arousal and beta waves (a well-known type of EEG waveform) along with attenuated alpha waves (another type of EEG waveform). The opposite pole of this detection is referred to as "Boredom" in Emotiv Control Panel and the Emotiv API; however, please note that this does not always correspond to a subjective emotional experience that all users describe as boredom.

Related emotions: alertness, vigilance, concentration, stimulation, interest

Scoring behavior: The greater the attention, focus and cognitive workload, the greater the output score reported by the detection. Examples of engaging video game events that result in a peak in the detection are difficult tasks requiring concentration, discovering something new, and entering a new area. Deaths in a game often result in bell-shaped transient responses. Shooting or sniping targets also produce similar transient responses. Writing something on paper or typing typically increase the engagement score, while closing the eyes almost always rapidly decreases the score.

3.5 COGNITIV™ SUITE

3.5.1 COGNITIV SUITE Introduction

The Cognitiv detection suite evaluates a user's real time brainwave activity to discern the user's conscious intent to perform distinct physical actions on a real or virtual object. The detection is designed to work with up to 13 different actions: 6 directional movements (push, pull, left, right, up and down) and 6 rotations (clockwise, counter-clockwise, left, right, forward and backward) plus one additional action that exists only in the realm of the user's imagination: disappear.

Cognitiv allows the user to choose up to 4 actions that can be recognized at any given time. The detection reports a single action or neutral (i.e. no action) at a time, along with an action power which represents the detection's certainty that the user has entered the cognitive state associated with that action.

Increasing the number of concurrent actions increases the difficulty in maintaining conscious control over the Cognitiv detection results. Almost all new users readily gain control over a single action quite quickly. Learning to control multiple actions typically requires practice and becomes progressively harder as additional actions are added. Although Emotiv Control Panel allows a user to select up to 4 actions at a time, it is important that each user masters the use of the Cognitiv detection one action at a time, only increasing the number of concurrent actions after he has first gained confidence and accuracy with a lower number of actions.
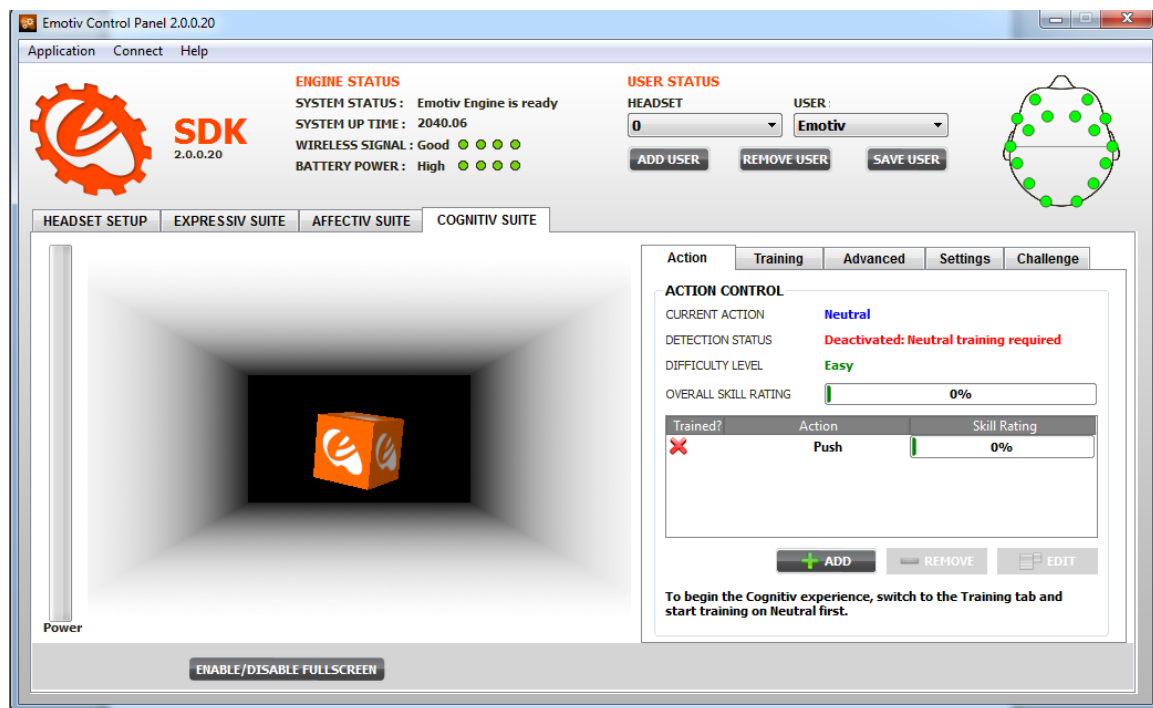
Figure 11 *COGNITIV SUITE Panel*

3.5.2 Understanding the Cognitiv Panel Display

The **COGNITIV SUITE** panel uses a virtual 3D cube to display an animated representation of the Cognitiv detection output. This 3D cube is also used to assist the user in visualizing the intended action during the training process. The **Power** gauge to the left of the 3D display is an indicator of the "action power", or relative certainty that the user is consciously visualizing the current action.

The default tab on the Cognitiv panel is the **Action** tab. This tab displays information about the current state of the Cognitiv detection and allows the user to define the current set of actions.

In order to enable the Cognitiv detection, each chosen action, plus the Neutral action, must first be trained. For more information about the training process please refer to Section 3.5.3 below. Alongside each currently selected action is another gauge displaying a **SKILL RATING**. This skill rating is calculated during the training process and provides a measure of how consistently the user can mentally perform the intended action. It is necessary to train the same action at least two times before the action skill is updated. The **OVERALL SKILL RATING** is simply the average of all the individual action

skills and can be used as a general measure of the user's skill with the selected set of actions and existing training data.

A green checkmark is used to indicate that the corresponding action has been trained; a red X indicates a lack of training data. Remember, in order for the Cognitiv detection to be activated, all actions, plus Neutral (the user's background mental state) must be trained.

Use the **ADD**, **REMOVE**, and **EDIT** push buttons to modify the number and type of enabled actions.

3.5.3 Cognitiv Training

The Cognitiv training process enables the EmoEngine to analyze your brainwaves and develop a personalized signature which corresponds to each particular action, as well as the background state, or "neutral". As the EmoEngine learns and refines the signatures for each of the actions, as well as neutral, detections become more precise and easier to perform.
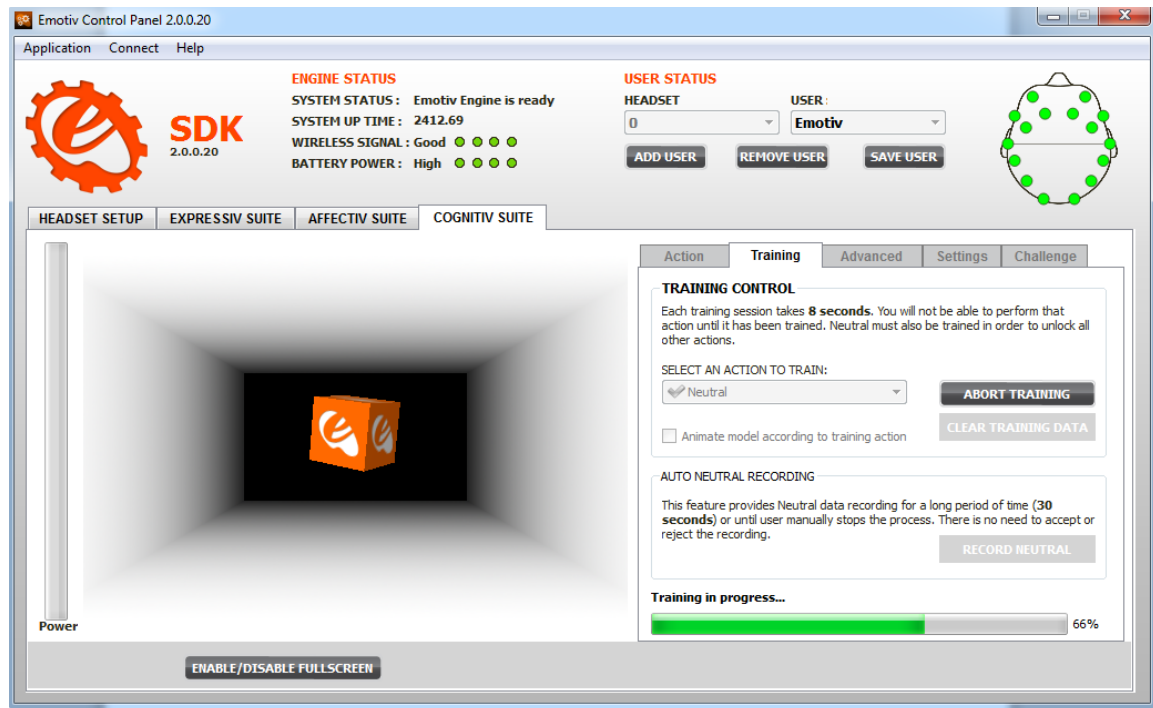
Figure 12 *Cognitiv training in action*

The **Training** tab contains the user interface controls that support the Cognitiv training process. The training process consists of three steps:

First, select an action from the dropdown list. Actions that have already been trained are paired with a green checkmark; actions with no training data are paired with a red X. Only actions that have been selected on the **Action** tab are available for training. The default action is one without training data; otherwise the default action is "Neutral."

Next, when you are ready to begin imagining or visualizing the action you wish to train, press the **START TRAINING** button. During the training process it is very important to maintain your mental focus for the duration of the training period (currently 8 seconds).

Physical gestures, such as pushing an imaginary object with one hand, may be used to heighten your focus on the intended action, but are not required. You should also refrain from making substantial head movements or dramatic facial expressions during the training period, as these actions can interfere with the recorded EEG signal.

Initially, the cube on screen will not move, as the system has not yet acquired the training data necessary to construct a personalized signature for the current set of actions. After Neutral and each enabled action have been trained at least once, the Cognitiv detection is activated and the cube will respond to the Cognitiv detection, and your mental control, in real time.

Some users will find it easier to maintain the necessary mental focus if the cube is automatically animated to perform the intended action as a visualization aid during training. If you think you will benefit from this, then you may select the *Animate model according to training action* checkbox. Otherwise, the cube will remain stationary or, if you have already supplied training data and the detection is active, will be animated by the current detection results for the action being trained, while you supply new training data.

Finally, you are prompted to accept or reject the training recording. Ideal Cognitiv detection performance is typically achieved by supplying consistent training data (i.e. a consistent mental visualization on the part of the user) across several training sessions for each enabled action. The ability to reject the last training recording allows you to decide whether you were able to remain mentally focused on the appropriate action during the last training session. Alternatively, you may press the *ABORT TRAINING* button to abort the training recording if you are interrupted, become distracted, or notice problems with the neuroheadset contact quality indicators during the recording.

A training session is automatically discarded if the wireless signal strength or EEG signal quality is poor for a significant portion of the training period. A notification will be displayed to the user if this has occurred.
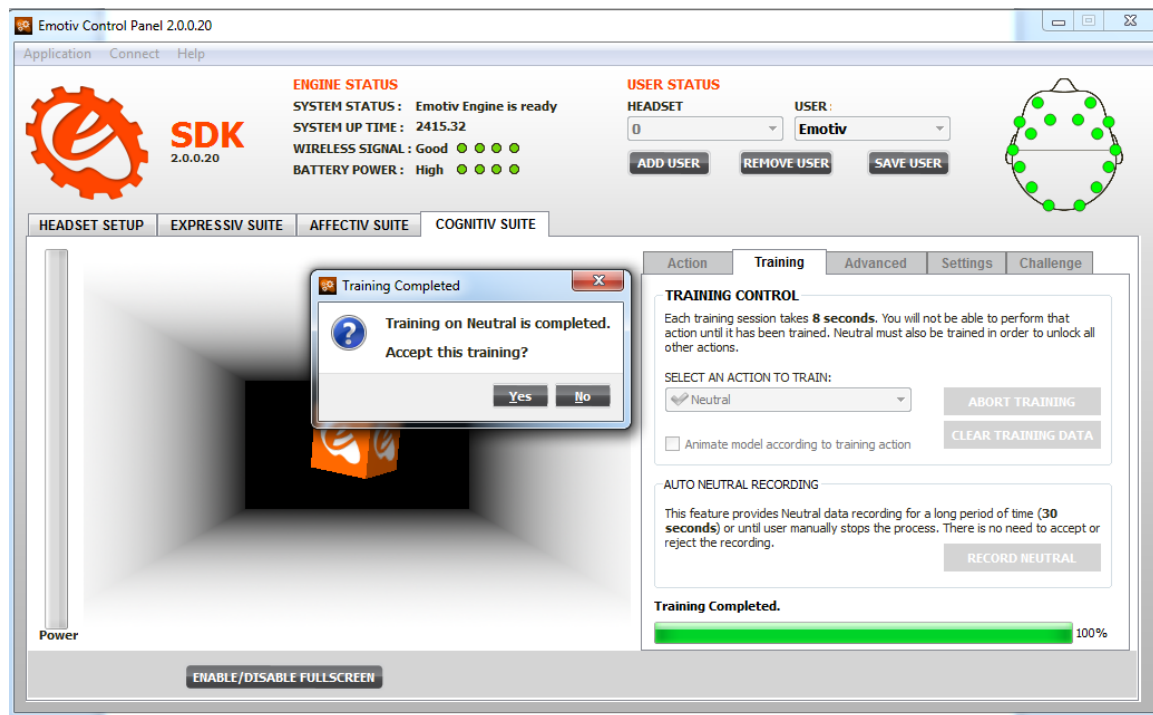
Figure 13 *Accepting or Rejecting a Cognitiv Training Session*

3.5.4 Training Neutral

The Neutral "action" refers to the user's passive mental state; one that isn't associated with any of the selected Cognitiv actions. While training Neutral, you should enter a mental state that doesn't involve the other Cognitiv actions. Typically this means engaging in passive mental activities such as reading or just relaxing. However, to minimize "false-positive" Cognitiv action results (i.e. incorrect reporting of unintended actions), it may also be helpful to emulate other mental states and facial expressions that are likely to be encountered in the application context and environment in which you'll be using Cognitiv. For many users, providing more Neutral training data will result in better overall Cognitiv performance. In order to facilitate the acquisition of this data, and because most users have an easier time maintaining a relaxed mental state for longer periods of time, the Training tab provides a second method for training Neutral.

The **RECORD NEUTRAL** button allows you to record up to 30 seconds of Neutral training data. The recording will automatically finish, and the Cognitiv signature will be rebuilt, after 30 seconds, or you may press the **STOP TRAINING** button at any time you feel sufficient data has been collected. Note: at least 6 seconds of recorded data will be required to update the signature with data collected using the **RECORD NEUTRAL** button.

3.5.5 CLEAR TRAINING DATA Button

Occasionally, you may find that a particular trained action doesn't work as well as it once did. This may indicate that the training data used to construct your personalized Cognitiv signature was "contaminated" by a more recent, inconsistent training session or that some characteristics of your brainwaves have changed over time. It may also happen that you wish to change the mental imagery or technique that you associate with a particular action. In either situation, you can use the **CLEAR TRAINING DATA** button to delete the training data for the selected action. Keep in mind that doing so will disable the Cognitiv detection until new training data has been recorded for this action.

3.5.6 Advanced Cognitiv Options

The **Advanced** tab exposes settings and controls that allow you to customize the behavior of the Cognitiv detection. By default, the Cognitiv detection is pre-configured in a manner that produces the best results for the largest population of users and environments. It is strongly recommended that you only change these settings with the guidance of Emotiv personnel who are better acquainted with the internal details of the Emotiv EmoEngine.

3.5.7 Cognitiv Tips

Mental dexterity with the Cognitiv Suite is a skill that will improve over time. As you learn to train distinct, reproducible mental states for each action, the detection becomes increasingly precise. Most users typically achieve their best results after training each action several times. Overtraining can sometimes produce a decrease in accuracy – although this may also indicate a lack of consistency and mental fatigue. Practice and experience will help determine the ideal amount of training required for each individual user.

If it becomes hard for you to return to neutral (i.e. to stop the cube from moving) you should try refreshing your mental state by momentarily shifting your focus away from the screen and relaxing. It is easy to become immersed in the experience and to have the Cognitiv actions at the "front of your mind" while trying to be neutral.

Successful training relies on consistency and focus. For best results, you must perform the intended action continuously for the full training period. It is common for novice users to become distracted at some point during the training period and then mentally restart an action, but this practice will result in poorer results than training with a mental focus that spans the entire training period.

A short latency, of up to two seconds, in the initiation and cessation of the cube's animated action on screen is typical.

3.5.8 Cognitiv Settings

SETTINGS is a new feature of Emotiv SDK Control Panel. You can use your EPOC headset to customize and control imported 3D Objects.

First you have to Import your model, edit the scale to suit, and arrange the background to create custom scenery. Then you can use the EPOC Cognitiv Suite to control any 3D Object in the Control Panel window.

Cube:

The Cube tab allows you to set the Background, Surface and Frame for the usual Cube. There are two types display for cube: Image and 3D cube.

3D settings

There are two included 3D Models which may be selected
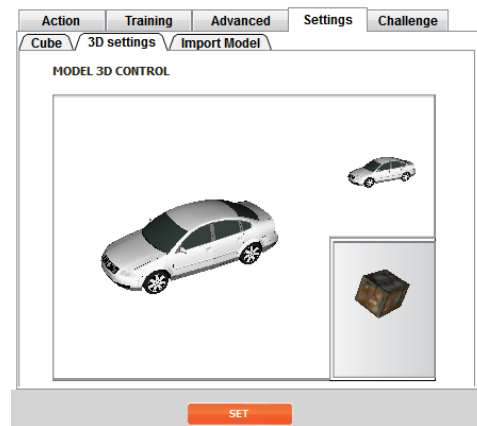
Figure 14   *Set 3D Model*

Select the 3D Object from the Option tab and use the EPOC headset to control them in Cognitiv Suite.
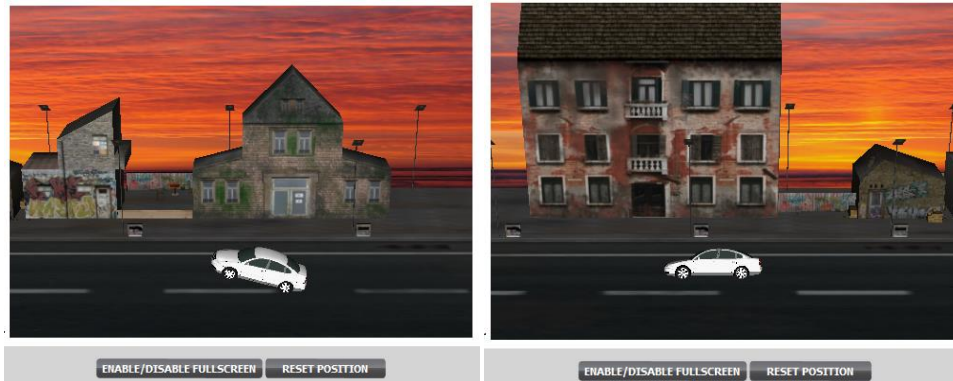
Figure 15 *Demo actions for model*

You can combine multiple actions to control the model (RotateClockwise and Left/Right) or (RotateReverse vs Left/Right).

IMPORT MODEL:

IMPORT MODEL allows the user to set up the scene with your own3D models (.3ds format only) which can be manipulated using your mental commands.

Step 1: Import model
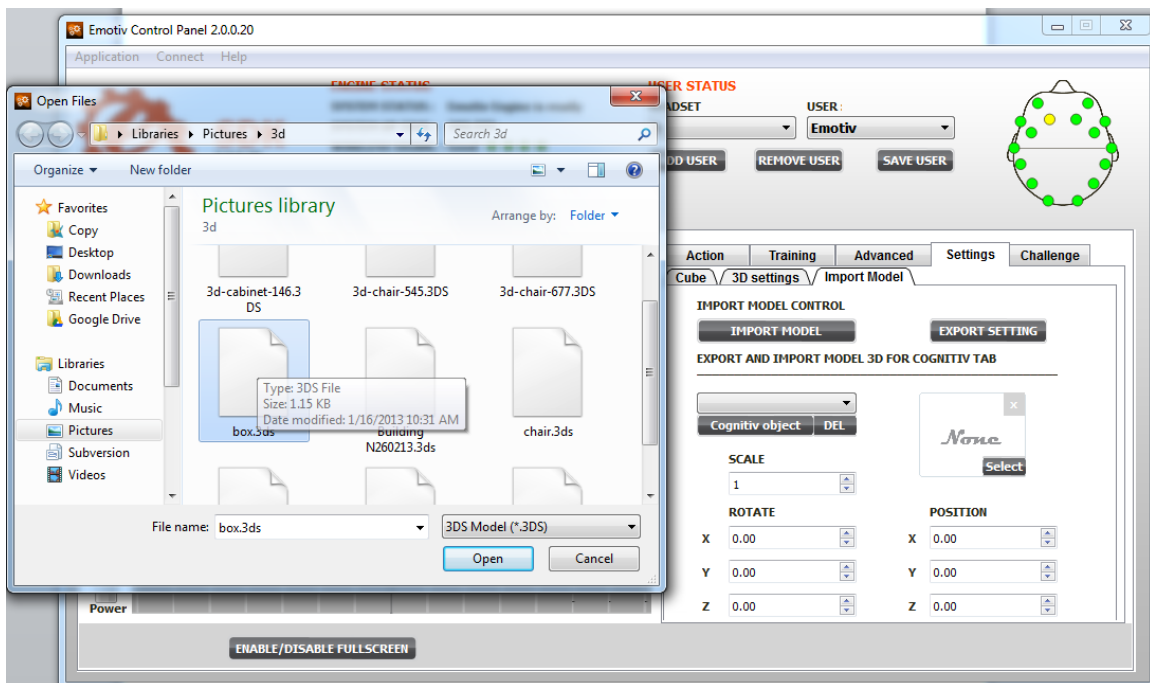
Figure 16 *IMPORT MODEL*

Step 2: Place the models in your desired position then add Texture for Model
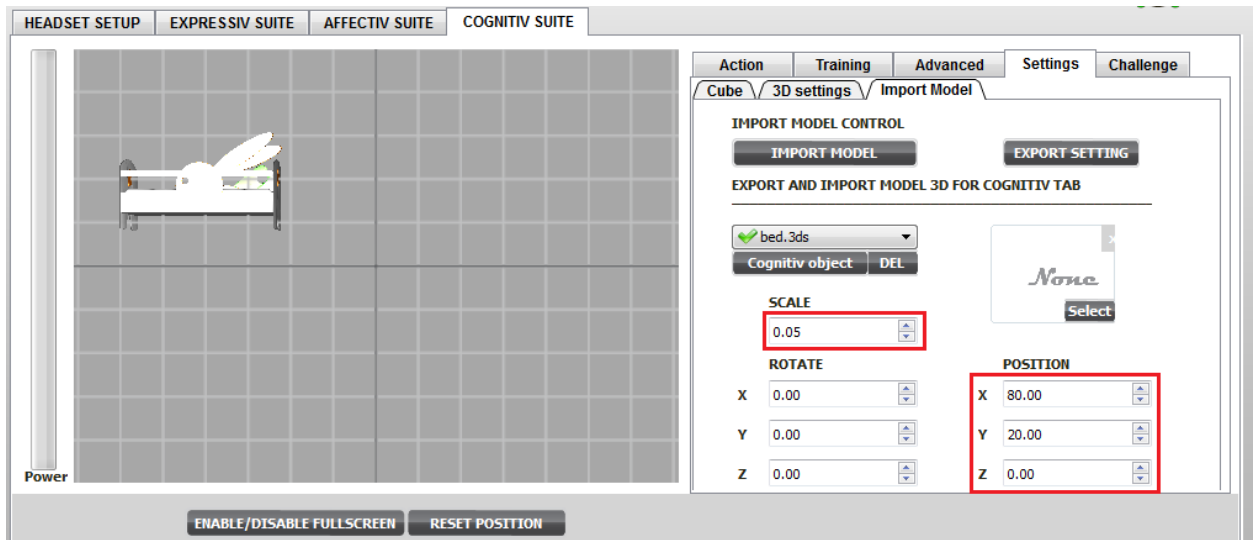
Figure 17 *EDIT SCALE, POSITION for model*
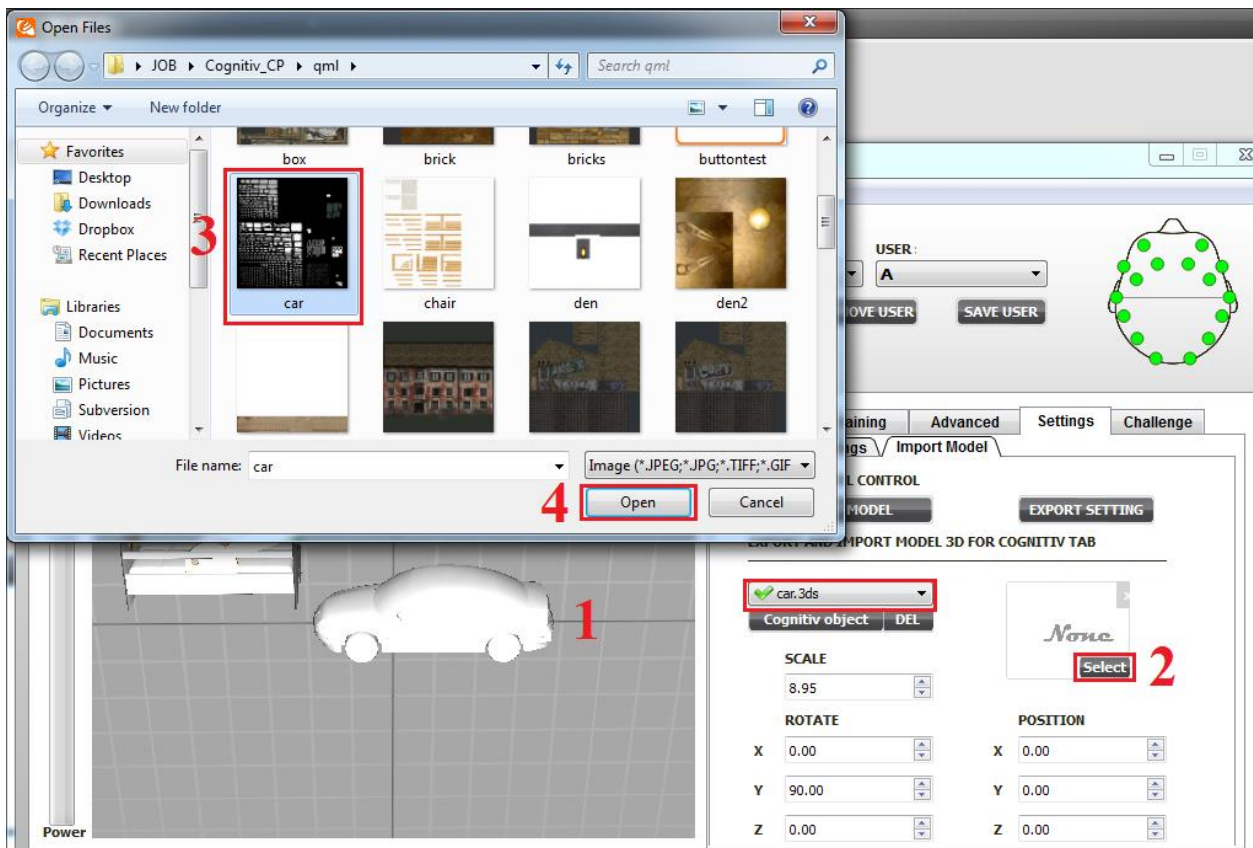
Figure 18 *Add texture for model*

Step 3: Repeat step 1 and step 2 until your scene is completed
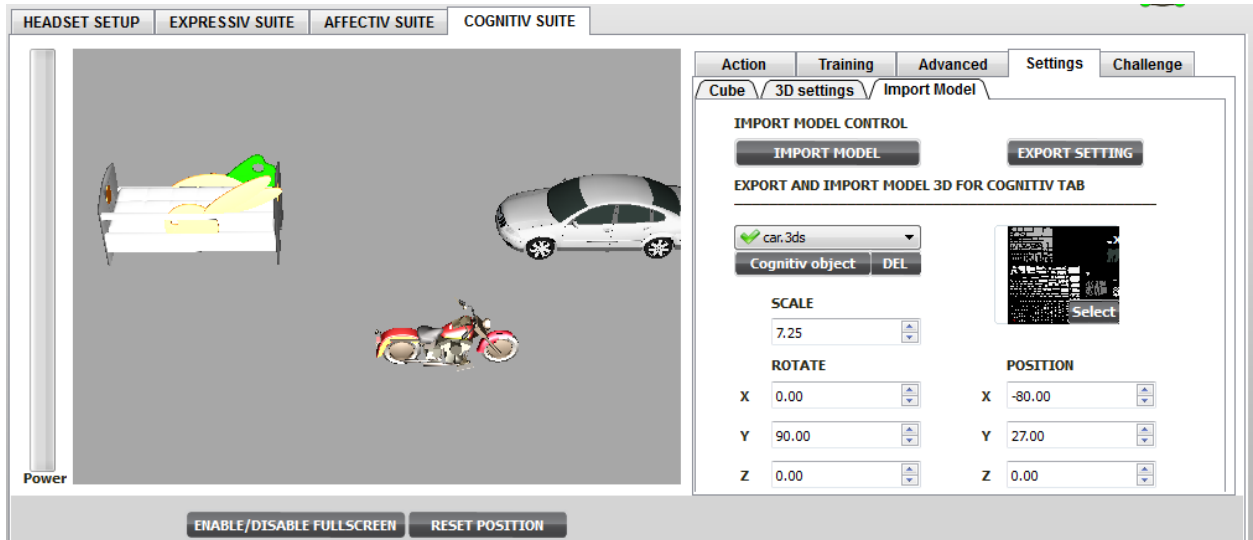
Figure 19   *Model after add texture*

Step 4: Select any model in that scene. You can use the EPOC headset to control it.
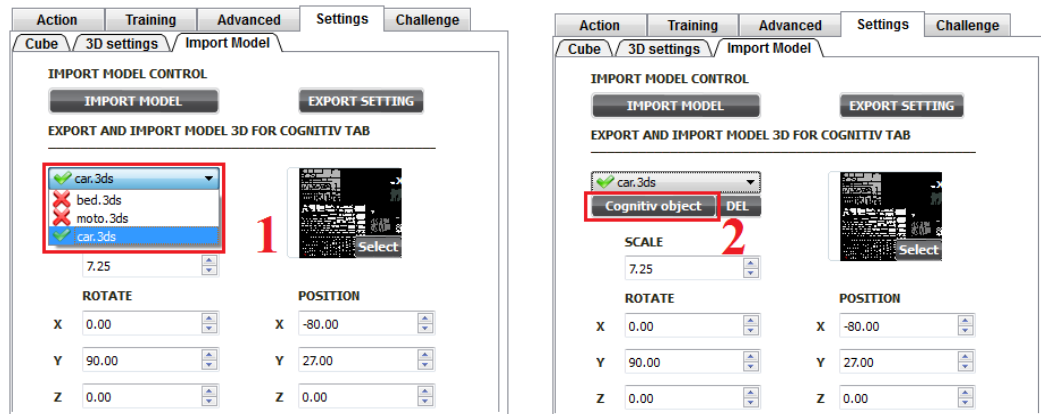
Figure 20   *Select model for Cognitiv*

Step 5: IMPORT MODEL and Export Setting

Export Setting: After you have created a scene you like and want to save it to be able to use it again next time you can use the Export Settings. This file is saved with an. ems
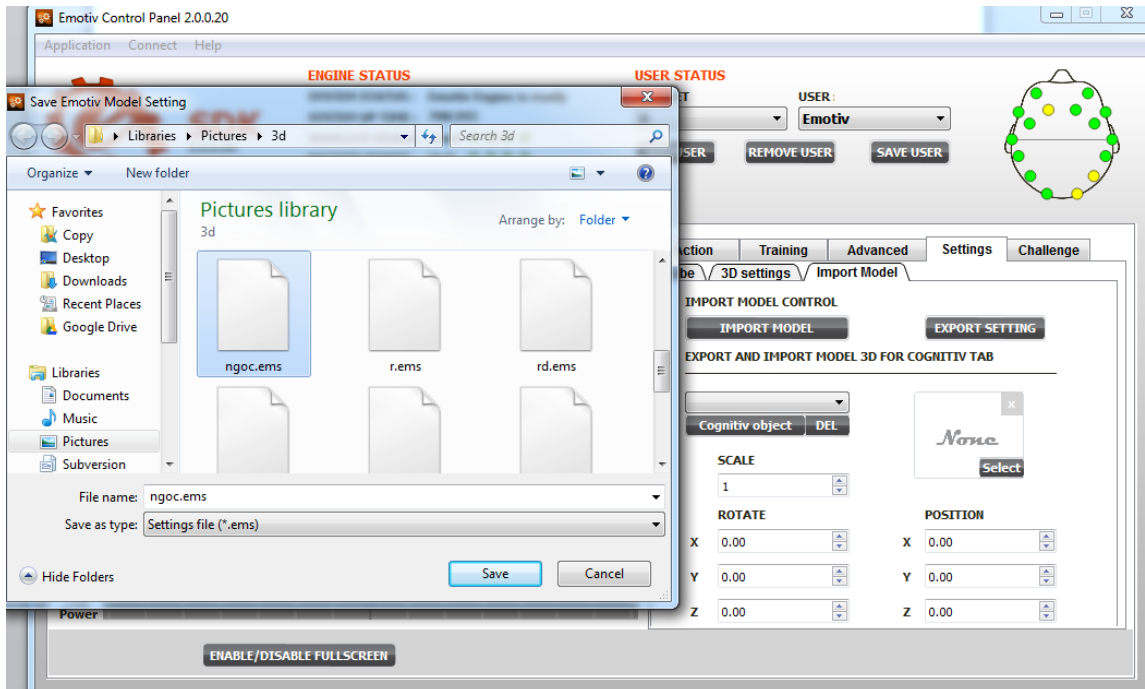
Figure 21 *EXPORT MODEL*

IMPORT MODEL: If you want to reload a scene which you have saved the previous use, you can reload it from .ems.

The file idea1.ems is provided as an example.
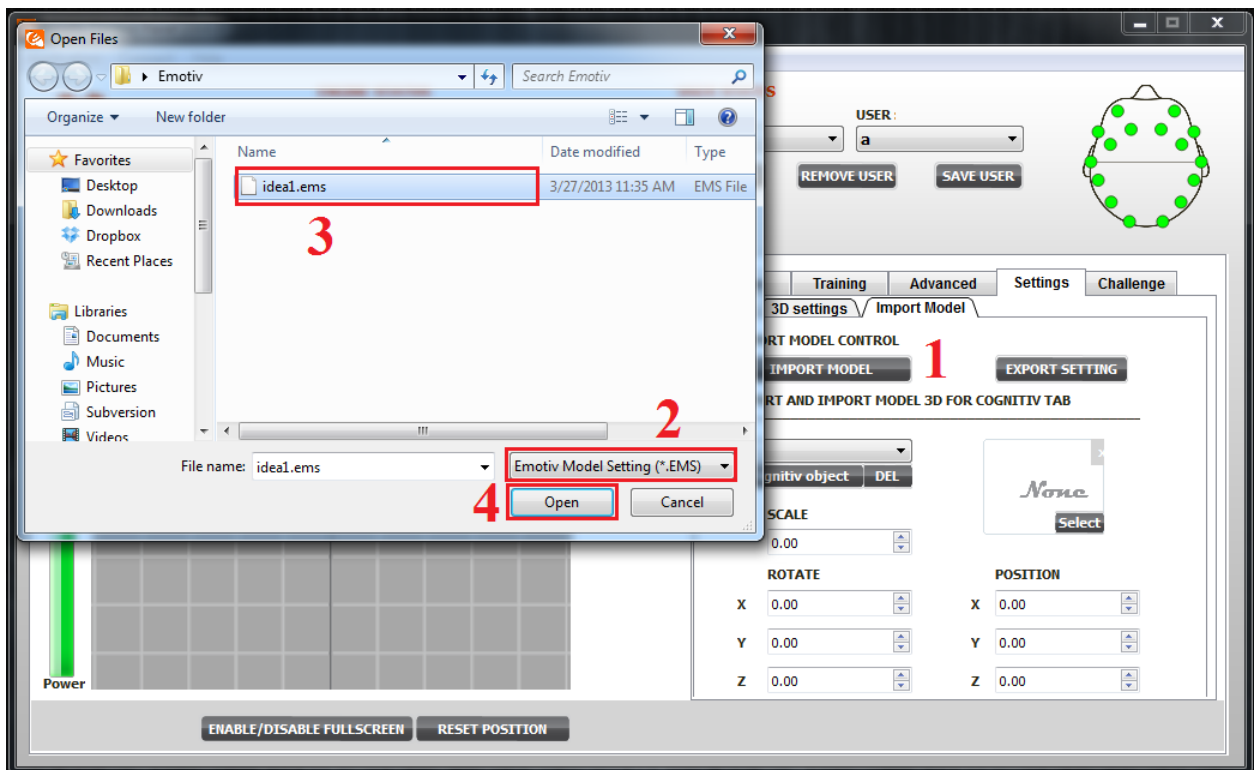
Figure 22   *IMPORT MODEL from .ems file*

The program also supports tools for manipulating the design:

Key G: enable/disable Grid

Key V: show camera Viewport

Key S: Pause/Run Action

3.5.9 Cognitiv Challenge

The Challenge tab allows you to play with cube model and submit your score to Emotiv's website. You need to select at least one trained action to begin play.
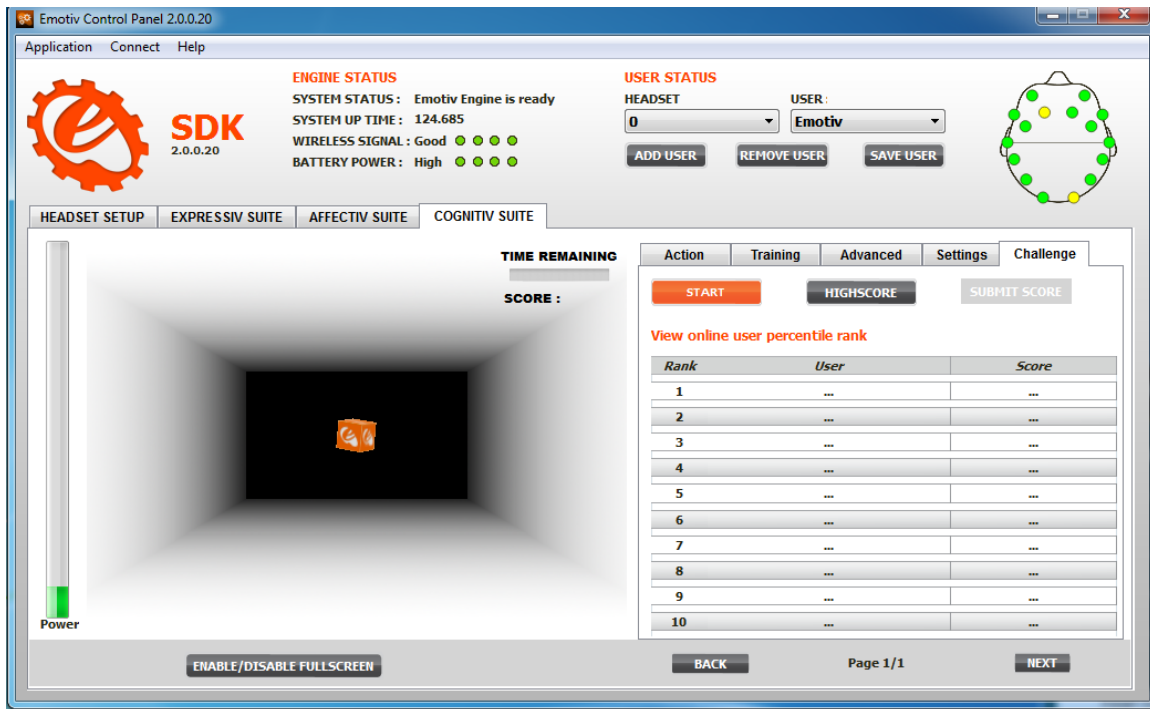
Figure 23   *The Challenge tab*

Click the START button to begin playing. After you finish playing, you can enter your Emotiv website login user name and password then click SUBMIT SCORE to submit score to Emotiv's website. The HIGHSCORE button displays the highest score online.

The right-hand table displays Scores recorded only on your local machine

4. Emotiv SDK Tools

This section explains the software utilities provided with the Emotiv SDK: EmoKey™ and EmoComposer™.

EmoKey allows you to connect detection results received from the EmoEngine to predefined keystrokes according to easy-to-define, logical rules. This functionality may be used to experiment with the headset as an input controller during application development. It also provides a mechanism for integrating the Emotiv neuroheadset with a preexisting application via the application's legacy keyboard interface.

EmoComposer emulates the behavior of the EmoEngine with an attached Emotiv neuroheadset. It is intended to be used as a development and testing tool; it makes it easy to send simulated EmoEngine events and request responses to applications using the Emotiv API in a transparent and deterministic way.

4.1 Introduction to EmoKey™

EmoKey translates Emotiv detection results to predefined sequences of keystrokes according to logical rules defined by the user through the EmoKey user interface. A set of rules, known as an "EmoKey Mapping", can be saved for later reuse. EmoKey communicates with Emotiv EmoEngine in the same manner as would a third-party application: by using the Emotiv API exposed by edk.dll.

4.1.1 Connecting EmoKey to Emotiv EmoEngine

By default, EmoKey will attempt to connect to Emotiv Control Panel when the application launches. If Emotiv Control Panel isn't running, then EmoKey will display a warning message above the system tray. The reason EmoKey connects to the Control Panel, instead of connecting directly to the EmoEngine and the neuroheadset, is to allow the user to select his profile, configure the detection suite settings, and get contact quality feedback through the Control Panel user interface. Please see Section 5.3 for more details about when Emotiv SDK developers might wish to follow a similar strategy.

EmoKey can also be connected to EmoComposer. This is useful when creating and testing a new EmoKey Mapping since the user can easily generate EmoState update events that satisfy the conditions for rules defined in EmoKey. Refer to Section 4.2 for more information about EmoComposer.
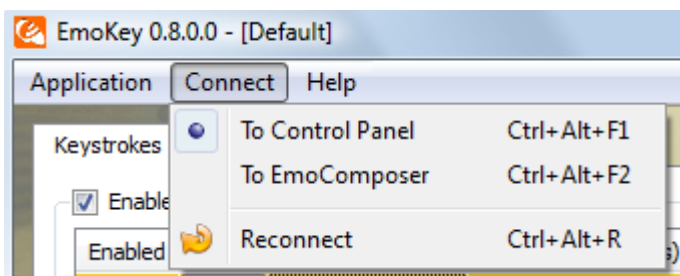
Figure 24   *EmoKey Connect Menu*

EmoKey's connection settings can be changed by using the application's **Connect** menu. If EmoKey is not able to connect to the target application (usually because the connection target is not running), then the EmoKey icon in the system tray will be drawn as gray, instead of orange). If this occurs, then run the desired application (either Emotiv Control Panel or EmoComposer) and then select **Reconnect** from the **Connect** menu.

4.1.2 Configuring EmoKey Rules
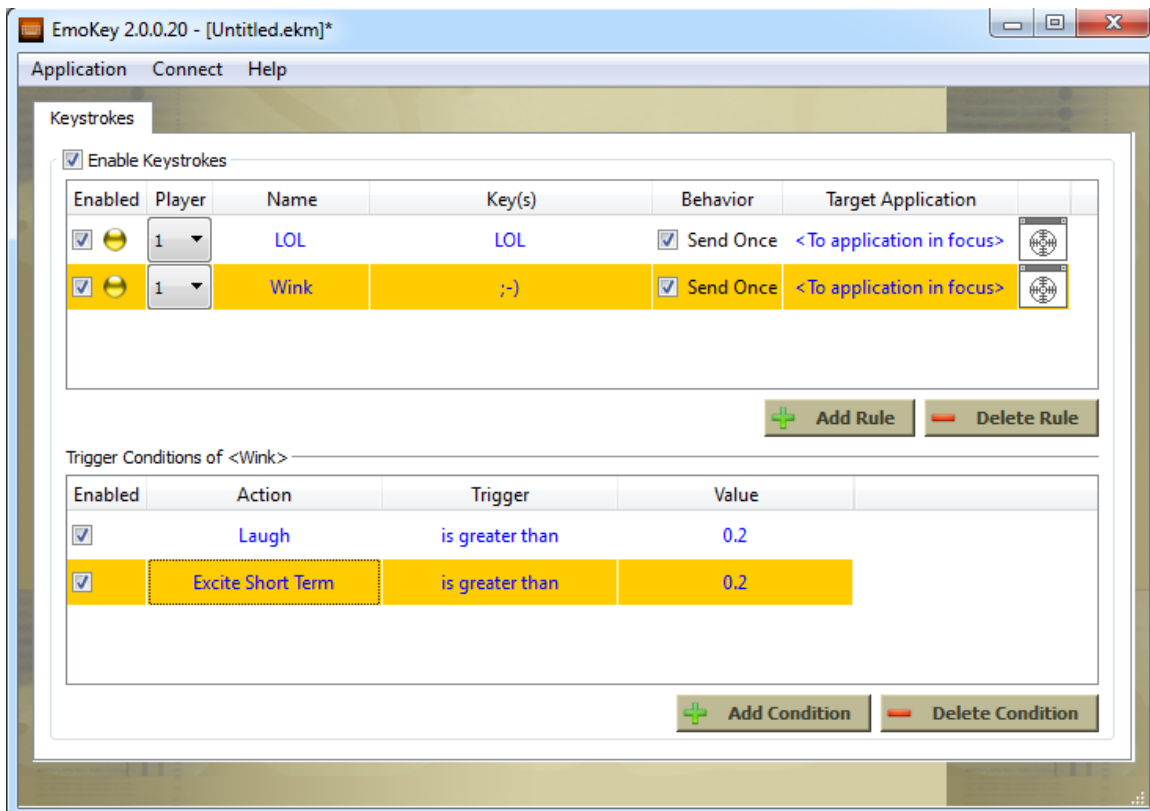
Figure 25   *Example EmoKey Mapping*

Figure 25 shows an example of an EmoKey Mapping as it might be configured to communicate with an Instant Messenger (IM) application. In this example, EmoKey will translate Laugh events generated by Emotiv's Expressiv Suite to the text "LOL" (as long as the Affectiv Suite's Instantaneous Excitement detection is also reporting a score > 0.3), which causes the IM program to send "LOL" automatically when the user is laughing.

The topmost table in EmoKey contains user interface controls that allow you to define rules that specify which keystrokes are emulated. Rules can be added by clicking on the ***Add Rule*** button. Existing rules can be deleted by selecting the appropriate row and pressing the ***Delete Rule*** button. In Figure 25, two rules, named "LOL" and "Wink", were added. Rules can be edited as outlined below, in Table 1.

Field	Description	Notes
Enabled	Checkbox to selectively enable or disable individual rules	The indicator "light" will turn green when the rule conditions are satisfied.
Player	Identifies the neuroheadset that is associated with this rule	Player 1 corresponds to user ID 0 in EmoComposer and Control Panel.
Name	User-friendly rule name	Edit by double clicking on the cell.
Key	Keystroke sequence to be sent to the Windows input queue	Edit by double clicking on the cell.
Behavior	Checkbox to control whether the key sequence is sent only once, or repeatedly, each time an EmoState update satisfies the rule conditions	If checked, then EmoKey must receive an EmoState update that does NOT satisfy the rule's conditions before this rule can be triggered again.

Table 1   EmoKey Rule Definition Fields

4.1.3 EmoKey Keyboard Emulation

EmoKey emulates a Windows-compatible keyboard and sends keyboard input to the Windows operating system's input queue. The application with the input focus will receive the emulated keystrokes. In practice, this often means that EmoKey is run in the background. Please note that closing the EmoKey window will only hide the application window and that it will continue to run. When running, the EmoKey/Emotiv icon will be visible in the Windows system tray. Double-click on this icon to bring EmoKey back to the foreground. Choose **Quit** from the **Application** or system-tray menu to really quit the application.
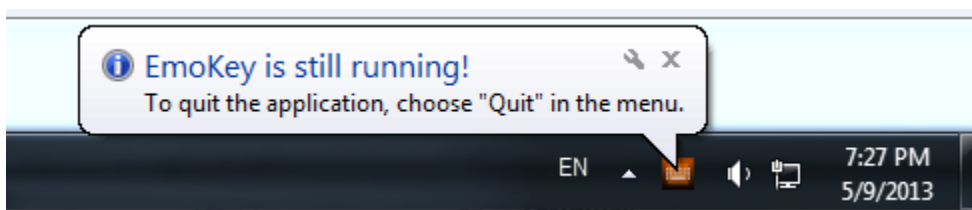
Figure 26   EmoKey System Tray Icon

Double-clicking in the **Key** field of a rule will bring up the **Keys** dialog as shown in **Error! Reference source not found.**.
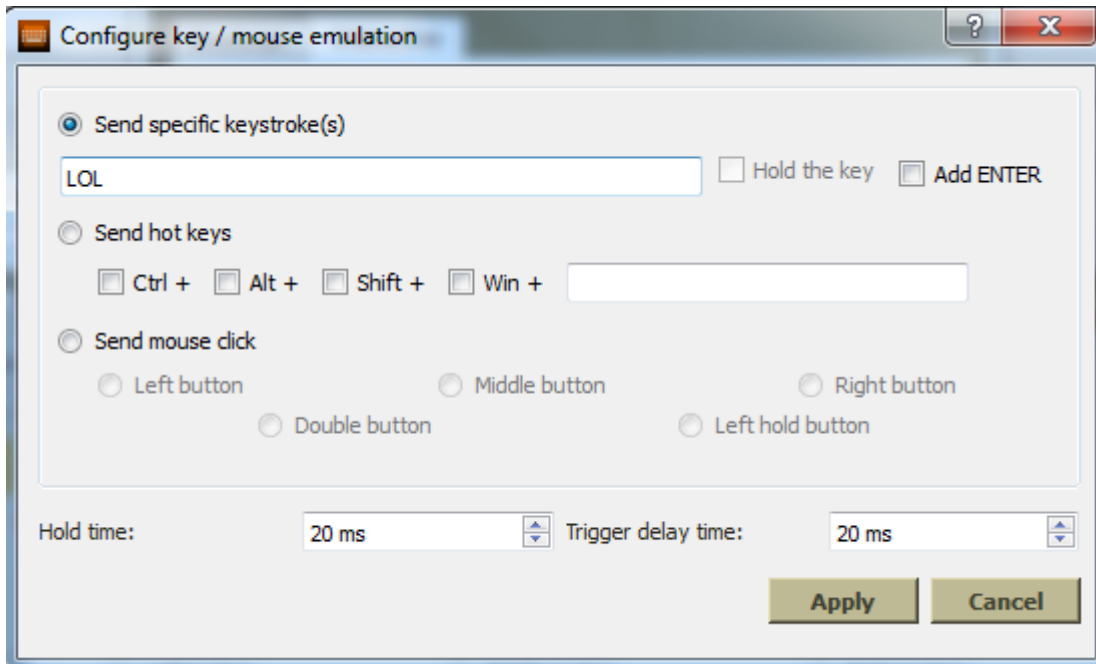
Figure 27 *Defining Keys and Keystroke Behavior*

The **Keys** dialog allows the user to specify the desired keystrokes and customize the keystroke behavior. The customizable options include:

- Holding a key press: hold the key down for the duration of the rule activation period. The **Hold the key** checkbox is only enabled when a single key has been specified in the keystroke edit box.
- Hot keys or special keyboard keys: any combination of control, alt, shift, the Windows key, and another keystroke. You may also use this option if you need to specify special keys such as Caps Lock, Shift, or Enter.
- Key press duration and delay times: some applications, especially games, are sensitive to the timing of key presses. If necessary, use these controls to adjust the simulated keyboard behavior.

4.1.4 Configuring EmoKey Rule Trigger Conditions

The **Trigger Conditions** table in EmoKey contains user interface controls that allow you to define logical conditions that determine when the corresponding rule is activated. Clicking on a new rule in the Rules table will refresh the contents of the **Trigger Conditions** table, causing it to display only the conditions associated with the selected rule.

Conditions can be added by clicking on the **Add Condition** button. Existing rules can be deleted by selecting the appropriate condition and pressing the **Delete Condition** button. In Figure 25, two conditions, which examine the state of the Expressiv Laugh detection and the Affectiv Instantaneous Excitement detection, respectively, are associated with the LOL rule. All enabled conditions must be satisfied by the most recent EmoState update received from Emotiv Control Panel or EmoComposer for a rule to be triggered.

The fields of the **Trigger Conditions** table are described below, in Table 2.

Field	Description
Enabled	Checkbox to selectively enable or disable individual trigger conditions
Action	The name of the Expressiv expression, Affectiv detection, or Cognitiv action being examined by this condition.
Trigger	Description of the trigger condition being evaluated
Value	For non-binary trigger conditions, the value being compared to the action score returned by the designated detection

Table 2   *EmoKey Trigger Condition Fields*

Double-clicking on any of the fields of a condition in the **Trigger Conditions** table will reveal the **Configure Condition** dialog box, as shown in Figure 28. Use the controls on this dialog to specify an action (or detection) name, a comparison function, and a value, that must evaluate to true for this condition to be satisfied.
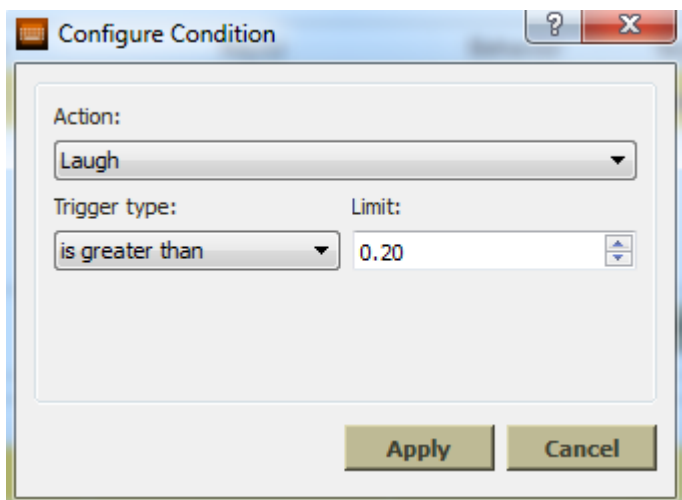
Figure 28   *Defining an EmoKey Condition*

4.1.5 Saving Rules to an EmoKey Mapping file

EmoKey allows you to save the current set of rule definitions to an EmoKey Mapping file that can be reloaded for subsequent use. Use the appropriate command in EmoKey's **Application** menu to rename, save, and load EmoKey mapping files.

4.2 EmoComposer™ usage

EmoComposer allows you to send user-defined EmoStates™ to Emotiv Control Panel, EmoKey, or any other application that makes use of the Emotiv API. EmoComposer supports two modes of EmoState generation: Interactive mode and EmoScript mode. In addition to generating EmoStates, EmoComposer can also simulate Emotiv EmoEngine's handling of profile management and training requests.

SDKLite users will rely on EmoComposer to simulate the behavior of Emotiv EmoEngine and Emotiv neuroheadsets. However, it is a very useful tool for all Emotiv SDK developers, allowing for easy experimentation with the Emotiv API early in the development process, and facilitating manual and automated testing later in the development cycle.

4.2.1 Interactive mode
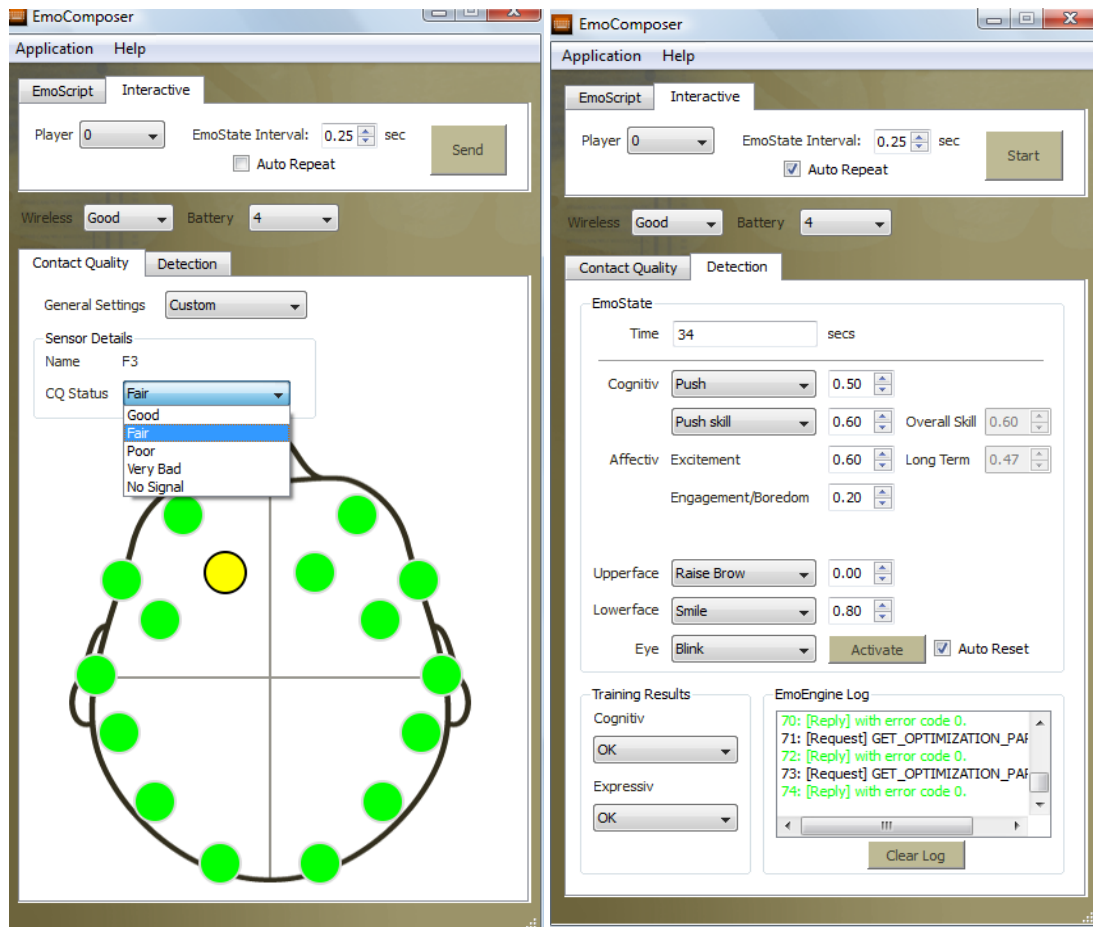
Figure 29   *EmoComposer interactive mode*

EmoComposer Interactive mode allows you to define and send specific EmoState values to any application using the Emotiv SDK. The user interface settings are described below.

- ***Player***: choose the player number who's EmoState you wish to define and send. The player number defaults to 0. When the player number is changed for the first time, an application connected to EmoComposer will receive an `EE_UserAdded` event with the new player number reported as the user ID.
- ***Wireless***: sets the simulated wireless signal strength. Note: if the signal strength is set to "Bad" or "No Signal" then EmoComposer™ simulates the behavior of the EmoEngine by setting subsequent EmoState detection values and signal quality values to 0.
- ***Contact Quality* tab**: this tab allows you to adjust the reported contact quality for each sensor on the Emotiv neuroheadset. When the ***Contact Quality*** tab is active, all other EmoState detection values are set to 0. You can choose between typical sensor signal quality readings by selecting a preset from the ***General Settings*** drop-down list box. If you choose the ***Custom*** preset, each sensor value can be controlled

individually by clicking on a sensor and then selecting a new CQ Status value in the **Sensor Details** box. Note that the two sensors above and behind the ears, correspond to the reference sensors on the Emotiv SDK Neuroheadset, must always report a CQ value of **Good** and cannot be adjusted.

- **Detection tab**: this tab allows you to interactively control EmoState™ detection values and training result values. When the Detection tab is active, the contact quality values for generated EmoStates will always be set to `EEG_CQ_GOOD`.

- **EmoState**: define the detection settings in an EmoState™ by selecting the particular event type for each detection group in the appropriate drop-down list box. Set the event's value in the spin box adjacent to the event name. You can define your own time value in the **Time** edit box, or allow EmoComposer™ to set the value automatically by incrementing it by the value of the **EmoState Interval** spin box after each EmoState™ is sent. The Affectiv™ Excitement state is unique in that the EmoEngine returns both short-term (for Instantaneous Excitement) and long-term values. EmoComposer™ simulates the long-term value calculation adequately enough for testing purposes but does not reproduce the exact algorithm used by the Affectiv detection suite in EmoEngine™. Note that the value for the eye detections is binary (Active or Inactive) and that it is automatically reset to be Inactive after an EmoState™ is sent. If Auto Repeat mode is active, then you can press and hold the **Activate** button to maintain a particular eye state across multiple time intervals. Also note that the value for a neutral Cognitiv™ detection is automatically set to 0.

- **Training Results**: specify the desired return value for EmoEngine™ requests generated for the current player by the `EE_CognitivSetTrainingControl` and `EE_ExpressivSetTrainingControl` functions.

- **EmoEngine Log**: contents are intended to give developers a clearer picture about how the EmoEngine processes requests generated by various Emotiv API functions. The log displays three different output types: `Request, Reply, CogResult`, and `ExpResult`. An API function call that results in a new request to the EmoEngine will cause a `Request` output line to be displayed in the log. The multitude of API functions are translated to roughly a dozen different strings intended to allow the Emotiv SDK developer to see that an API function call has been serviced . These strings include: `PROFILE_ADD_USER, PROFILE_CHANGE_USER, PROFILE_REMOVE_USER, PROFILE_LIST_USER, PROFILE_GET_CURRENT_USER, PROFILE_LOAD, PROFILE_SAVE, EXPRESSIV_GET, EXPRESSIV_SET, AFFECTIV_GET, AFFECTIV_SET, COGNITIV_SET` and `COGNITIV_GET`. Because of the comparatively complex API protocol used to facilitate training the Cognitiv™ algorithms, we display additional detail when we receive training control messages generated by the `EE_CognitivSetTrainingControl` API function. These strings are: `COGNITIV_START, COGNITIV_ACCEPT`, and `COGNITIV_REJECT`, which correspond to the `EE_TrainingControl_t` constants exposed to developers in edk.h. Similar strings are used for the equivalent Expressiv messages. All other request types are displayed as `API_REQUEST`. The `Reply` output line displays the error code and is either green or red, depending on whether an error has occurred (i.e. the error code != 0). The `CogResult` and `ExpResult` outputs are used to inform the developer of an asynchronous response sent from the EmoEngine via an EmoState update as the result of an active Cognitiv or Expressiv training request.

- **Send**: sends the EmoState™ to a connected application or the Emotiv Control Panel. Note that Control Panel will display "Cannot Acquire Data" until the first EmoState™ is received from EmoComposer.

- **Auto Repeat**: check this box to tell EmoComposer™ to automatically send EmoStates™ at the time interval specified in the EmoState™ Interval spin box. Use the

Start/Stop button to turn the automated send on and off. You may interact with the EmoState™ controls to dynamically change the EmoState™ values while the automated send is active. Note: switching between the ***Contact Quality*** and ***Detection*** tabs in Interactive mode will automatically stop an automated send.

4.2.2 EmoScript Mode
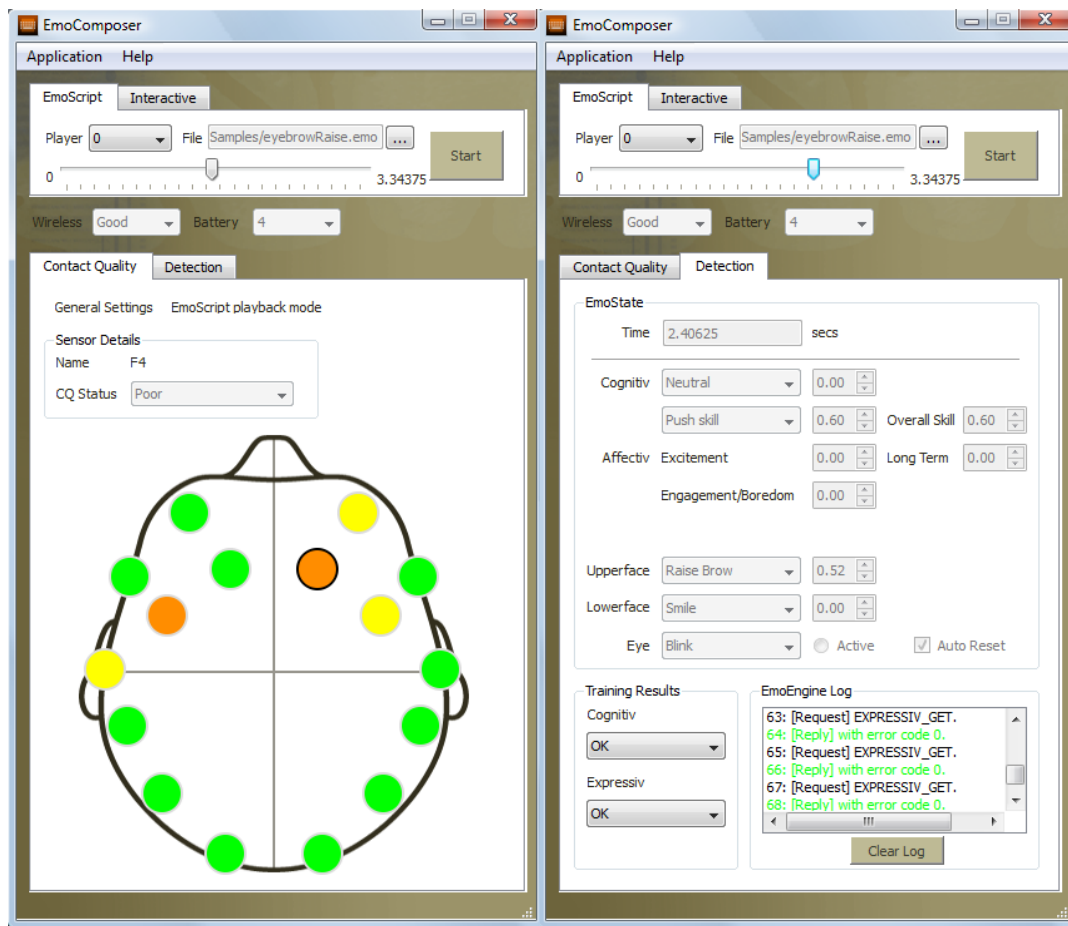
Figure 30 *EmoComposer EmoScript Mode*

EmoComposer™ EmoScript mode allows you to playback a predefined sequence of EmoState™ values to any application using the EmoEngine. The user interface settings are described below. EmoScript files are written in EML (EmoComposer™ Markup Language). EML syntax details can be found in the EML Language Specification section in Appendix 1 of this document.

- ***Player***: choose the player number to associate with the generated EmoStates™.
- ***File***: click the "..." button to select and load an EmoScript file from disk. If the file loads successfully then the timeline slider bar and Start button will be activated. If an error occurs, a message box will appear with a description and approximate location in the file.
- ***Timeline Slider***: move the slider control to see the EmoState™ and signal quality values for any point on the timeline defined by the EmoScript file.
- ***Start/Stop* button**: starts and stops the playback of the EmoState™ values generated by the EmoScript file.
- ***Wireless***: the wireless signal strength setting is disabled while in EmoScript mode and the wireless signal strength is always set to "Good."

- **Contact Quality tab**: the indicators on the head model correspond to the values defined by the `contact_quality` tag at specific time code in the EmoScript file. If no `contact_quality` tag has been specified then the contact quality values in the generated EmoStates default to `CQ_GOOD`.
- **Detection tab**: this tab allows you to view EmoState detection values and provides interactive control over training result values. Unlike Real Time Interactive mode, the signal quality and detection values are determined entirely by the contents of the EmoScript file and you can switch between the Signal Quality tab and the Detection tab to view the appropriate data during playback or timeline navigation.
- **EmoState**: the values displayed correspond to the EmoState™ values for a particular point in time as defined by the EmoScript file. Note that these EmoScript values are not interactive and cannot be modified by the user (use the Interactive mode for this instead).
- **Training Results and EmoEngine Log**: these controls operate exactly the same as they do in Interactive Mode. See the Interactive Mode documentation (above) for more details.

5. Programming with the Emotiv SDK

5.1 Overview

This section introduces key concepts for using the Emotiv SDK to build software that is compatible with Emotiv headsets. It also walks you through some sample programs that demonstrate these concepts and serve as a tutorial to help you get started with the Emotiv API. The sample programs are written in C++ and are intended to be compiled with Microsoft Visual Studio 2005 (Visual Studio 2008 is also supported). They are installed with the Emotiv SDK and are organized into a Microsoft Visual Studio 2005 solution, EmoTutorials.sln, which can be found in the \doc\Examples directory of your installation.

5.2 Introduction to the Emotiv API and Emotiv EmoEngine™

The Emotiv API is exposed as an ANSI C interface that is declared in 3 header files (edk.h, EmoStateDLL.h, edkErrorCode.h) and implemented in 2 Windows DLLs (edk.dll and edk_utils.dll). C or C++ applications that use the Emotiv API simply include edk.h and link with edk.dll. See Appendix 4 for a complete description of redistributable Emotiv SDK components and installation requirements for your application.

The Emotiv EmoEngine refers to the logical abstraction of the functionality that Emotiv provides in edk.dll. The EmoEngine communicates with the Emotiv headset, receives preprocessed EEG and gyroscope data, manages user-specific or application-specific settings, performs post-processing, and translates the Emotiv detection results into an easy-to-use structure called an EmoState. Emotiv API functions that modify or retrieve EmoEngine settings are prefixed with "EE_."
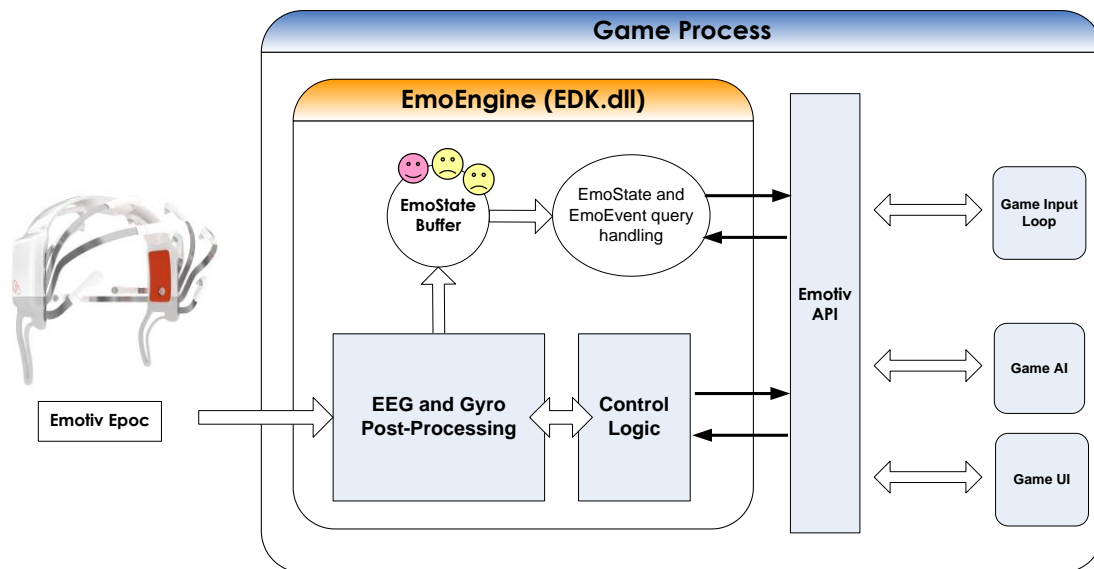
Figure 1 Integrating the EmoEngine and Emotiv EPOC with a videogame

An EmoState is an opaque data structure that contains the current state of the Emotiv detections, which, in turn, reflect the user's facial, emotional and cognitive state. EmoState data is retrieved by Emotiv API functions that are prefixed with "ES_."

EmoStates and other Emotiv API data structures are typically referenced through opaque handles (e.g. `EmoStateHandle` and `EmoEngineEventHandle`). These data structures and their handles are allocated and freed using the appropriate Emotiv API functions (e.g. `EE_EmoEngineEventCreate` and `EE_EmoEngineEventFree`).
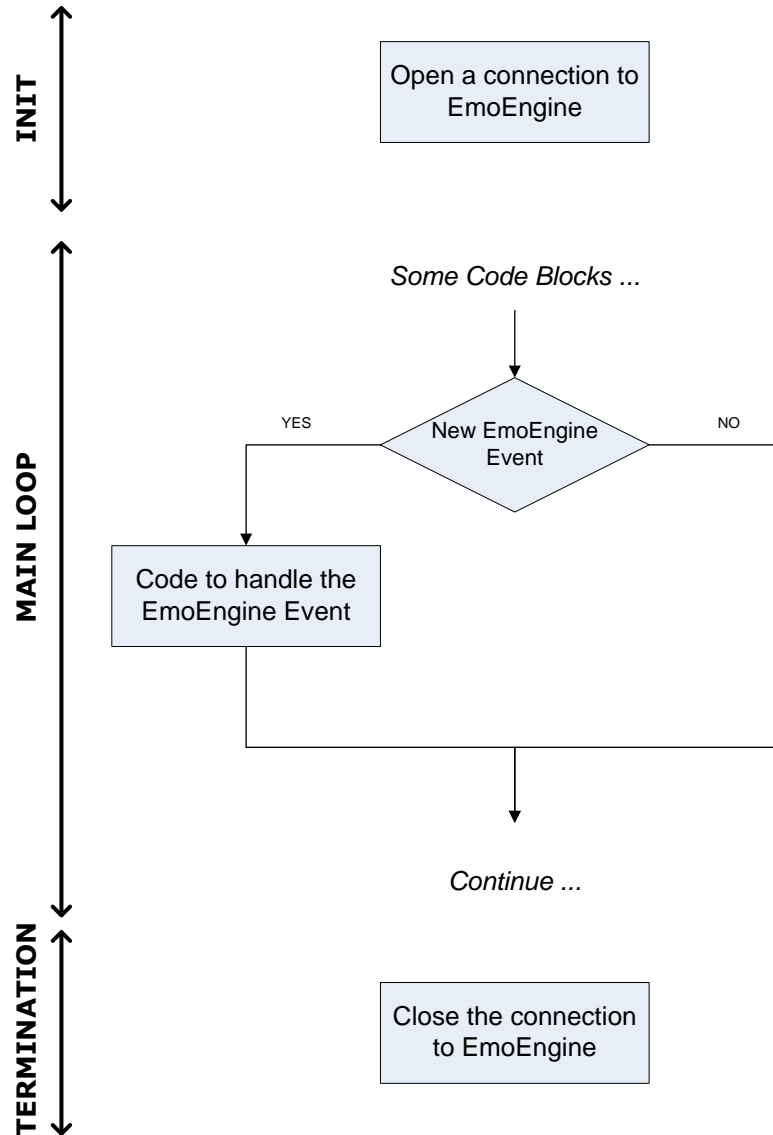
Figure 31   *Using the API to communicate with the EmoEngine*

Figure 31 above shows a high-level flow chart for applications that incorporate the EmoEngine.   During initialization, and prior to calling Emotiv API functions, your application must establish a connection to the EmoEngine by calling `EE_EngineConnect` or `EE_EngineRemoteConnect`. Use `EE_EngineConnect` when you wish to communicate directly with an Emotiv headset. Use `EE_EngineRemoteConnect` if you are using SDKLite and/or wish to connect your application to EmoComposer or Emotiv Control Panel. More details about using `EE_EngineRemoteConnect` follow in Section 5.3.

The EmoEngine communicates with your application by publishing events that can be retrieved by calling `EE_EngineGetNextEvent()`. For near real-time responsiveness, most applications should poll for new EmoStates at least 10-15 times per second. This is typically done in an application's main event loop or, in the case of most videogames, when other input devices are periodically queried. Before your application terminates, the connection to EmoEngine should be explicitly closed by calling `EE_EngineDisconnect()`.

There are three main categories of EmoEngine events that your application should handle:

- **Hardware-related events**: Events that communicate when users connect or disconnect Emotiv input devices to the computer (e.g. `EE_UserAdded`).
- **New EmoState events**: Events that communicate changes in the user's facial, cognitive and emotional state. You can retrieve the updated EmoState by calling `EE_EmoEngineEventGetEmoState()`. (e.g. `EE_EmoStateUpdated`).
- **Suite-specific events**: Events related to training and configuring the Cognitiv and Expressiv detection suites (e.g. `EE_CognitivEvent`).

A complete list of all EmoEngine events can be found in Appendix 3 .

Most Emotiv API functions are declared to return a value of type int. The return value should be checked to verify the correct operation of the API function call. Most Emotiv API functions return `EDK_OK` if they succeed. Error codes are defined in edkErrorCode.h and documented in Appendix 2.

5.3 Development Scenarios Supported by EE_EngineRemoteConnect

The `EE_EngineRemoteConnect()` API should be used in place of `EE_EngineConnect()` in the following circumstances:

1. The application is being developed with Emotiv SDKLite. This version of the SDK does not include an Emotiv headset so all Emotiv API function calls communicate with EmoComposer, the EmoEngine emulator that is described in Section 4.2. EmoComposer listens on port 1726 so an application that wishes to connect to an instance of EmoComposer running on the same computer must call `EE_EngineRemoteConnect("127.0.0.1", 1726)`.

2. The developer wishes to test his application's behavior in a deterministic fashion by manually selecting which Emotiv detection results to send to the application. In this case, the developer should connect to EmoComposer as described in the previous item.

3. The developer wants to speed the development process by beginning his application integration with the EmoEngine and the Emotiv headset without having to construct all of the UI and application logic required to support detection tuning, training, profile management and headset contact quality feedback. To support this case, Emotiv Control Panel can act as a proxy for either the real, headset-integrated EmoEngine or EmoComposer. Control Panel listens on port 3008 so an application that wishes to connect to Control Panel must call `EE_EngineRemoteConnect("127.0.0.1", 3008)`.

5.4 Example 1 – EmoStateLogger

This example demonstrates the use of the core Emotiv API functions described in Sections 5.2 and 5.3. It logs all Emotiv detection results for the attached users after successfully establishing a connection to Emotiv EmoEngine™ or EmoComposer™.

```
// ... print some instructions...
std::string input;
std::getline(std::cin, input, '\n');
option = atoi(input.c_str());

switch (option) {
   case 1: {
         if (EE_EngineConnect() != EDK_OK) {
               throw exception("Emotiv Engine start up failed.");
         }
         break;
   }
   case 2: {
         std::cout << "Target IP of EmoComposer? [127.0.0.1] ";
         std::getline(std::cin, input, '\n');
         if (input.empty()) {
               input = std::string("127.0.0.1");
         }
         if (EE_EngineRemoteConnect(input.c_str(), 1726) != EDK_OK){
               throw exception("Cannot connect to EmoComposer!");
         }
         break;
   }
   default:
         throw exception("Invalid option...");
         break;
}
```

Listing 1 *Connect to the EmoEngine*

The program first initializes the connection with Emotiv EmoEngine™ by calling `EE_EngineConnect()` or, with EmoComposer™, via `EE_EngineRemoteConnect()` together with the target IP address of the EmoComposer machine and the fixed port 1726. It ensures that *t*he remote connection has been successfully established by verifying the return value of the `EE_EngineRemoteConnect()` function.

```
EmoEngineEventHandle eEvent= EE_EmoEngineEventCreate();
EmoStateHandle eState       = EE_EmoStateCreate();
unsigned int userID         = 0;
while (...) {
int state = EE_EngineGetNextEvent(eEvent);
// New event needs to be handled
if (state == EDK_OK) {
        EE_Event_t eventType = EE_EmoEngineEventGetType(eEvent);
        EE_EmoEngineEventGetUserId(eEvent, &userID);
        // Log the EmoState if it has been updated
        if (eventType == EE_EmoStateUpdated) {
            // New EmoState from user
            EE_EmoEngineEventGetEmoState(eEvent, eState);
            // Log the new EmoState
            logEmoState(ofs, userID, eState, writeHeader);
            writeHeader = false;
        }
    }
}
```

Listing 2   *Buffer creation and management*

An `EmoEngineEventHandle` is created by `EE_EmoEngineEventCreate()`. An EmoState™ buffer is created by calling `EE_EmoStateCreate()`. The program then queries the EmoEngine to get the current EmoEngine event by invoking `EE_EngineGetNextEvent()`. If the result of getting the event type using `EE_EmoEngineEventGetType()` is `EE_EmoStateUpdated`, then there is a new detection event for a particular user (extract via `EE_EmoEngineEventGetUserID()`). The function `EE_EmoEngineEventGetEmoState()` can be used to copy the EmoState™ information from the event handle into the pre-allocated EmoState buffer.

Note that `EE_EngineGetNextEvent()` will return `EDK_NO_EVENT` if no new events have been published by EmoEngine since the previous call. The user should also check for other error codes returned from `EE_EngineGetNextEvent()` to handle potential problems that are reported by the EmoEngine.

Specific detection results are retrieved from an EmoState by calling the corresponding EmoState accessor functions defined in `EmoState.h`. For example, to access the blink detection, `ES_ExpressivIsBlink(eState)` should be used.

```
EE_EngineDisconnect();
EE_EmoStateFree(eState);
EE_EmoEngineEventFree(eEvent);
```

Listing 3   *Disconnecting from the EmoEngine*

Before the end of the program, `EE_EngineDisconnect()` is called to terminate the connection with the EmoEngine and free up resources associated with the connection. The user should also call `EE_EmoStateFree()` and `EE_EmoEngineEventFree()` to free up memory allocated for the EmoState buffer and `EmoEngineEventHandle`.

Before compiling the example, use the **Property Pages** and set the **Configuration Properties→Debugging→Command Arguments** to the name of the log file you wish to create, such as **log.txt**, and then build the example.

To test the example, launch EmoComposer. Start a new instance of EmoStateLogger and when prompted, select option 2 (**Connect to EmoComposer**). The EmoStates generated by EmoComposer will then be logged to the file **log.txt**.

Tip: If you examine the log file, and it is empty, it may be because you have not used the controls in the EmoComposer to generate any EmoStates. SDKLite users should only choose option 2 to connect to EmoComposer since option 1 (**Connect to EmoEngine**) assumes that the user will attach a neuroheadset to the computer.

5.5 Example 2 – Expressiv™ Demo

This example demonstrates how an application can use the Expressiv™ detection suite to control an animated head model called BlueAvatar. The model emulates the facial expressions made by the user wearing an Emotiv headset. As in Example 1, ExpressivDemo connects to Emotiv EmoEngine™ and retrieves EmoStates™ for all attached users. The EmoState is examined to determine which facial expression best matches the user's face. ExpressivDemo communicates the detected expressions to the separate BlueAvatar application by sending a UDP packet which follows a simple, pre-defined protocol.

The Expressiv state from the EmoEngine can be separated into three groups of mutually-exclusive facial expressions:

- **Upper face actions**: Raised eyebrows, furrowed eyebrows
- **Eye related actions**: Blink, Wink left, Wink right, Look left, Look right
- **Lower face actions**: Smile, Smirk left, Smirk right, Clench, Laugh

```
EmoStateHandle eState = EE_EmoStateCreate();
...
EE_ExpressivAlgo_t upperFaceType =
ES_ExpressivGetUpperFaceAction(eState);
EE_ExpressivAlgo_t lowerFaceType =
ES_ExpressivGetLowerFaceAction(eState);
float upperFaceAmp = ES_ExpressivGetUpperFaceActionPower(eState);
float lowerFaceAmp = ES_ExpressivGetLowerFaceActionPower(eState);
```

Listing 4   *Excerpt from ExpressivDemo code*

This code fragment from ExpressivDemo shows how upper and lower face actions can be extracted from an EmoState buffer using the Emotiv API functions `ES_ExpressivGetUpperFaceAction()` and `ES_ExpressivGetLowerFaceAction()`, respectively. In order to describe the upper and lower face actions more precisely, a floating point value ranging from 0.0 to 1.0 is associated with each action to express its "power", or degree of movement, and can be extracted via the `ES_ExpressivGetUpperFaceActionPower()` and `ES_ExpressivGetLowerFaceActionPower()` functions.

Eye and eyelid-related state can be accessed via the API functions which contain the corresponding expression name such as `ES_ExpressivIsBlink()`, `ES_ExpressivIsLeftWink()`, `ES_ExpressivIsLookingRight()`, etc.

The protocol that ExpressivDemo uses to control the BlueAvatar motion is very simple. Each facial expression result will be translated to plain ASCII text, with the letter prefix describing the type of expression, optionally followed by the amplitude value if it is an upper or lower face action. Multiple expressions can be sent to the head model at the

same time in a comma separated form. However, only one expression per Expressiv grouping is permitted (the effects of sending smile and clench together or blinking while winking are undefined by the BlueAvatar). Table 3 below excerpts the syntax of some of expressions supported by the protocol.

Expressiv™ action type	Corresponding ASCII Text (case sensitive)	Amplitude value
Blink	B	n/a
Wink left	l	n/a
Wink right	r	n/a
Look left	L	n/a
Look right	R	n/a
Eyebrow	b	0 to 100 integer
Smile	S	0 to 100 integer
Clench	G	0 to 100 integer

Table 3   *BlueAvatar control syntax*

Some examples:
- Blink and smile with amplitude 0.5: **B,S50**
- Eyebrow with amplitude 0.6 and clench with amplitude 0.3: **b60, G30**
- Wink left and smile with amplitude 1.0: **l, S100**

The prepared ASCII text is subsequently sent to the BlueAvatar via UDP socket. ExpressivDemo supports sending expression strings for multiple users. BlueAvatar should start listening to port 30000 for the first user. Whenever a subsequent Emotiv USB receiver is plugged-in, ExpressivDemo will increment the target port number of the associated BlueAvatar application by one. Tip: when an Emotiv USB receiver is removed and then reinserted, ExpressivDemo will consider this as a new Emotiv EPOC and still increases the sending UDP port by one.

In addition to translating Expressiv results into commands to the BlueAvatar, the ExpressivDemo also implements a very simple command-line interpreter that can be used to demonstrate the use of personalized, trained signatures with the Expressiv suite. Expressiv supports two types of "signatures" that are used to classify input from the Emotiv headset as indicating a particular facial expression.

The default signature is known as the universal signature, and it is designed to work well for a large population of users for the supported facial expressions. If the application or user requires more accuracy or customization, then you may decide to use a trained signature. In this mode, Expressiv requires the user to train the system by performing the desired action before it can be detected. As the user supplies more training data, the accuracy of the Expressiv detection typically improves. If you elect to use a trained signature, the system will only detect actions for which the user has supplied training data. The user must provide training data for a neutral expression and at least one other supported expression before the trained signature can be activated. Important note: not all Expressiv expressions can be trained. In particular, eye and eyelid-related expressions (i.e. "blink", "wink", "look left", and "look right") can not be trained.

The API functions that configure the Expressiv detections are prefixed with "EE_Expressiv." The **training_exp** command corresponds to the `EE_ExpressivSetTrainingAction()`

function. The **trained_sig** command corresponds to the `EE_ExpressivGetTrainedSignatureAvailable()` function. Type "help" at the ExpressivDemo command prompt to see a complete set of supported commands.

The figure below illustrates the function call and event sequence required to record training data for use with Expressiv. It will be useful to first familiarize yourself with the training procedure on the Expressiv tab in Emotiv Control Panel before attempting to use the Expressiv training API functions.

```
Expressiv Detection          EDK API              Application
      |                          |                      |
      |<-------- EE_ExpressivSetTrainingAction( EE_ExpressivAlgo_t ) ------|
      |                          |                      |
      |<-------- EE_ExpressivSetTrainingControl( EXP_START ) -------------|
      |  )                       |                      |
      |    2 sec delay           |                      |
      |<-|                       |                      |
      | EE_ExpressivTrainingStarted event -------------------------------->|
      |  )                       |                      |
      |    8 sec training period |                      |
      |<-|                       |                      |
      |                          |                      |
      | Signal is noisy : EE_ExpressivTrainingFailed event --------------->|
      |                          |                      |) Restart Training
      |                          |                      |<-|
      | Signal is clean : EE_ExpressivTrainingSucceeded event ----------->|
      |                          |                      |) Ask for Accept/Reject
      |                          |                      |<-|
      |<-------- EE_ExpressivSetTrainingControl( EXP_REJECT ) ------------|
      |                          |                      |
      |<-------- EE_ExpressivSetTrainingControl( EXP_ACCEPT ) ------------|
      |  )                       |                      |
      |    update signature      |                      |
      |<-|                       |                      |
      | EE_ExpressivTrainingCompleted event ----------------------------->|
      |                          |                      |
```

Figure 32 *Expressiv training command and event sequence*

The below sequence diagram describes the process of training an Expressiv facial expression. The Expressiv-specific training events are declared as enumerated type `EE_ExpressivEvent_t` in `EDK.h`. Note that this type differs from the `EE_Event_t` type used by top-level EmoEngine Events.

```
EmoEngineEventHandle eEvent= EE_EmoEngineEventCreate();
if (EE_EngineGetNextEvent(eEvent) == EDK_OK) {
    EE_Event_t eventType = EE_EmoEngineEventGetType(eEvent);
    if (eventType == EE_ExpressivEvent) {
        EE_ExpressivEvent_t cEvt=EE_ExpressivEventGetType(eEvent);
        ...
}
```

Listing 5   *Extracting Expressiv event details*

Before the start of a training session, the expression type must be first set with the API function `EE_ExpressivSetTrainingAction()`. In `EmoStateDLL.h`, the enumerated type `EE_ExpressivAlgo_t` defines all the expressions supported for detection. Please note, however, that only non-eye-related detections (lower face and upper face) can be trained. If an expression is not set before the start of training, `EXP_NEUTRAL` will be used as the default.

`EE_ExpressivSetTrainingControl()` can then be called with argument `EXP_START` to start the training the target expression. In `EDK.h`, enumerated type `EE_ExpressivTrainingControl_t` defines the control command constants for Expressiv training. If the training can be started, an `EE_ExpressivTrainingStarted` event will be sent after approximately 2 seconds. The user should be prompted to engage and hold the desired facial expression prior to sending the `EXP_START command`. The training update will begin after the EmoEngine sends the `EE_ExpressivTrainingStarted` event. This delay will help to avoid training with undesirable EEG artifacts resulting from transitioning from the user's current expression to the intended facial expression.

After approximately 8 seconds, two possible events will be sent from the EmoEngine™:

`EE_ExpressivTrainingSucceeded`: If the quality of the EEG signal during the training session was sufficiently good to update the Expressiv algorithm's trained signature, the EmoEngine will enter a waiting state to confirm the training update, which will be explained below.

`EE_ExpressivTrainingFailed`: If the quality of the EEG signal during the training session was not good enough to update the trained signature then the Expressiv training process will be reset automatically, and user should be asked to start the training again.

If the training session succeeded (`EE_ExpressivTrainingSucceeded` was received) then the user should be asked whether to accept or reject the session. The user may wish to reject the training session if he feels that he was unable to maintain the desired expression throughout the duration of the training period. The user's response is then submitted to the EmoEngine through the API call `EE_ExpressivSetTrainingControl()` with argument `EXP_ACCEPT` or `EXP_REJECT`. If the training is rejected, then the application should wait until it receives the `EE_ExpressivTrainingRejected` event before restarting the training process. If the training is accepted, EmoEngine™ will rebuild the user's trained Expressiv™ signature, and an `EE_ExpressivTrainingCompleted` event will be sent out once the calibration is done. Note that this signature building process may take up several seconds depending on system resources, the number of expression being trained, and the number of training sessions recorded for each expression.

To run the ExpressivDemo example, launch the Emotiv Control Panel and EmoComposer. In the Emotiv Control Panel select **Connect→To EmoComposer**, accept the default values and then enter a new profile name. Next, navigate to the doc\Examples\example2\blueavatar folder and launch the BlueAvatar application. Enter 30000 as the UDP port and press the **Start Listening** button. Finally, start a new instance of ExpressivDemo, and observe that when you use the **Upperface**, **Lowerface** or **Eye** controls in EmoComposer, the BlueAvatar model responds accordingly.

Next, experiment with the training commands available in ExpressivDemo to better understand the Expressiv training procedure described above. Listing 6 shows a sample ExpressivDemo sessions that demonstrates how to train an expression.

```
Emotiv Engine started!
Type "exit" to quit, "help" to list available commands...
ExpressivDemo>
New user 0 added, sending Expressiv animation to localhost:30000...
ExpressivDemo> trained_sig 0
==> Querying availability of a trained Expressiv signature for user
0...
A trained Expressiv signature is not available for user 0

ExpressivDemo> training_exp 0 neutral
==> Setting Expressiv training expression for user 0 to neutral...

ExpressivDemo> training_start 0
==> Start Expressiv training for user 0...

ExpressivDemo>
Expressiv training for user 0 STARTED!
ExpressivDemo>
Expressiv training for user 0 SUCCEEDED!
ExpressivDemo> training_accept 0
==> Accepting Expressiv training for user 0...

ExpressivDemo>
Expressiv training for user 0 COMPLETED!
ExpressivDemo> training_exp 0 smile
==> Setting Expressiv training expression for user 0 to smile...

ExpressivDemo> training_start 0
==> Start Expressiv training for user 0...

ExpressivDemo>
Expressiv training for user 0 STARTED!
ExpressivDemo>
Expressiv training for user 0 SUCCEEDED!
ExpressivDemo> training_accept 0
==> Accepting Expressiv training for user 0...

ExpressivDemo>
Expressiv training for user 0 COMPLETED!
ExpressivDemo> trained_sig 0
==> Querying availability of a trained Expressiv signature for user
0...
A trained Expressiv signature is  available for user 0
```

```
ExpressivDemo> set_sig 0 1
==> Switching to a trained Expressiv signature for user 0...

ExpressivDemo>
```

Listing 6 *Training "smile" and "neutral" in ExpressivDemo*

5.6 Example 3 – Profile Management

User-specific detection settings, including trained Cognitiv™ and Expressiv™ signature data, currently enabled Cognitiv actions, Cognitiv and Expressiv sensitivity settings, and Affectiv calibration data, are saved in a user profile that can be retrieved from the EmoEngine and restored at a later time.

This example demonstrates the API functions that can be used to manage a user's profile within Emotiv EmoEngine™. Please note that this example requires the Boost C++ Library in order to build correctly. Boost is a modern, open source, peer-reviewed, C++ library with many powerful and useful components for general-purpose, cross-platform development. For more information and detailed instructions on installing the Boost library please visit http://www.boost.org.

```
if (EE_EngineConnect() == EDK_OK) {
    // Allocate an internal structure to hold profile data
    EmoEngineEventHandle eProfile= EE_ProfileEventCreate();
    // Retrieve the base profile and attach it to the eProfile handle
    EE_GetBaseProfile(eProfile);
}
```

Listing 7 *Retrieve the base profile*

`EE_EngineConnect()` or `EE_EngineRemoteConnect()` must be called before manipulating EmoEngine profiles. Profiles are attached to a special kind of event handle that is constructed by calling `EE_ProfileEventCreate()`. After successfully connecting to EmoEngine, a base profile, which contains initial settings for all detections, may be obtained via the API call `EE_GetBaseProfile()`.

This function is not required in order to interact with the EmoEngine profile mechanism – **a new user profile with all appropriate default settings is automatically created when a user connects to EmoEngine and the EE_UserAdded event is generated** - it is, however, useful for certain types of applications that wish to maintain valid profile data for each saved user.

It is much more useful to be able to retrieve the custom settings of an active user. Listing 8 demonstrates how to retrieve this data from EmoEngine.

```
if (EE_GetUserProfile(userID, eProfile) != EDK_OK) {
    // error in arguments...
}
// Determine the size of a buffer to store the user's profile data
unsigned int profileSize;
if (EE_GetUserProfileSize(eProfile, &profileSize) != EDK_OK) {
    // you didn't check the return value above…
}
// Copy the content of profile byte stream into local buffer
```

```
unsigned char* profileBuffer = new unsigned char[profileSize];
int result;
result=EE_GetUserProfileBytes(eProfile, profileBuffer, profileSize);
```

Listing 8 *Get the profile for a particular user*

`EE_GetUserProfile()` is used to get the profile in use for a particular user. This function requires a valid user ID and an `EmoEngineEventHandle` previously obtained via a call to `EE_ProfileEventCreate()`. Once again, the return value should always be checked. If successful, an internal representation of the user's profile will be attached to the `EmoEngineEventHandle` and a serialized, binary representation can be retrieved by using the `EE_GetUserProfileSize()` and `EE_EngineGetUserProfileBytes()` functions, as illustrated above.

The application is then free to manage this binary profile data in the manner that best fits its purpose and operating environment. For example, the application programmer may choose to save it to disk, persist it in a database or attach it to another app-specific data structure that holds its own per-user data.

```
unsigned int profileSize = 0;
unsigned char* profileBuf = NULL;

// assign and populate profileBuf and profileSize correctly
...

if (EE_SetUserProfile(userID, profileBuf, profileSize) != EDK_OK) {
    // error in arguments...
}
```

Listing 9 *Setting a user profile*

`EE_SetUserProfile()` is used to dynamically set the profile for a particular user. In Listing 9, the `profileBuf` is a pointer to the buffer of the binary profile and `profileSize` is an integer storing the number of bytes of the buffer. The binary data can be obtained from the base profile if there is no previously saved profile, or if the application wants to return to the default settings. The return value should always be checked to ensure the request has been made successfully.

```
...
EE_Event_t eventType = EE_EmoEngineEventGetType(eEvent);
EE_EmoEngineEventGetUserId(eEvent, &userID);
switch (eventType) {
    // New Emotiv device connected
    case EE_UserAdded:
         ...
          break;

    // Emotiv device disconnected
    case EE_UserRemoved:
         ...
          break;
```

```
    // Handle EmoState update
    case EE_EmoStateUpdated:
            ...
            break;

    default:
            break;
}
...
```

Listing 10 *Managing profiles*

Examples 1 and 2 focused chiefly on the proper handling of the `EE_EmoStateUpdated` event to accomplish their tasks. Two new event types are required to properly manage EmoEngine profiles in Example 3:

1. `EE_UserAdded`: Whenever a new Emotiv USB receiver is plugged into the computer, EmoEngine will generate an `EE_UserAdded` event. In this case, the application should create a mapping between the Emotiv user ID for the new device and any application-specific user identifier. The Emotiv USB receiver provides 4 LEDs that can be used to display a player number that is assigned by the application. After receiving the `EE_UserAdded` event, the `EE_SetHardwarePlayerDisplay()` function can be called to provide a visual indication of which receiver is being used by each player in a game.

2. `EE_UserRemoved`: When an existing Emotiv USB receiver is removed from the host computer, EmoEngine™ will send an `EE_UserRemoved` event to the application and release internal resources associated with that Emotiv device. The user profile that is coupled with the removed Emotiv EPOC™ will be embedded in the event as well. The developer can retrieve the binary profile using the `EE_GetUserProfileSize()` and `EE_GetUserProfileBytes()` functions as described above. The binary profile can be saved onto disc to decrease memory usage, or kept in the memory to minimize the I/O overhead, and can be reused at a later time if the same user reconnects.

5.7 Example 4 – Cognitiv™ Demo

This example demonstrates how the user's conscious mental intention can be recognized by the Cognitiv™ detection and used to control the movement of a 3D virtual object. It also shows the steps required to train the Cognitiv suite to recognize distinct mental actions for an individual user.

The design of the CognitivDemo application is quite similar to the ExpressivDemo covered in Example 2. In Example 2, ExpressivDemo retrieves EmoStates™ from Emotiv EmoEngine™ and uses the EmoState data describing the user's facial expressions to control an external avatar. In this example, information about the cognitive mental activity of the users is extracted instead. The output of the Cognitiv detection indicates whether users are mentally engaged in one of the trained Cognitiv actions (pushing, lifting, rotating, etc.) at any given time. Based on the Cognitiv results, corresponding commands are sent to a separate application called `EmoCube` to control the movement of a 3D cube.

Commands are communicated to EmoCube via a UDP network connection. As in Example 2, the network protocol is very simple: an action is communicated as two comma-separated, ASCII-formatted values. The first is the action type returned by

`ES_CognitivGetCurrentAction()`, and the other is the action power returned by `ES_CognitivGetCurrentActionPower()`, as shown in Listing 11.

```
void sendCognitivAnimation(SocketClient& sock, EmoStateHandle eState)
{
    std::ostringstream os;

    EE_CognitivAction_t actionType;
    actionType = ES_CognitivGetCurrentAction(eState);
    float actionPower;
    actionPower = ES_CognitivGetCurrentActionPower(eState);

    os << static_cast<int>(actionType) << ","
       << static_cast<int>(actionPower*100.0f);
    sock.SendBytes(os.str());
}
```

Listing 11   *Querying EmoState for Cognitiv detection results*

5.7.1 Training for Cognitiv

The Cognitiv detection suite requires a training process in order to recognize when a user is consciously imagining or visualizing one of the supported Cognitiv actions. Unlike the Expressiv suite, there is no universal signature that will work well across multiple individuals. An application creates a trained Cognitiv signature for an individual user by calling the appropriate Cognitiv API functions and correctly handling appropriate EmoEngine events. The training protocol is very similar to that described in Example 2 in order to create a trained signature for Expressiv.

To better understand the API calling sequence, an explanation of the Cognitiv detection is required. As with Expressiv, it will be useful to first familiarize yourself with the operation of the Cognitiv tab in Emotiv Control Panel before attempting to use the Cognitiv API functions.

Cognitiv can be configured to recognize and distinguish between up to 4 distinct actions at a given time. New users typically require practice in order to reliably evoke and switch between the mental states used for training each Cognitiv action. As such, it is imperative that a user first masters a single action before enabling two concurrent actions, two actions before three, and so forth.

During the training update process, it is important to maintain the quality of the EEG signal and the consistency of the mental imagery associated with the action being trained. Users should refrain from moving and should relax their face and neck in order to limit other potential sources of interference with their EEG signal.

Unlike Expressiv, the Cognitiv algorithm does not include a delay after receiving the `COG_START` training command before it starts recording new training data.
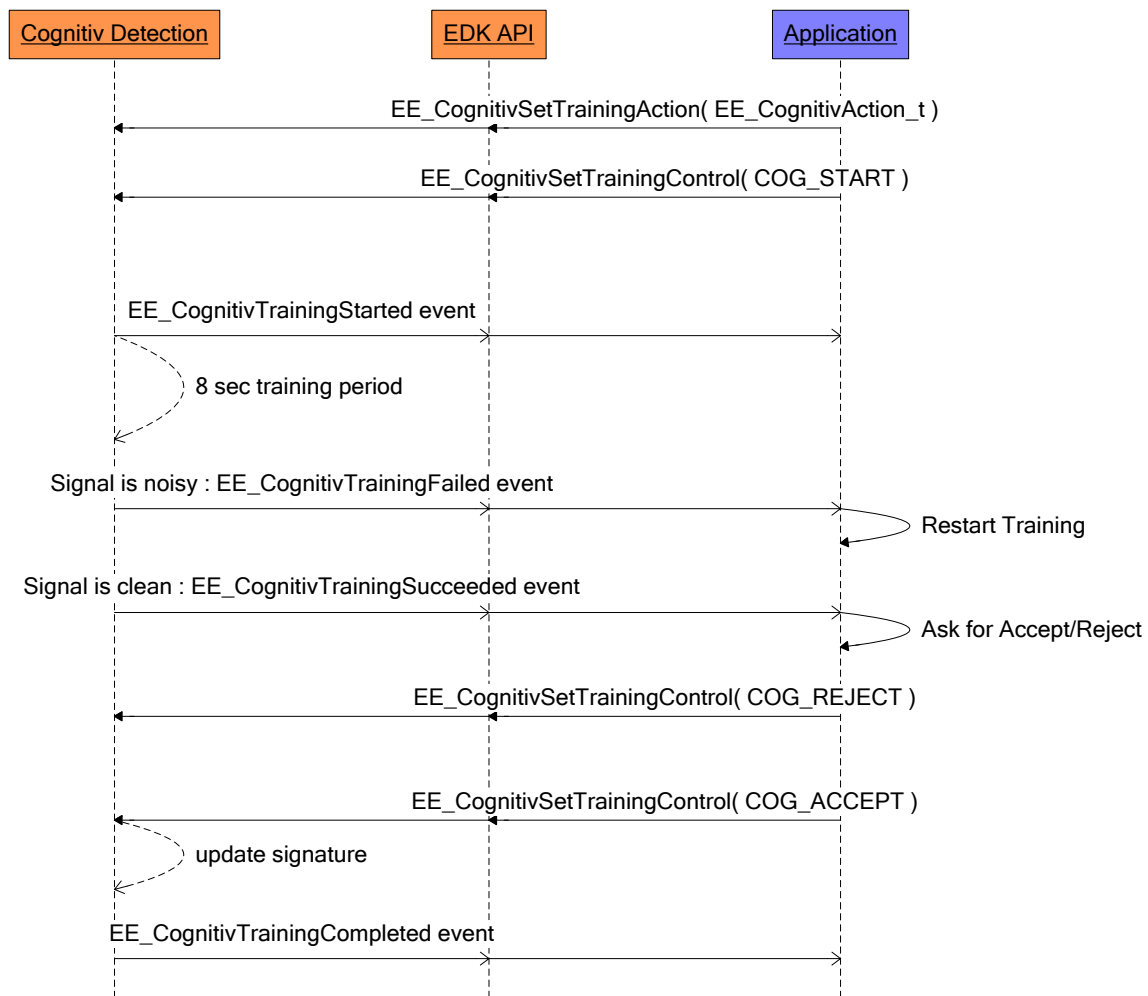
Figure 33 *Cognitiv training*

The above sequence diagram describes the process of carrying out Cognitiv training on a particular action. The Cognitiv-specific events are declared as enumerated type `EE_CognitivEvent_t` in `EDK.h`. Note that this type differs from the `EE_Event_t` type used by top-level EmoEngine Events. The code snippet in Listing 12 illustrates the procedure for extracting Cognitiv-specific event information from the EmoEngine event.

```
EmoEngineEventHandle eEvent= EE_EmoEngineEventCreate();
if (EE_EngineGetNextEvent(eEvent) == EDK_OK) {
    EE_Event_t eventType = EE_EmoEngineEventGetType(eEvent);
    if (eventType == EE_CognitivEvent) {
        EE_CognitivEvent_t cEvt = EE_CognitivEventGetType(eEvent);
        ...
}
```

Listing 12 *Extracting Cognitiv event details*

Before the start of a training session, the action type must be first set with the API function `EE_CognitivSetTrainingAction()`. In `EmoStateDLL.h`, the enumerated type `EE_CognitivAction_t` defines all the Cognitiv actions that are currently supported (`COG_PUSH`, `COG_LIFT`, etc.). If an action is not set before the start of training, `COG_NEUTRAL` will be used as the default.

`EE_CognitivSetTrainingControl()` can then be called with argument `COG_START` to start the training on the target action. In `EDK.h`, enumerated type `EE_CognitivTrainingControl_t` defines the control command constants for Cognitiv training. If the training can be started, an `EE_CognitivTrainingStarted` event will be sent almost immediately. The user should be prompted to visualize or imagine the appropriate action prior to sending the `COG_START command`. The training update will begin after the EmoEngine sends the `EE_CognitivTrainingStarted` event. This delay will help to avoid training with undesirable EEG artifacts resulting from transitioning from a "neutral" mental state to the desired mental action state.

After approximately 8 seconds, two possible events will be sent from the EmoEngine™:

`EE_CognitivTrainingSucceeded`: If the quality of the EEG signal during the training session was sufficiently good to update the algorithms trained signature, EmoEngine™ will enter a waiting state to confirm the training update, which will be explained below.

`EE_CognitivTrainingFailed`: If the quality of the EEG signal during the training session was not good enough to update the trained signature then the Cognitiv™ training process will be reset automatically, and user should be asked to start the training again.

If the training session succeeded (`EE_CognitivTrainingSucceeded` was received) then the user should be asked whether to accept or reject the session. The user may wish to reject the training session if he feels that he was unable to evoke or maintain a consistent mental state for the entire duration of the training period. The user's response is then submitted to the EmoEngine through the API call `EE_CognitivSetTrainingControl()` with argument `COG_ACCEPT` or `COG_REJECT`. If the training is rejected, then the application should wait until it receives the `EE_CognitivTrainingRejected` event before restarting the training process. If the training is accepted, EmoEngine™ will rebuild the user's trained Cognitiv™ signature, and an `EE_CognitivTrainingCompleted` event will be sent out once the calibration is done. Note that this signature building process may take up several seconds depending on system resources, the number of actions being trained, and the number of training sessions recorded for each action.

To test the example, launch the Emotiv Control Panel and the EmoComposer. In the Emotiv Control Panel select **Connect→To EmoComposer** and accept the default values and then enter a new profile name. Navigate to the \example4\EmoCube folder and launch the EmoCube, enter 20000 as the UDP port and select **Start Server**. Start a new instance of **CognitivDemo**, and observe that when you use the Cognitiv control in the EmoComposer the EmoCube responds accordingly.

Next, experiment with the training commands available in CognitivDemo to better understand the Cognitiv training procedure described above. Listing 13 shows a sample CognitivDemo session that demonstrates how to train.

```
CognitivDemo> set_actions 0 push lift
==> Setting Cognitiv active actions for user 0...

CognitivDemo>
Cognitiv signature for user 0 UPDATED!
CognitivDemo> training_action 0 push
==> Setting Cognitiv training action for user 0 to "push"...

CognitivDemo> training_start 0
==> Start Cognitiv training for user 0...

CognitivDemo>
Cognitiv training for user 0 STARTED!
CognitivDemo>
Cognitiv training for user 0 SUCCEEDED!
CognitivDemo> training_accept 0
==> Accepting Cognitiv training for user 0...

CognitivDemo>
Cognitiv training for user 0 COMPLETED!
CognitivDemo> training_action 0 neutral
==> Setting Cognitiv training action for user 0 to "neutral"...

CognitivDemo> training_start 0
==> Start Cognitiv training for user 0...

CognitivDemo>
Cognitiv training for user 0 STARTED!
CognitivDemo>
Cognitiv training for user 0 SUCCEEDED!
CognitivDemo> training_accept 0
==> Accepting Cognitiv training for user 0...

CognitivDemo>
Cognitiv training for user 0 COMPLETED!
CognitivDemo>
```

Listing 13 *Training "push" and "neutral" with CognitivDemo*

5.8 Example 5 – EEG Logger Demo

This example demonstrates how to extract live EEG data using the EmoEngineTM in C++. Data is read from the headset and sent to an output file for later analysis. ***Please note that this examples only works with the SDK versions that allow raw EEG access (Research, Education and Enterprise Plus).***

The example starts in the same manner as the earlier examples (see Listing 1 & 2, Section 5.4). A connection is made to the EmoEngine through a call to EE_EngineConnect(), or to EmoComposer through a call to EE_EngineRemoteConnect(). The EmoEngine event handlers and EmoState Buffer's are also created as before.

```
        float secs                              = 1;
        …
            DataHandle hData = EE_DataCreate();
        DataSetBufferSizeInSec(secs);

        std::cout << "Buffer size in secs:" << secs << std::endl;
        …==> Setting Cognitiv active actions for user 0...

```

Listing 14 *Access to EEG data*

Access to EEG measurements requires the creation of a DataHandle, a handle that is used to provide access to the underlying data. This handle is initialized with a call to EE_DataCreate(). During the measurement process, EmoEngine will maintain a data buffer of sampled data, measured in seconds. This data buffer must be initialized with a call to DataSetBufferSizeInSec(…), prior to collecting any data.

```
    while (…)

        state = EE_EngineGetNextEvent(eEvent);

        if (state == EDK_OK) {

            EE_Event_t eventType =
EE_EmoEngineEventGetType(eEvent);
            EE_EmoEngineEventGetUserId(eEvent, &userID);

                // Log the EmoState if it has been updated

                if (eventType == EE_UserAdded) {

                    EE_DataAcquisitionEnable(userID,true);
                    readytocollect = true;
                }
        }
```

Listing 15 *Start Acquiring Data*

When the connection to EmoEngine is first made via EE_EngineConnect(), the engine will not have registered a valid user. The trigger for this registration is an EE_UserAdded event, which is raised shortly after the connection is made. Once the user is registered, it is possible to enable data acquisition via a call to DataAcquisitionEnable.   With this enabled, EmoEngine will start collecting EEG for the user, storing it in the internal EmoEngine sample buffer. Note that the developer's application should access the EEG data at a rate that will ensure the sample buffer is not overrun.

```
    if (readytocollect)

…

        DataUpdateHandle (0, hData);

        unsigned int nSamplesTaken=0;
        DataGetNumberOfSample(hData,&nSamplesTaken);
```

```
            if (nSamplesTaken != 0)
            …
                    double* data = new double[nSamplesTaken];
                    EE_DataGet(hData, targetChannelList[i], data,
nSamplesTaken);
                    delete[] data;
```

Listing 16 *Acquiring Data*

To initiate retrieval of the latest EEG buffered data, a call is made to DataUpdateHandle(). When this function is processed, EmoEngine will ready the latest buffered data for access via the hData handle. All data captured since the last call to DataUpdateHandle will be retrieved. Place a call to DataGetNumberOfSample() to establish how much buffered data is currently available. The number of samples can be used to set up a buffer for retrieval into your application as shown.

Finally, to transfer the data into a buffer in our application, we call the EE_DataGet function. To retrieve the buffer we need to choose from one of the available data channels:

ED_COUNTER,ED_AF3, ED_F7, ED_F3, ED_FC5, ED_T7,

ED_P7, ED_O1, ED_O2, ED_P8, ED_T8, ED_FC6, ED_F4,

ED_F8, ED_AF4, ED_GYROX, ED_GYROY, ED_TIMESTAMP,

ED_FUNC_ID, ED_FUNC_VALUE, ED_MARKER, ED_SYNC_SIGNAL

For example, to retrieve the first sample of data held in the sensor AF3, place a call to EE_DataGet as follows:

EE_DataGet(hData, ED_AF3, databuffer, 1);

You may retrieve all the samples held in the buffer using the bufferSizeInSample parameter.

Finally, we need to ensure correct clean up by disconnecting from the EmoEngine and free all associated memory.

EE_EngineDisconnect();

EE_EmoStateFree(eState);

EE_EmoEngineEventFree(eEvent);

5.9 Example 6 – Affectiv Demo

Affectiv Demo allows log score of Affectiv(including raw score and scaled score) in csv file format.

The program runs with command line syntax: EmoStateLogger [log_file_name], log_file_name is set by the user.

```
        if (argc != 2)
        {
        throw std::exception("Please supply the log file name.\nUsage:
        EmoStateLogger [log_file_name].");
        }
```

Listing 17   Creat log_file_name

The example starts in the same manner as the earlier examples (see Listing 1 & 2, Section 5.4). A connection is made to the EmoEngine through a call to EE_EngineConnect(), or to EmoComposer through a call to EE_EngineRemoteConnect().

```
std::cout << "==================================================" << std::endl;
std::cout << "Example to show how to log the EmoState from EmoEngine/EmoComposer."
<< std::endl;
std::cout << "==================================================" << std::endl;
std::cout << "Press '1' to start and connect to the EmoEngine      " << std::endl;
std::cout << "Press '2' to connect to the EmoComposer              " << std::endl;
std::cout << ">> ";
std::getline(std::cin, input, '\n');
option = atoi(input.c_str());

switch (option) {
        case 1:
            {
            if (EE_EngineConnect() != EDK_OK) {
                throw std::exception("Emotiv Engine start up failed.");
            }
            break;
            }
        case 2:
            {
              std::cout << "Target IP of EmoComposer? [127.0.0.1] ";
              std::getline(std::cin, input, '\n');
              if (input.empty()) {
                                input = std::string("127.0.0.1");
                                }
              if (EE_EngineRemoteConnect(input.c_str(), composerPort) != EDK_OK)
                {
                        std::string errMsg = "Cannot connect to EmoComposer on ["
                + input + "]";
                    throw std::exception(errMsg.c_str());
                    }
                    break;
                    }
        default:
                throw std::exception("Invalid option...");
                break;
                }


std::cout << "Start receiving Affectiv Score! Press any key to stop logging...\n"
```

```cpp
<< std::endl;
std::ofstream ofs(argv[1]);
//std::ofstream ofs("test.csv");
bool writeHeader = true;

while (!_kbhit()) {

        state = EE_EngineGetNextEvent(eEvent);

        // New event needs to be handled
        if (state == EDK_OK) {
                EE_Event_t eventType = EE_EmoEngineEventGetType(eEvent);
                EE_EmoEngineEventGetUserId(eEvent, &userID);

        // Log the EmoState if it has been updated
        if (eventType == EE_EmoStateUpdated) {

                EE_EmoEngineEventGetEmoState(eEvent, eState);
                const float timestamp = ES_GetTimeFromStart(eState);
                printf("%10.3fs : New Affectiv score from user %d ...\r", timestamp,
                userID);

                logAffectivScore(ofs, userID, eState, writeHeader);
                writeHeader = false;
                }
                }
        else if (state != EDK_NO_EVENT) {
                std::cout << "Internal error in Emotiv Engine!" << std::endl;
        break;
                }

        Sleep(1);
                }

        ofs.close();
        }
        catch (const std::exception& e) {
                std::cerr << e.what() << std::endl;
                std::cout << "Press any key to exit..." << std::endl;
                getchar();
        }
```

Listing 18 *Connect to EmoEngine and EmoComposer*

Log file log.csv has columns as time (time from the beginning of the log), user id, raw score, min, max, scaled score of the Affective (frustration, engagement, mediation, excitement

```cpp
        // Create the top header
        if (withHeader) {
                os << "Time,";
                os << "UserID,";
                os << "Frustration raw score,";
```

```cpp
                    os << "Frustration min score,";
                    os << "Frustration max score,";
                    os << "Frustration scaled score,";
                    os << "Engagement boredom raw score,";
                    os << "Engagement boredom min score,";
                    os << "Engagement boredom max score,";
                    os << "Engagement boredom scaled score,";
                    os << "Meditation raw score,";
                    os << "Meditation min score,";
                    os << "Meditation max score,";
                    os << "Meditation scaled score,";
                    os << "Excitement raw score,";
                    os << "Excitement min score,";
                    os << "Excitement max score,";
                    os << "Excitement scaled score,";
                    os << std::endl;
            }
            // Log the time stamp and user ID
            os << ES_GetTimeFromStart(eState) << ",";
            os << userID << ",";
            // Affectiv results
            double rawScore=0;
            double minScale=0;
            double maxScale=0;
            double scaledScore=0;
            ES_AffectivGetFrustrationModelParams(eState,&rawScore,&minScale,&maxScale
);
            os << rawScore << ",";
            os << minScale << ",";
            os << maxScale << ",";
            if (minScale==maxScale)
            {
                    os << "undefined" << ",";
            }
            else{
                    CaculateScale(rawScore,maxScale, minScale,scaledScore);
                    os << scaledScore << ",";
            }
            ES_AffectivGetEngagementBoredomModelParams(eState,&rawScore,&minScale,&ma
xScale);
            os << rawScore << ",";
            os << minScale << ",";
            os << maxScale << ",";
            if (minScale==maxScale)
            {
                    os << "undefined" << ",";
            }
            else{
                    CaculateScale(rawScore,maxScale, minScale,scaledScore);
                    os << scaledScore << ",";
            }
            ES_AffectivGetMeditationModelParams(eState,&rawScore,&minScale,&maxScale)
;
            os << rawScore << ",";
            os << minScale << ",";
            os << maxScale << ",";
            if (minScale==maxScale)
            {
```

```cpp
            os << "undefined" << ",";
        }
        else{
            CaculateScale(rawScore,maxScale, minScale,scaledScore);
            os << scaledScore << ",";
        }
        ES_AffectivGetExcitementShortTermModelParams(eState,&rawScore,&minScale,&
maxScale);
        os << rawScore << ",";
        os << minScale << ",";
        os << maxScale << ",";
        if (minScale==maxScale)
        {
            os << "undefined" << ",";
        }
        else{
            CaculateScale(rawScore,maxScale, minScale,scaledScore);
            os << scaledScore << ",";
        }
        os << std::endl;
}
void CaculateScale (double& rawScore, double& maxScale, double& minScale,
double& scaledScore){

        if (rawScore<minScale)
        {
            scaledScore =0;
        }else if (rawScore>maxScale)
        {
            scaledScore = 1;
        }
        else{
            scaledScore = (rawScore-minScale)/(maxScale-minScale);
        }
}
```

Listing 19   *Log score to csv file*

Finally, we need to ensure correct clean up by disconnecting from the EmoEngine and free all associated memory.

EE_EngineDisconnect();

EE_EmoStateFree(eState);

EE_EmoEngineEventFree(eEvent);

5.10 Example 7 – EmoState and EEGLogger

This example demonstrates the use of the core Emotiv API functions described in Sections 5.2 and 5.3. It logs all Emotiv detection results for the attached users after successfully establishing a connection to Emotiv EmoEngine™ or EmoComposer™.

Please note that this examples only works with the SDK versions that allow raw EEG access (Research, Education and Enterprise Plus).

The data is recorded in EEG_Data.csv files and Affectiv_Data.csv, they put in the folder .. \ bin \.

```cpp
std::cout << "Start receiving EEG Data and affectiv data! Press any key to stop
logging...\n" << std::endl;
std::ofstream ofs("../bin/EEG_Data.csv",std::ios::trunc);
ofs << header << std::endl;
std::ofstream ofs2("../bin/Affectiv_Data.csv",std::ios::trunc);
ofs2 << affectivSuitesName << std::endl;

DataHandle hData = EE_DataCreate();
EE_DataSetBufferSizeInSec(secs);

std::cout << "Buffer size in secs:" << secs << std::endl;
```

Listing 20   *Log score to EEG_Data.csv and Affectiv_Data.csv*

EEG_Data.csv file stores channels : ED_COUNTER,
 ED_AF3, ED_F7, ED_F3, ED_FC5, ED_T7,
 ED_P7, ED_O1, ED_O2, ED_P8, ED_T8,
 ED_FC6, ED_F4, ED_F8, ED_AF4, ED_GYROX, ED_GYROY, ED_TIMESTAMP,
 ED_FUNC_ID, ED_FUNC_VALUE, ED_MARKER, ED_SYNC_SIGNAL

Affectiv_Data.csv file stores : Engagement, Frustration, Meditation, Excitement

```cpp
while (!_kbhit()) {

        state = EE_EngineGetNextEvent(eEvent);
        EE_Event_t eventType;

        if (state == EDK_OK) {

                eventType = EE_EmoEngineEventGetType(eEvent);
                EE_EmoEngineEventGetUserId(eEvent, &userID);
                EE_EmoEngineEventGetEmoState(eEvent, eState);
        // Log the EmoState if it has been updated
        if (eventType == EE_UserAdded) {
                std::cout << "User added";
                EE_DataAcquisitionEnable(userID,true);
                readytocollect = true;
                }


        if (readytocollect && (eventType == EE_EmoStateUpdated)) {

                EE_DataUpdateHandle(0, hData);

                unsigned int nSamplesTaken=0;
                EE_DataGetNumberOfSample(hData,&nSamplesTaken);
                std::cout << "Updated " << nSamplesTaken << std::endl;
```

```
        if (nSamplesTaken != 0  ) {

                double* data = new double[nSamplesTaken];
                for(int sampleIdx=0 ; sampleIdx<(int)nSamplesTaken ; ++ sampleIdx)
                {
                for (int i = 0 ; <sizeof(targetChannelList)/sizeof(EE_DataChannel_t)
                ; i++) {

                EE_DataGet(hData, targetChannelList[i], data, nSamplesTaken);
                ofs << data[sampleIdx] << ",";
                        }
                ofs << std::endl;
                }
                delete[] data;
                }

float affEngegement = ES_AffectivGetEngagementBoredomScore(eState);
float affFrus = ES_AffectivGetFrustrationScore(eState);
float affMed = ES_AffectivGetMeditationScore(eState);
float affExcitement = ES_AffectivGetExcitementShortTermScore(eState);
printf("Engagement: %f, Frustration: %f, ...\n",affEngegement,affFrus);
ofs2<<affEngegement<<","<<affFrus<<","<<affMed<<","<<affExcitement<<","<<std::endl
;
}
}
Sleep(100);
}

ofs.close();
ofs2.close();
EE_DataFree(hData);

}
```

Listing 21 *Write data channels and score*

Before the end of the program, `EE_EngineDisconnect()` is called to terminate the connection with the EmoEngine and free up resources associated with the connection. The user should also call `EE_EmoStateFree()` and `EE_EmoEngineEventFree()` to free up memory allocated for the EmoState buffer and `EmoEngineEventHandle`

5.11 Example 8 – Gyro Data

Gyro data example allows built-in 2-axis gyroscope position.

Simply turn your head from left to right, up and down. You will also notice the red indicator dot move in accordance with the movement of your head/gyroscope.

```
void display(void)
{
    glClear(GL_COLOR_BUFFER_BIT);
    glPushMatrix();
```

```
    glColor3f(1.0,1.0,1.0);
    drawCircle(800,100);
    glColor3f(0.0,0.0,1.0);
    drawCircle(maxRadius-4000,800);
    glColor3f(0.0,1.0,1.0);
    drawCircle(maxRadius,1000);


    glColor3f(1.0, 0.0, 0.0);
    glRectf(currX-400.0, currY-400.0, currX+400.0, currY+400.0);

    glPopMatrix();
    glutSwapBuffers();
}

void changeXY(int x) // x = 0 : idle
{
        if( currX >0 )
        {
                float temp = currY/currX;
                currX -= incOrDec;
                currY = temp*currX;
        }
        else if( currX < 0)
        {
                float temp = currY/currX;
                currX += incOrDec;
                currY = temp*currX;
        }
        else
        {
                if( currY > 0 ) currY -= incOrDec;
                else if( currY <0 ) currY += incOrDec;
        }
        if( x == 0)
        {
                if( (abs(currX) <= incOrDec) && (abs(currY) <= incOrDec))
                {
                        xmax = 0;
                        ymax = 0;
                }
                else
                {
                        xmax = currX;
                        ymax = currY;
                }
        }
        else
        {
                if( (abs(currX) <= incOrDec) && (abs(currY) <= incOrDec))
                {
                        xmax = 0;
                        ymax = 0;
                }
        }
}
```

```cpp
void updateDisplay(void)
{
    int gyroX = 0,gyroY = 0;
    EE_HeadsetGetGyroDelta(0,&gyroX,&gyroY);
    xmax += gyroX;
    ymax += gyroY;

    if( outOfBound )
    {
            if( preX != gyroX && preY != gyroY )
            {
                    xmax = currX;
                    ymax = currY;
            }
    }

    double val = sqrt((float)(xmax*xmax + ymax*ymax));

     std::cout <<"xmax : " << xmax <<" ; ymax : " << ymax << std::endl;


    if( val >= maxRadius )
    {
            changeXY(1);
            outOfBound = true;
            preX = gyroX;
            preY = gyroY;
    }
    else
    {
            outOfBound = false;
                if(oldXVal == gyroX && oldYVal == gyroY)
                {
                        ++count;
                        if( count > 10 )
                        {
                                changeXY(0);
                        }
                }
                else
                {
                        count = 0;
                        currX = xmax;
                        currY = ymax;
                        oldXVal = gyroX;
                        oldYVal = gyroY;
                }
    }
    Sleep(15);
    glutPostRedisplay();
}
void reshape(int w, int h)
{
    glViewport (0, 0, (GLsizei) w, (GLsizei) h);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glOrtho(-50000.0, 50000.0, -50000.0, 50000.0, -1.0, 1.0);
```

```c
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
}
void mouse(int button, int state, int x, int y)
{
    switch (button) {
        case GLUT_LEFT_BUTTON:
            if (state == GLUT_DOWN)
                glutIdleFunc(updateDisplay);
            break;
        case GLUT_MIDDLE_BUTTON:
            if (state == GLUT_DOWN)
                glutIdleFunc(NULL);
            break;
        default:
            break;
    }
}
/*
 *  Request double buffer display mode.
 *  Register mouse input callback functions
 */
int main(int argc, char** argv)
{
    EmoEngineEventHandle hEvent = EE_EmoEngineEventCreate();
    EmoStateHandle eState = EE_EmoStateCreate();
    unsigned int userID = -1;
    EE_EngineConnect();
    if(oneTime)
    {
        printf("Start after 8 seconds\n");
        Sleep(8000);
        oneTime = false;
    }

    globalElapsed = GetTickCount();

    glutInit(&argc, argv);
    glutInitDisplayMode (GLUT_DOUBLE | GLUT_RGB);
    glutInitWindowSize (650, 650);
    glutInitWindowPosition (100, 100);
    glutCreateWindow (argv[0]);
    init ();
    glutDisplayFunc(display);
    glutReshapeFunc(reshape);
    glutIdleFunc(updateDisplay);
    glutMainLoop();

```

Listing 22   *Gyro Data*

Before the end of the program, EE_EngineDisconnect() is called to terminate the connection with the EmoEngine and free up resources associated with the connection. The user should also call EE_EmoStateFree() and EE_EmoEngineEventFree() to free up memory allocated for the EmoState buffer and EmoEngineEventHandle

5.12 Example 9 – Multi Dongle Connection

This example captures event when you plug or unplug dongle .

Every time you plug or unplug dongle, there is a notice that dongle ID is added or removed

```cpp
int main(int argc,char** argv[])
{
    EmoEngineEventHandle hEvent = EE_EmoEngineEventCreate();
    EmoStateHandle eState = EE_EmoStateCreate();
    unsigned int userID = -1;
    list<int> listUser;

    if( EE_EngineConnect() == EDK_OK )
    {
        while(!_kbhit())
        {
        int state = EE_EngineGetNextEvent(hEvent);
        if( state == EDK_OK )
        {
            EE_Event_t eventType = EE_EmoEngineEventGetType(hEvent);

            EE_EmoEngineEventGetUserId(hEvent, &userID);
            if(userID==-1)
                continue;

            if(eventType == EE_EmoStateUpdated  )
            {
            // Copies an EmoState returned with a EE_EmoStateUpdate event
            // to memory referenced by an EmoStateHandle.
            if(EE_EmoEngineEventGetEmoState(hEvent,eState)==EDK_OK)
            {

                if(EE_GetUserProfile(userID,hEvent)==EDK_OK)
                {
                //Affective score, short term excitement

                cout <<"userID: " << userID  <<endl;
                cout <<"    affectiv excitement score: " <<
            ES_AffectivGetExcitementShortTermScore (eState) << endl;
                cout <<"    expressiv smile extent : " <<
                ES_ExpressivGetSmileExtent(eState) <<endl;


                    }


                }

            }
            // userremoved event
            else if( eventType == EE_UserRemoved )
            {
                cout <<"user ID: "<<userID<<" have removed" <<
    endl;
                listUser.remove(userID);
            }
```

```
                         // useradded event
                         else if(eventType == EE_UserAdded)
                         {
                                 listUser.push_back(userID);
                                 cout <<"user ID: "<<userID<<" have added" <<
     endl;
                         }
                         userID=-1;
                 }
            }
     }
```

Listing 23   *Multi Dongle Connection*

Before the end of the program, EE_EngineDisconnect() is called to terminate the connection with the EmoEngine and free up resources associated with the connection. The user should also call EE_EmoStateFree() and EE_EmoEngineEventFree() to free up memory allocated for the EmoState buffer and EmoEngineEventHandle

5.13 Example 10 – Multi Dongle EEGLogger

This example logs EEG data from two headset to data1.csv và data2.csv file in folder "..\bin\".

```
// Create some structures to hold the data
      EmoEngineEventHandle eEvent = EE_EmoEngineEventCreate();
      EmoStateHandle eState = EE_EmoStateCreate();

      std::ofstream ofs1("../bin/data1.csv",std::ios::trunc);
      ofs1 << header << std::endl;
      std::ofstream ofs2("../bin/data2.csv",std::ios::trunc);
      ofs2 << header << std::endl;
```

Listing 24   *Creat data1.csv and data2.csv for Multi Dongle EEGLogger*

Please note that this examples only works with the SDK versions that allow raw EEG access (Research, Education and Enterprise Plus).

Data1.csv or data2.csv file stores channels : ED_COUNTER,
            ED_AF3, ED_F7, ED_F3, ED_FC5, ED_T7,
            ED_P7, ED_O1, ED_O2, ED_P8, ED_T8,
            ED_FC6, ED_F4, ED_F8, ED_AF4, ED_GYROX, ED_GYROY, ED_TIMESTAMP,
            ED_FUNC_ID, ED_FUNC_VALUE, ED_MARKER, ED_SYNC_SIGNAL

```
// Make sure we're connect
if( EE_EngineConnect() == EDK_OK )
{
```

```cpp
// Create the data holder
        DataHandle eData = EE_DataCreate();
        EE_DataSetBufferSizeInSec(secs);

// Let them know about it
        std::cout << "Buffer size in secs:" << secs << std::endl;

// How many samples per file?
        int samples_per_file = 384;        // 3 seconds
// Presumably this will fail when we no longer
//      receive data...
        while(!_kbhit())
                {
                // Grab the next event.
                // We seem to mainly care about user adds and removes
                int state = EE_EngineGetNextEvent(eEvent);
                if( state == EDK_OK )
                {
                // Grab some info about the event
                EE_Event_t eventType = EE_EmoEngineEventGetType(eEvent); // same

                EE_EmoEngineEventGetUserId(eEvent, &userID); // same

                // Do nothing if no user...
                if(userID==-1) {
                        continue;
                                }


                // Add the user to the list, if necessary
                if (eventType == EE_UserAdded)
                {
                        std::cout << "User added: " << userID << endl;
                        EE_DataAcquisitionEnable(userID,true);
                        userList[numUsers++] = userID;

                // Check
                if (numUsers > 2)
                {
                        throw std::exception("Too many users!");
                }
                }
                else if (eventType == EE_UserRemoved)
                {
                        cout << "User removed: " << userID << endl;
                        if (userList[0] == userID)
                                {
                                userList[0] = userList[1];
                                userList[1] = -1;
                                numUsers--;
                                }
                        else if (userList[1] == userID)
                                {
                                userList[1] = -1;
                                numUsers--;
                                        }
                                }
```

```cpp
                         // Might be ready to get going.
                         if (numUsers == 2) {
                                 readytocollect = true;
                         }
                         else {
                                 readytocollect = false;
                         }
                         }

                 //EE_DataUpdateHandle(userID, eData);

                 // If we've got both, then start collecting
                 if (readytocollect && (state==EDK_OK))
                 {
                 int check = EE_DataUpdateHandle(userID, eData);
                 unsigned int nSamplesTaken=0;
                 EE_DataGetNumberOfSample(eData,&nSamplesTaken);

                 if( userID == 0 )
                 {
                         if( nSamplesTaken != 0)
                         {
                          IsHeadset1On = true;
                          if( onetime) { write = userID; onetime = false; }
                          for (int c = 0 ; c < sizeof(targetChannelList)
                          /sizeof(EE_DataChannel_t) ; c++)
                          {
                          data1[c] = new double[nSamplesTaken];
                         EE_DataGet(eData, targetChannelList[c], data1[c],
                                 nSamplesTaken);
                          numberOfSample1 = nSamplesTaken;
                         }
                         }
                 else IsHeadset1On = false;
                         }

                 if( userID == 1  )
                 {
                         if(nSamplesTaken != 0)
                         {
                          IsHeadset2On = true;
                          if( onetime) { write = userID; onetime = false; }
                          for (int c = 0 ; c < sizeof(targetChannelList)/
                         sizeof(EE_DataChannel_t) ; c++)
                          {
                          data2[c] = new double[nSamplesTaken];
                          EE_DataGet(eData, targetChannelList[c], data2[c],
                         nSamplesTaken);
                          numberOfSample2 = nSamplesTaken;
                         }
                         }
                 else
                         IsHeadset2On = false;

                         }

         if( IsHeadset1On && IsHeadset2On)
             {
```

```
            cout <<"Update " << 0 <<" : " << numberOfSample1 << endl;
            for (int c = 0 ; c < numberOfSample1  ; c++)
            {
             for (int i = 0 ; i<sizeof(targetChannelList)/
                    sizeof(EE_DataChannel_t) ; i++)
            {
                    ofs1 << data1[i][c] <<",";
            }
                    ofs1 << std::endl;
            //delete data1[c];
            }
            cout <<"Update " << 1 <<" : " << numberOfSample2 << endl;
            for (int c = 0 ; c < numberOfSample2  ; c++)
            {
             for (int i = 0 ; i<sizeof(targetChannelList)/
                    sizeof(EE_DataChannel_t) ; i++)
            {
                    ofs2 << data2[i][c] << ",";
            }
                    ofs2 << std::endl;
            //delete[] data2[c];
            }

            // Don't overload */
            //Sleep(100);
            IsHeadset1On = false;
            IsHeadset2On = false;
                }
            }
        }
    ofs1.close();
    ofs2.close();
```

Listing 25 *Write data1.csv and data2.csv file*

Finally, we need to ensure correct clean up by disconnecting from the EmoEngine and free all associated memory.

EE_EngineDisconnect();

EE_EmoStateFree(eState);

EE_EmoEngineEventFree(eEvent);

5.14 DotNetEmotivSDK Test

The Emotiv SDK comes with C# support. The wrapper is provided at \doc\examples\DotNet\DotNetEmotivSDK.   The   test   project   at

\doc\examples_DotNet\DotNetEmotivSDKTest demonstrates how programmers can interface with the Emotiv SDK via the C# wrapper.

It's highly recommended that developers taking advantage of this test project read through other sections of these chapters. Concepts about the EmoEngine, the EmoEvents and EmoState are the same. DotNetEmotivSDK is merely a C# wrapper for the native C++ Emotiv SDK.

6. Troubleshooting

Transceiver Dongle lights don't work (Should see either one slow flashing LED – if unpaired – or one bright and one dim LED – if paired).

- Try different USB port, different computer.

- Check other USB equipment works in same port

- Possible LED failure – carry on regardless

Transceiver Dongle not recognized (USB "ding" on insertion, stream of Hardware Identifier strings observed on first installation)

- Use only on XP, Window Vista and Windows 7 and virtual Windows XP/Vista/7 machines in Mac OS X

- Check Device Manager list as Dongle is inserted and removed.

- Try different USB port, different computer.

- If Control Panel is running it will ask for User Profile selection when the Transceiver Dongle is recognized, whether the neuroheadset is paired or not

Does not pair (Transceiver Dongle switches from single slow flashing LED to one bright & one dim LED if correctly paired)

- Check for blue LED on neuroheadset (recharge neuroheadset if absent).

- Hold neuroheadset very close to USB, switch off and on.

- Unplug and replace Transceiver Dongle in USB port, repeat pairing attempt

- Obtain a USB male - USB female extension cable and plug into the PC. Attach the Transceiver Dongle to the extension cable and position in a prominent location away from your PC, monitor, wireless router and other sources of radio-frequency interference.

- Turn off or disconnect other wireless and Bluetooth devices in the area to isolate possible causes

Weak wireless connection or repeated drop-outs (expected range 3-5 metres within line of sight)

- Move closer to the Transceiver Dongle

- Obtain a USB male - USB female extension cable and plug into the PC. Attach the Transceiver Dongle to the extension cable and position in a prominent location away from your PC, monitor, wireless router and other sources of radio-frequency interference.

- Turn off or disconnect other wireless and Bluetooth devices in the area to isolate possible causes

- Turn off neuroheadset, unplug Transceiver Dongle and repeat the pairing exercise

Sensors do not positively lock in place (sensors should click in place as they are rotated a quarter-turn clockwise after insertion into the socket)

- Sensors are made deliberately tight to start with and the locking tabs are slightly deformed to the correct shape during the first fitting. Sometimes the sensors are very tight and the user feels they will not turn further into the socket.

- Take each sensor in turn and use an empty socket in the Hydrator Pack. Forcibly rotate the sensor fully clockwise in the socket until a definite click is felt (use a cloth or tool to hold the sensor if your fingers are easily damaged). Click and unclick the sensor into the socket a few times to complete the process.

- The sensor can now be fitted to the headset, Make sure you feel a distinct catch as you rotate the sensor into the socket and the finger tabs are aligned along the axis of each arm.

All sensors black except RED references (directly below ears on Contact Quality map)

- Start with RUBBER COMFORT PADS in locations directly behind the ears.

- The primary reference sensor locations are behind the head, elevated at about 30 degrees backwards behind the ears (see diagram in main section of this manual)

- Ensure reference sensors and at least one forehead sensor are sufficiently wet

- Ensure all sensors are properly located in the neuroheadset receptacles. They should not spin or fall out when gently moved.

- Try to minimize the amount of hair trapped underneath the reference sensors. The neuroheadset can be wriggled to allow the sensors to pass through the hair, or you can try to displace some of the hair with a pencil or similar.

- Gently press reference sensors onto the head for at least 5 seconds, then release. It may take another 20 seconds or so for the sensors to respond.

- Gently press wet forehead sensor for 5 seconds then release. It may take another 20 seconds or so for the sensors to respond.

- If still no signals, switch reference sensors to the alternate location as follows: remove the RUBBER COMFORT PADS (including the plastic holders) from their location behind the ears. They should twist out just like any of the sensors. Move the felt pads from the usual reference locations and place them in the sockets recently vacated by the RUBBER COMFORT PADS. You can put the COMFORT PADS into the original reference sockets, or leave them out if you prefer. Make sure you don't lose them if you choose to leave them out.

- Make sure the sensors are sufficiently damp and repeat the above procedure, taking care to locate the new reference sensors onto a patch of bare skin on or near the bony lump located just behind the ear flap. Within a few seconds the sensors should come to life, especially if you press gently on some of the other sensors for a few seconds.

One or both of the sensors immediately adjacent to the ears remains black.

- These sensors are located on the main body of the Arm assembly, closest to the arm pivot point. They detect activity in the temporal lobes and are known as T7 (left side) and T8 (right side). A combination of the shape of the arm assembly and the user's head shape (particularly long, narrow heads with relatively flat sides) can sometimes result in these sensors failing to touch the head, being held off by some of the other sensors.

- Check that the sensors are clean and attached properly as per the general comments in the next section

- Check that the sensors are clean and attached properly as per the general

- Remove the RUBBER COMFORT PAD including the plastic holder, from the side or sides where the contact cannot be achieved. The neuroheadset can be worn comfortably without these pads for people with this head shape, and no harm will come to the connector sockets because they are fully enclosed. The change in balance point is usually sufficient to ensure contact occurs.

- In the unlikely event that contact is still impossible to obtain, you can use a longer felt pad or use a cotton ball soaked in saline to fill the gap or replace the felt piece.

One or more sensors remain black or red for every fitting and user

- Check that the sensor is properly located in the socket. It should click firmly in place, the finger tabs should be aligned along the axis of the arm and it should not freely rotate in the socket.

- Check that the sensor is sufficiently damp for operation

- Check that the sensor is applying gentle but positive pressure to your head in this location. When pressed, it should not move inwards. You may find that slightly relocating the neuroheadset settles the sensors into a better location

- Remove the neuroheadset and gently press the felt pad into the sensor. It should protrude by no more than 2mm. It should also feel damp and be free of obstructions.

- Remove the sensor and inspect it. The sensor should have a felt pad on the front side and a domed gold-plated metal plate which is visible from the rear side. The metal plate should be clean and free of obstructions at least across the central third of the domed section in order to make proper contact when inserted.

- Check the socket. Make sure the gold contact plate is clean and there is nothing trapped in the socket which could affect the contact. The gold contacts inside the socket consist of three spring tabs which bend slightly upwards towards the sensor. Make sure they are not bent away, damaged or dirty.

- Swap a different sensor into that location from somewhere that is known to be working. This will eliminate a faulty sensor

- Please contact Customer Service if you cannot resolve this problem and the location seems to be failing for each fitting and all users.

If you have other problems or your problem is not rectified by the above procedures, please check the updated Troubleshooting information, visit our Live Chat Support, initiate a support ticket at www.emotiv.com or email support@emotiv.com for further assistance.

Appendix 1 EML Language Specification

A1.1 Introduction

EmoComposer™ is a hardware emulator for the Emotiv Software Development Kit. Using EmoComposer, game developers can emulate the behavior of Emotiv EmoEngine™ without needing to spend time in the real Emotiv EPOC™. EmoComposer operates in two modes, interactive and EmoScript playback.

In interactive mode, EmoComposer provides game developers with real time control over generating emulated detection events. EmoComposer also responds to a game's requests in real time. In EmoScript mode, game developers can pre-define these two-way interactions by preparing an EmoComposer Markup Language (EML) document. EML documents are XML documents that can be interpreted by EmoComposer. This section outlines the EML specification.

A1.2 EML Example

A typical EML document is shown in Listing 26 below:

```
01 <?xml version="1.0" encoding="utf-8"?>
02 <!DOCTYPE EML>
03 <EML version="1.0" language="en_US">
04  <config>
05    <autoreset value ="1" group="expressiv_eye" event="blink" />
06    <autoreset value ="1" group="expressiv_eye" event="wink_left"
/>
07    <autoreset value ="1" group="expressiv_eye" event="wink_right"
/>
08  </config>
09  <sequence>
10    <time value="0s15t">
11      <cognitiv event="push" value ="0.85" />
12      <expressiv_upperface event="eyebrow_raised" value ="0.85" />
13      <expressiv_lowerface event="clench" value ="0.85" />
14      <expressiv_eye event="blink" value="1" />
15      <affectiv event="excitement_short_term" value="1" />
16      <affectiv event="excitement_long_term"  value="0.6" />
17      <contact_quality value="G, G, G, G, G, G, F, F, G,
18          G, G, G, G, G, G, G, G, G" />
19    </time>
20    <time value="2s4t">
21      <cognitiv event="push" value ="0" />
22      <expressiv_upperface event="eyebrow_raised" value ="0.75" />
23      <expressiv_lowerface event="clench" value ="0.5" />
24      <expressiv_eye event="blink" value="1" />
25      <affectiv event="excitement_short_term" value="0.7" />
26      <affectiv event="excitement_long_term"  value="0.6" />
27    </time>
28    <time value="3s6t">
29 <cognitiv event="push" shape="normal" offset_left="0.4"
offset_right="0.2"
30          scale_width="1.5" scale_height="0.8" />
31      <expressiv_upperface event="eyebrow_raised" value ="0.85" />
32      <expressiv_lowerface event="clench" value ="0.85" />
33      <expressiv_eye event="blink" value="1" repeat="1"
34          repeat_interval="0.5" repeat_num="15" />
```

```
35      <affectiv event="excitement_short_term" value="0.4" />
36      <affectiv event="excitement_long_term" value="0.5" />
37    </time>
38  </sequence>
39 </EML>
```

Listing 26 *EML Document Example*

Apart from standard headers (lines 1-3 and 39), an EML document consists of two sections:

- `config`: Section to configure global parameter for the EmoComposer behaviors.
- `sequence`: Section to define detection events as they would occur in a real Emotiv SDK.

A1.2.1 EML Header

Line 1-3 specifies the EML header. EML is a special implementation of a generic XML document which uses UTF-8 encoding and English US language. Line 2 is a normal XML comment to specify the document type and is optional.

```
01 <?xml version="1.0" encoding="utf-8"?>
02 <!DOCTYPE EML>
03 <EML version="1.0" language="en_US">
```

Listing 27 *EML Header*

A1.2.2 EmoState Events in EML

EmoState events are defined within the <sequence> element. In Listing 28, the <sequence> element is between line 9 and line 38:

```
09   <sequence>
10     <time value="0s15t">
11       <cognitiv event="push" value ="0.85" />
12       <expressiv_upperface event="eyebrow_raised" value ="0.85" />
13       <expressiv_lowerface event="clench" value ="0.85" />
14       <expressiv_eye event="blink" value="1" />
15       <affectiv event="excitement_short_term" value="1" />
16       <affectiv event="excitement_long_term"  value="0.6" />
17       <contact_quality value="G, G, G, G, F, F, P, F, G,
18           G, G, G, G, G, G, G, G, G" />
19     </time>
20     <time value="2s4t">
21       <cognitiv event="push" value ="0" />
22       <expressiv_upperface event="eyebrow_raised" value ="0.75" />
23       <expressiv_lowerface event="clench" value ="0.5" />
24       <expressiv_eye event="blink" value="1" />
25       <affectiv event="excitement_short_term" value="0.7" />
26       <affectiv event="excitement_long_term"  value="0.6" />
27     </time>
28     <time value="3s6t">
29 <cognitiv event="push" shape="normal" offset_left="0.4"
offset_right="0.2"
30          scale_width="1.5" scale_height="0.8" />
31       <expressiv_upperface event="eyebrow_raised" value ="0.85" />
32       <expressiv_lowerface event="clench" value ="0.85" />
33       <expressiv_eye event="blink" value="1" repeat="1"
```

```
34              repeat_interval="0.5" repeat_num="15" />
35        <affectiv event="excitement_short_term" value="0.4" />
36        <affectiv event="excitement_long_term" value="0.5" />
37     </time>
38  </sequence>
```

Listing 28   *Sequence in EML document*

The `<sequence>` section consists of a series of discrete times at which there are events that will be sent from the EmoComposer to the game. These time events are ascending in time. Since each second is divided into 32 ticks (or frames), the time value in this example should be understood as follows:

Time	Line Number	Description
`value = "0s15t"`	10	This event is at 0 seconds and 15th frame
`value = "2s4t"`	20	This event is at 2 seconds and 4th frame
`value = "3s6t"`	28	This event is at 3 seconds and 6th frame

Table 4   *Time values in EML documents*

At each time event, game developers can specify up to six different parameters, corresponding to the five distinct detection groups plus the current signal quality:

Detection Group	Events	Notes
`cognitiv`	push pull lift drop left right rotate_left rotate_right rotate_clockwise rotate_counter_clockwise rotate_forwards rotate_reverse disappear	
`expressiv_eye`	blink wink_left wink_right look_left look_right	"`value`" attribute is treated as a boolean (0 or not 0) to determine whether to set the specified eye state.
`expressiv_upperface`	eyebrow_raised furrow	
`expressiv_lowerface`	smile	

	`clench`
	`laugh`
	`smirk_left`
	`smirk_right`	

`affectiv`	`excitement_short_term`
	`excitement_long_term`
	`engagement_boredom`
		Notes:
		
		1. The `affectiv` tag is a special case in that it is allowed to appear multiple times, in order to simulate output from all the Affectiv detections.
		
		2. In order to simulate the behavior of the EmoEngine™, both short and long term values should be specified for excitement.

`signal_quality`	`value`	**This tag has been deprecated. It has been replaced with the contact_quality tag.** Expects "`value`" attribute to be formatted as 18 comma-separated floating point values between 0 and 1. The first two values must be the same.
`contact_quality`	`value`	Expects "`value`" attribute to be formatted as 18 comma-separated character codes that correspond to valid CQ constants: G = `EEG_CQ_GOOD` F = `EEG_CQ_FAIR` P = `EEG_CQ_POOR` VB = `EEG_CQ_VERY_BAD` NS = `EEG_CQ_NO_SIGNAL` The first two values must be the same, and can only be set to G, VB, or NS, in order to most accurately simulate possible values produced by the Emotiv neuroheadset hardware. The order of the character codes is the same as the contants in the `EE_InputChannels_enum` declared in EmoStateDLL.h. Note that two of the channels,

		FP1 and FP2, do not currently exist on the SDK or EPOC neuroheadsets.

Table 5 *Detection groups in EML document*

Detection group names are created by grouping mutually exclusive events together. For example, only one of {blink, wink_left, wink_right, look_left, look_right} can happen at a given time, hence the grouping expressiv_eye.

Cognitiv detection group belongs to the Cognitiv Detection Suite. Expressiv_eye, Expressiv_upperface, and Expressiv_lowerface detection groups belong to the Expressiv Detection Suite. Affectiv detection group belongs to the Affectiv Detection Suite.

In its simplest form, a detection definition parameter looks like: <cognitiv event="push" value="0.85" />, which is a discrete push action of the Cognitiv detection group with a value of 0.85. In EML, the maximum amplitude for any detection event is 1. By default, the detection event retains its value for this detection group until the game developer explicitly set it to a different value. However, game developers can also alter the reset behaviors as shown in the config section where the values for blink, wink_left, wink_right of the expressiv_eye detection group automatically reset themselves.

```
04  <config>
05    <autoreset value ="1" group="expressiv_eye" event="blink" />
06    <autoreset value ="1" group="expressiv_eye" event="wink_left"
/>
07    <autoreset value ="1" group="expressiv_eye" event="wink_right"
/>
08  </config>
```

Listing 29 *Configuring detections to automatically reset*

Instead of a discrete detection event as above, game developers can also define a series of detection events based on an event template function. An event template function generates a burst of discrete events according to the following parameters:
- shape: "normal" or "triangle"
- offset_left, offset_right, scale_width: A template has a 1 second width by default. These three parameters allow game developers to morph the template shape in the time domain.
- scale_height: A template, by default, has maximum amplitude of 1. This parameter allows game developers to morph the template's height.
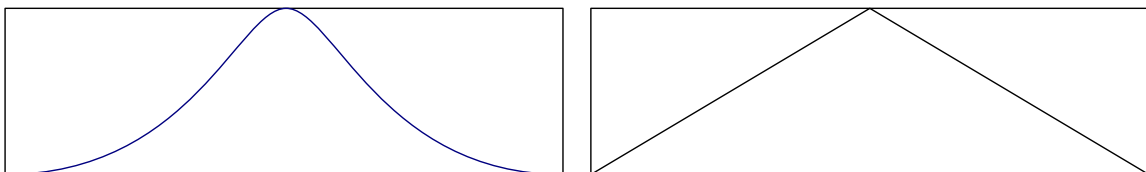
Normal and Triangle shapes are shown below:

Figure 34 *Normal and Triangle template shapes*

An example of morphing template to specify detection event is:

```
29 <cognitiv event="push" shape="normal" offset_left="0.4"
offset_right="0.2"
30        scale_width="1.5" scale_height="0.8" />
```

The above detection event can be illustrated as below:

First, start with a normal template with `height = 1` and `width = 1`. Second, the template is adjusted by `offset_left` and `offset_right`. It now has a height of 1 and a width of 1-0.4-0.2 = 0.4.

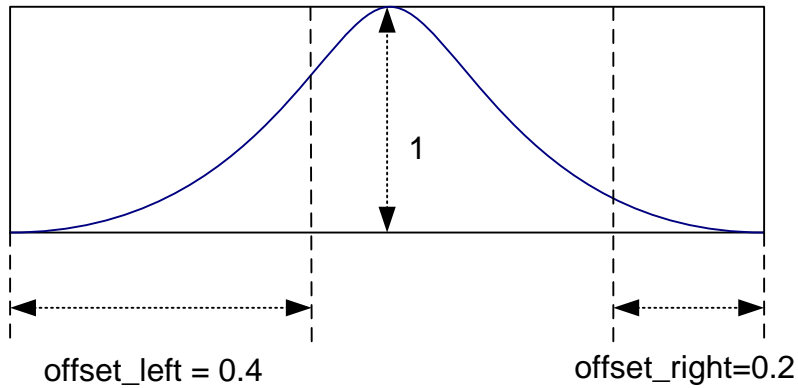offset_left = 0.4   offset_right=0.2

Figure 35   *Morphing a template*

Last, after height is scaled by `scale_height` and width is scaled by `scale_width`, the template becomes:
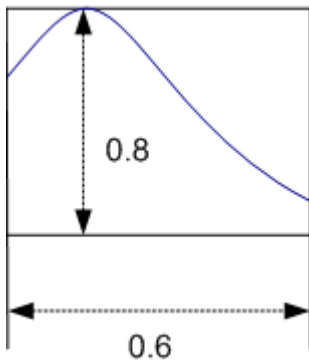

Figure 36   *Morphed template*

Full specifications of an event's attributes is shown below:

Attribute	Description	Required
`[detection_group]`	One of six available detection groups as specified in Table 5.	Yes
`event=[event_name]`	Corresponding values of the `[detection_group]` as specified in Table 5.	Yes
`value=[value]`	A detection event can be interpreted as either a discrete event or a series of events whose values are determined by an event template function. The presence of the "value" attribute indicates that this is a	Either "`value`" or "shape" attribute must be specified. If "`value`" is present, none of the event template attributes (shape, `offset_left`, `offset_right`,

	discrete event.	scale_width, scale_height) are allowed
shape=[shape]	The presence of the "shape" attribute indicates that this represents the starting point for a series of events generated according to an event template function. Allowed values are "normal" and "triangle".	Either "value" or "shape" attribute must be specified. If "shape" is present, then the "value" attribute is not allowed.
offset_left=[offset_left]	This attribute is a parameter of an event template function (see above for a detailed description of its meaning). offset_left+offset_right must be less than 1.	The "shape" attribute must also be specified. The "value" attribute can not be specified.
offset_right=[offset_right]	This attribute is a parameter of an event template function (see above for a detailed description of its meaning). offset_left+offset_right must be less than 1.	The "shape" attribute must also be specified. The "value" attribute can not be specified.
scale_width=[scale_width]	This attribute is a parameter of an event template function (see above for a detailed description of its meaning). Must be greater than 0.	The "shape" attribute must also be specified. The "value" attribute can not be specified.
scale_height=[scale_height]	This attribute is a parameter of an event template function (see above for a detailed description of its meaning). 0 < scale_height <= 1	- The "shape" attribute must also be specified. - The "value" attribute can not be specified.

Table 6   *Attributes for an event specification*

Appendix 2 Emotiv EmoEngine™ Error Codes

Every time you use a function provided by the API, the value returned indicates the EmoEngine™ status. Table 7 below shows possible EmoEngine error codes and their meanings. Unless the returned code is EDK OK, there is an error. Explanations of these messages are in Table 7 below.

EmoEngine Error Code	Hex Value	Description
EDK_OK	0x0000	Operation has been carried out successfully.
EDK_UNKNOWN_ERROR	0x0001	An internal fatal error occurred.
EDK_INVALID_PROFILE_ARCHIVE	0x0101	Most likely returned by EE_SetUserProfile() when the content of the supplied buffer is not a valid serialized EmoEngine profile.
EDK_NO_USER_FOR_BASE_PROFILE	0x0102	Returns when trying to query the user ID of a base profile.
EDK_CANNOT_ACQUIRE_DATA	0x0200	Returns when EmoEngine is unable to acquire any signal from Emotiv EPOC™ for processing
EDK_BUFFER_TOO_SMALL	0x0300	Most likely returned by EE_GetUserProfile() when the size of the supplied buffer is not large enough to hold the profile.
EDK_OUT_OF_RANGE	0x0301	One of the parameters supplied to the function is out of range.
EDK_INVALID_PARAMETER	0x0302	One of the parameters supplied to the function is invalid (e.g. null pointers, zero size buffer)
EDK_PARAMETER_LOCKED	0x0303	The parameter value is currently locked by a running detection and cannot be modified at this time.
EDK_COG_INVALID_TRAINING_ACTION	0x0304	The specified action is not an allowed training action at this time.
EDK_COG_INVALID_TRAINING_CONTROL	0x0305	The specified control flag is not an allowed training control at this time.
EDK_COG_INVALID_ACTIVE_ACTION	0x0306	An undefined action bit has been set in the actions bit vector.
EDK_COG_EXCESS_MAX_ACTIONS	0x0307	The current action bit vector contains more than maximum number of concurrent actions.
EDK_EXP_NO_SIG_AVAILABLE	0x0308	A trained signature is not currently

EmoEngine Error Code	Hex Value	Description
		available for use – some actions may still require training data.
`EDK_INVALID_USER_ID`	`0x0400`	The user ID supplied to the function is invalid.
`EDK_EMOENGINE_UNINITIALIZED`	`0x0500`	EmoEngine™ needs to be initialized via calling `EE_EngineConnect()` or `EE_EngineRemoteConnect()` before calling any other APIs.
`EDK_EMOENGINE_DISCONNECTED`	`0x0501`	The connection with EmoEngine™ via EE_EngineRemoteConnect() has been lost.
`EDK_EMOENGINE_PROXY_ERROR`	`0x0502`	Returned by `EE_EngineRemoteConnect()` when the connection to the EmoEngine™ cannot be established.
`EDK_NO_EVENT`	`0x0600`	Returned by `EE_EngineGetNextEvent()` when there is no pending event.
`EDK_GYRO_NOT_CALIBRATED`	`0x0700`	The gyroscope is not calibrated. Please ask the user to remain still for .5 seconds.
`EDK_OPTIMIZATION_IS_ON`	`0x0800`	Operation failed due to algorithm optimization settings.

Table 7 *Emotiv EmoEngine™ Error Codes*

Appendix 3 Emotiv EmoEngine™ Events

In order for an application to communicate with Emotiv EmoEngine, the program must regularly check for new EmoEngine events and handle them accordingly. Emotiv EmoEngine events are listed in Table 8 below:

EmoEngine events	Hex Value	Description
`EE_UserAdded`	`0x0010`	New user is registered with the EmoEngine
`EE_UserRemoved`	`0x0020`	User is removed from the EmoEngine's user list
`EE_EmoStateUpdated`	`0x0040`	New detection is available
`EE_ProfileEvent`	`0x0080`	Notification from EmoEngine in response to a request to acquire profile of an user
`EE_CognitivEvent`	`0x0100`	Event related to Cognitiv detection suite. Use the `EE_CognitivGetEventType` `function` to retrieve the Cognitiv-specific event type.
`EE_ExpressivEvent`	`0x0200`	Event related to the Expressiv detection suite. Use the `EE_ExpressivGetEventType function` to retrieve the Expressiv-specific event type.
`EE_InternalStateChanged`	`0x0400`	Not generated for most applications. Used by Emotiv Control Panel to inform UI that a remotely connected application has modified the state of the embedded EmoEngine through the API.
`EE_EmulatorError`	`0x0001`	EmoEngine internal error.

Table 8   *Emotiv EmoEngine™ Events*

Appendix 4 Redistributing Emotiv EmoEngine™ with your application

An application constructed to use Emotiv EmoEngine™ requires that EDK.dll be installed on the end-user's computer. EDK.dll has been compiled with Microsoft Visual Studio 2005 (VC 8.0) SP1 and depends upon the shared C/C++ run-time libraries (CRT) that ship with this version of the compiler. The appropriate shared run-time libraries are installed on the application developer's machine by the Emotiv SDK™ Installer, but the developer is responsible for ensuring that the appropriate run-time libraries are installed on an end-user's computer by the developer's application installer before EDK.dll can be used on that machine.

If the application developer is using Visual Studio 2005 SP1+ to build her application then it is likely that no additional run-time libraries, beyond those already required by the application, need to be installed on the end-user's computer in order to support EDK.dll. Specifically, EDK.dll requires that Microsoft.VC80.CRT version 8.0.50727.762 or later be installed on the end-user's machine. Please see the following Microsoft documentation: [Redistributing Visual C++ files](#) and [Visual C++ Libraries as Shared Side-by-Side Assemblies](#) for more information about how to install the appropriate Microsoft shared run-time libraries or contact Emotiv's SDK support team for further assistance.

If the application is built using an older or newer major version of the Visual Studio compiler, such as Visual Studio 2003 or 2008, or another compiler altogether, then EDK.dll and the application will use different copies of the C/C++ run-time library (CRT). This will usually not cause a problem because EDK.dll doesn't rely on any shared static state with the application's instance of the CRT, but the application developer needs to be aware of some potentially subtle implications of using multiple instances of the CRT in the same process. Please refer to Microsoft's [C Run-Time Libraries (CRT)](#) documentation for more information on this subject. Depending on the particular compiler/run-time library mismatch involved, Emotiv may be able to provide a custom build of EDK.dll for developers who wish to use another compiler. Please contact the Emotiv SDK support team if you think you might require such a custom build.