**ALPHA OMEGA**

**U.S. Office:**

Toll Free 1-877-919-6288

Fax 1-877-471-2055

**Europe Office:**

Toll Free: 00-800-2-574-2111

Tel +49-7-251-440-6620

**Home Office:**

Nazareth Industrial Park Building, Mount Precipice

P.O. Box 2268 Nazareth 1612102, Israel.

Tel. 972-4-6563-327 Fax: 972-4-6574-075

**Email:** info@alphaomega-eng.com

**Website:** www.alphaomega-eng.com

# Neuro Omega™

## Physiological Navigation System for Neurosurgery

### and Neurophysiological Clinical Applications

# Neuro Omega SDK User Manual
# Version 1.3

Refer to Neuro Omega User Manual

**December, 2014**

# PROPRIETARY NOTICE

This publication, or parts thereof, contains information proprietary to Alpha Omega Engineering and may not be reproduced, recorded or transmitted in any form, by any method, for any purpose without written permission from Alpha Omega Engineering.

Alpha Omega Engineering Ltd. makes no warranty, expressed or implied, including but not limited to any implied warranties of merchantability or fitness for a particular purpose or use, regarding these materials and makes such materials available solely on an “as-is” basis.

In no event shall Alpha Omega Engineering Ltd. be liable to anyone for specific, collateral, incidental, or consequential damages in connections with or arising from purchase or use of these materials. The sole and exclusive liability to Alpha Omega Engineering Ltd., regardless of the form of actions, shall not exceed the purchase price of the materials described herein.

The Neuro Omega and Alpha Omega are trademarks of Alpha Omega Engineering Ltd.

For additional information on the device, including questions on infection control procedures, please contact:

| 🏭 | EC REP |
|---|---|
| **Contact Information:** | **Name and Address of the European Authorized Representative:** |
| ALPHA OMEGA ENGINEERING<br>Nazareth Industrial Park Building<br>Mount Precipice<br>P.O. Box 2268<br>Nazareth 1612102<br>Israel<br>Tel: +972-4-656-3327<br>Fax: +972-4-657-4075<br>Email:<br>info@alphaomega-eng.com<br>support@alphaomega-eng.com<br>http://www.alphaomega-eng.com | Mr. Yousef Bsoul<br>Europe Sales Manager<br>Alpha Omega GmbH<br>Ubstadter Str. 28<br>Ubstadt-Weiher,76698<br>Germany<br>Tel: +49 (0) 7251-4406620<br>Fax: +49 (0) 7251-2391034<br>Toll free: 00-800-2574-2111<br>Email:<br>info-EU@alphaomega-eng.com<br>http://www.alphaomega-eng.com |

U.S. Office: Toll Free: 1-877-919-6288, Fax: 1-877-471-2055

Europe Office: Toll Free: 00-800-2-574-2111, Tel +49-7-251-440-6620

# Contents

# 1 Overview

The Neuro Omega is a physiological navigation system intended for different neurosurgery and neurophysiological clinical applications, including recording from and stimulating brain motor and sensory neurons to accurately navigate for neurosurgery target localization in treatment of movement disorders and to aid in the placement of depth electrodes.

The system records and stimulates brain peripheral-nerve electrical activity from various areas of the brain (deep structures and surface areas).

The device is also designed to measure bioelectric signals produced by muscles (EMG) and stimulate peripheral nerves to aid in the diagnosis and prognosis of neuromuscular disease for target localization surgeries for motor movement disorders or for intra-operative skeletal muscles activity. This can be done with recording or stimulation.

The device may also be used to measure and record the electrical activity of the patient's brain, obtained by placing two or more electrodes on the head (EEG). This is for cortical and surface electrical activity levels of the brain.

The device is also designed for temporary monitoring of brain electrical activity from deep or cortical brain during neurosurgery in the operating room or outside the clinical environment.

## 1.1 Regulatory

### 1.1.1 Adverse Effects

The possible adverse effects relating to Sterotactic Neurosurgery are:

- The possibility of intracranial hemorrhage associated with the introduction of probes into the brain.
- Visual field impairment with optic tract injuries.
- Contra lateral motor deficit with corticospinal injury.

### 1.1.2 FDA System Classification

- **Product Code**: GZL
- **Subsequent Product Code**: GWF, IKN, GWQ
- **CFR Section**: 21 CFR 882.1330
- **Regulation Name**: Depth electrode
- **Subsequent Regulation Names**:
  - Electroencephalograph
  - Stimulator
  - Electrical
  - Evoked response

    - Electromyograph
    - Diagnostic
- **Trade Name**: Neuro Omega System
- **Common Name**: Intraoperative neurophysiological recording and stimulating device
- **Classification**: Class II

## 1.2 Intended Uses

The Neuro Omega System is intended for the following:

- Assisting neurosurgeons in the operating room during functional neurosurgery
- Recording from and stimulating brain motor and sensory neurons to aid in the placement of depth electrodes
- Monitoring, recording, and displaying the bioelectric signals produced by muscles
- Stimulating peripheral nerves
- Monitoring, recording, and displaying the electrical activity produced by nerves (EMG) for aiding the clinician in the diagnosis and prognosis of neuromuscular disease.
- Measuring and recording the electrical activity of the patient's brain obtained by placing two or more electrodes on the head (EEG).

## 1.3 Conditions of Use

The device may be used by medical personnel within a hospital, laboratory, clinic, or nursing home setting, or outside of a medical facility under direct supervision of a medical professional. The device may also be placed in the intensive care unit or operating room for continuous recording.

The following are the Neuro Omega system use conditions:

- **Environment**:
    - Conditions of visibility:
        - Ambient luminance range: Normal
        - Viewing distance: N/A
        - Viewing angle: N/A
    - **Physical**:
        - Temperature range: 0°C to +40°C
        - Relative humidity range: 10% - 80%, non-condensing
        - Ambient pressure range: 500 hPa to 1060 hPa

  - Background sound pressure level: Normal
- **Frequency of Use**: As per specific case
- **Mobility**: Mobile

## 1.4 Warnings

⚠ **Warnings:**

- This is a Class A product. In a medical environment this product may cause radio interference in which case the user will be required to take adequate measures.
- Only qualified personnel, who have been trained by Alpha Omega Ltd., should be allowed to operate this equipment.
- Any modifications made to the equipment without explicit approval from Alpha Omega Ltd., voids warranty and service contract obligations, and poses a potential safety threat to both operators and patients.
- Do not install any software packages (Matlab, C++, SDK software or other) on the system unless provided by Alpha Omega Ltd. for the explicit use on the Neuro Omega.
- Neuro Omega system and Neuro Omega drive should be connected to Alpha Omega NeuroProbes for recording and stimulation
- External systems connected to the Neuro Omega must be independently isolated, or powered through the trolley, as this has its own isolation transformer.
- The Neuro Omega system should be placed outside of the patient environment or any area that can, intentionally or unintentionally, come in contact with the patient.
- A thorough understanding of the technical principles, clinical applications, and risks associated with this treatment is necessary before using this system. Please read this entire manual before attempting to activate the system. Completion of the training program is required prior to use of the Neuro Omega system.
- The analog and digital input output panel (ADIO) is not an applied part, and therefore should not be connected to the patient without proper electrical isolation.

⚠ **Cautions:**

- US federal law restricts the sale of this device to or on the order of a physician.
- Discard according to the local regulations and law.

**Notes:**

- The Neuro Omega system is provided non-sterile or sterile. Please refer to the Neuro Omega Manual for detailed sterilization instructions of system and accessories.
- It is the user’s responsibility to qualify any deviations from the recommended method of processing.

- Please contact the manufacturer or local distributor to request a copy of the insulation diagram if needed.
- This product has been tested and found to comply with the limits for Class a Medical Device according to IEC 60601-1 and IEC 60601-1-2 Standards. The limits for Class A equipment were derived for medical environments to provide reasonable protection against interference with licensed communication and medical equipment.

## 1.5 Electromagnetic Conformance

The following tables contain information on electromagnetic emissions for guidance and manufacturer's declaration:

- *Guidance and Manufacturer's Declaration – Electromagnetic Emissions*
- *Guidance and Manufacturer's Declaration – Electromagnetic Immunity*
- *Recommended Separation Distances between Portable and Mobile RF Communications Equipment and the Neuro Omega*

**Notes:**

- This product has been tested and found to comply with the limits for Class a Medical Device according to IEC 60601-1 and IEC 60601-1-2 Standards. The limits for Class A equipment were derived for medical environments to provide reasonable protection against interference with licensed communication and medical equipment.
- This product must be installed and put into service according to the EMC information provided in the tables below.
- Portable and mobile RF communications equipment can affect this product.

**Warnings:**

- This is a Class A product. This product is intended for use by healthcare professionals only. This equipment/system may cause radio interference or may disrupt the operation of nearby equipment. It may be necessary to take mitigation measures, such as re-orienting or relocating the Neuro Omega or shielding the location.
- The use of accessories, transducers, and cables other than those specified by the manufacturer may result in increased emissions or the decreased immunity of the Neuro Omega.
- The Neuro Omega should not be used adjacent to or stacked with other equipment. If adjacent of stacked use is necessary, the Neuro Omega should be observed to verify normal operation in the configuration in which it will be used.

The Neuro Omega is intended for use in the electromagnetic environment specified in ***Table 1***. The user of the Neuro Omega should assure that it is used in such an environment.

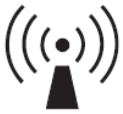
**Table 1: Guidance and Manufacturer's Declaration - Electromagnetic Emissions**

| Emissions Test | Compliance | Electromagnetic Environment Guidance |
|---|---|---|
| RF emissions CISPR 11 | Group 1 | The Neuro Omega uses RF energy only for its internal function. Therefore, its RF emissions are very low and are not likely to cause any interferences in nearby electronic equipment. |
| RF emissions CISPR 11 | Class A | The Neuro Omega is suitable for use in all establishments other than domestic, and may be used in domestic establishments and those directly connected to the public low-voltage power supply network that supplies buildings used for domestic purposes. |
| Harmonic emissions<br>IEC 61000-3-2 | Class A | |
| Voltage fluctuations/flicker emissions<br>IEC 61000-3-3 | Complies | |

The Neuro Omega is intended for use in the electromagnetic environment specified in ***Table 2***. The customer or the user of the Neuro Omega should assure that it is used in such an environment.

**Table 2: Guidance and Manufacturer's Declaration - Electromagnetic Immunity**

| Immunity Test | IEC 60601 test level | Compliance | Electromagnetic Environment Guidance |
|---|---|---|---|
| Electrostatic discharge (ESD)<br>IEC 61000-4-2 | ±6kV contact<br>±8kV air | ±6kV contact<br>±8kV air | Floors should be wood, concrete or ceramic tile. If floors are covered with synthetic material, the relative humidity should be less than 30%. |
| Electrostatic fast transient/burst<br>IEC 61000-4-4 | ±2kV for power supply lines<br>±1kV for input/output lines | ±2kV for power supply lines<br>±1kV for input/output lines | Mains power quality should be that of a typical commercial or hospital environment. |
| Surge<br>IEC 61000-4-5 | ±1kV line(s) to line(s)<br>±2kV line(s) to earth | ±1kV line(s) to line(s)<br>±2kV line(s) to earth | Mains power quality should be that of a typical commercial or hospital environment. |
| Voltage dips, short interruptions and voltage variations on power supply input lines<br>IEC 61000-4-11 | <5% *U*T for 0.5 cycles<br>40% *U*T for 5 cycles<br>70% *U*T for 25 cycles<br><5% *U*T for 5 s | <5% *U*T for 0.5 cycles<br>40% *U*T for 5 cycles<br>70% *U*T for 25 cycles<br><5% *U*T for 5 s | Mains power quality should be that of a typical commercial or hospital environment. If the user of the Neuro Omega requires continued operation during power mains interruptions, it is recommended that the NeuroOmega be powered from an uninterruptible power supply battery. |
| Power frequency (50/60 Hz) magnetic field<br>IEC 61000-4-8 | 3 A/m | 3 A/m | Mains power quality should be that of a typical commercial or hospital environment. |

| | | | |
|---|---|---|---|
| | | | Portable and mobile RF communications equipment should be used no closer to any part of the NeuroOmega, including cables, than the recommended separation distance calculated from the equation applicable to the frequency of the transmitter. |
| Conducted RF<br>IEC 61000-4-6 | 3 Vrms 150 kHz to 80 MHz | 3 Vrms 150 kHz to 80 MHz | Recommended separation distance:<br>$d=1.2\sqrt{P}$<br>$d=1.2\sqrt{P}$ *80 MHz to 800 MHz*<br>$d=2.4\sqrt{P}$ *800 MHz to 2.5GHz* |
| Radiated RF<br>IEC 61000-4-3 | 3 V/m 80 MHz to 2.5 GHz | 3 V/m 80 MHz to 2.5 GHz | Where *P* is the maximum output power rating of the transmitter in watts (W) according to the transmitter manufacturer and *d* is the recommended separation distance in meters (m).<br>Field strength from fixed RF transmitters, as determined by an electromagnetic site survey,[1] should be less than the compliance level in each frequency range.[2]<br>Interference may occur in the vicinity of equipment marked with the following symbol:<br> |

**Notes:**

- At 80 MHz and 800 MHz, the higher frequency range applies.
- These guidelines may not apply in all situations. Electromagnetic propagation is affected be absorption and reflection from structures, objects and people.

[1] Field strength from fixed transmitters, such as base stations for radio (cellular/cordless) telephones and land mobile radios, amateur radio, AM and FM radio broadcast and TV broadcast cannot be predicted theoretically with accuracy. To assess the electromagnetic environment due to fixed RF transmitters, an electromagnetic site survey should be considered. If the measured field strength in the location in which the Neuro Omega is used exceeds the applicable RF compliance level above, the Neuro Omega should be observed to verify normal operation. If abnormal performance is observed, additional measures may be necessary, such as re-orienting or relocating the Neuro Omega.

[2] Over the frequency range 150 kHz to 80 MHz, field strength should be less than 3 V/m·

The Neuro Omega is intended for use in the electromagnetic environment in which radiated RF disturbances are controlled. The customer or the user of the Neuro Omega can help prevent electromagnetic interference by maintaining a minimum distance between portable and mobile RF communications equipment (transmitters) and the Neuro Omega as recommended in ***Table 3***, according to the maximum output power of the communications equipment.

**Table 3: Recommended Separation Distances between Portable and Mobile RF Communications Equipment and the Neuro Omega**

| **Rated maximum output power of transmitter**<br>**W** | **Separation distance according to frequency of transmitter**<br>**m** | | |
|---|---|---|---|
| | 150 kHz to 80 MHz<br>$d=1.2\sqrt{P}$ | 80 MHz to 800 MHz<br>$d=1.2\sqrt{P}$ | 800 MHZ to 2.5 GHz<br>$d=2.4\sqrt{P}$ |
| 0.01 | 0.12 | 0.12 | 0.24 |
| 0.1 | 0.37 | 0.37 | 0.74 |
| 1 | 1.2 | 1.2 | 2.4 |
| 10 | 3.7 | 3.7 | 7.4 |
| 100 | 12 | 12 | 24 |

For transmitters rated at maximum output power not listed above, the recommended separation distance *d* in meters (m) can be estimated using the equation applicable to the frequency of the transmitter, where *p* is the maximum output power rating of the transmitter in watts (W) according to the transmitter manufacturer.

**Notes:**

- At 80 MHz and 800 MHz, the separation distance for the higher frequency range applies.
- These guidelines may not apply in all situations. Electromagnetic propagation is affected by absorption and reflection from structures, objects and people.

# 2 Software Development Kit Research Capabilities

## 2.1 Software Development Kit Research Overview

Aside from basic stimulation as described in the Neuro Omega Manual there is an additional method for controlling stimulation paradigms and completing data analysis:

- **Coding**: For greater control and complexity, code and run the stimulation paradigms through the MATLAB or C++ tool on an external computer. Coding is described in section *2.3*.

> **Note**: Some advanced research capabilities involve external systems. Connecting these systems is described in section *2.2*.

## 2.2 Connecting External Systems

This procedure describes how to connect any external systems to the Alpha Omega, such as:

- External analog or digital input or output systems
- The computer running MATLAB or  C++
- External monitors

You can power the systems through the trolley's isolation transformer, or through an independent isolation transformer.

> **Warning:**
>
> - External systems connected to the Neuro Omega must be independently isolated, or powered through the trolley, as this has its own isolation transformer.
> - External systems connected to the Neuro Omega by Ethernet port must include Ethernet Isolator in line.

**To power an external system:**

a. Do one of the following:

♦ Power the system through the trolley's isolation transformer (see *Figure 1: Neuro Omega Trolley Side View*) as follows:

i. On the base of the Main Unit, remove the cover to the isolation transformer.

ii. Plug the external computer in to the transformer.

iii. Return the cover, threading the power cord parallel to the Neuro Omega system's power cord.

♦ Power the system through an independent isolation transformer.

On the Input/output panel, connect the system to the required connection.

Repeat steps ***a*** and ***0*** for each system you want to connect.

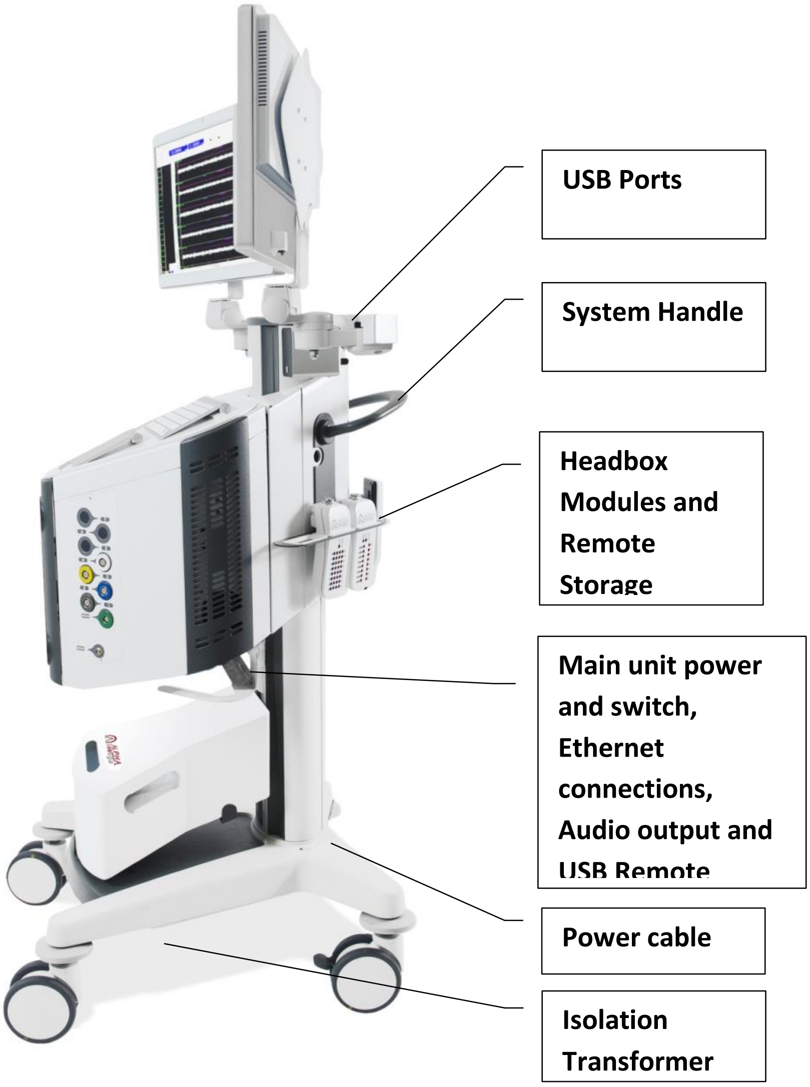

USB Ports
System Handle
Headbox Modules and Remote Storage
Main unit power and switch, Ethernet connections, Audio output and USB Remote
Power cable
Isolation Transformer

**Figure 1: Neuro Omega Trolley Side View**

**To connect the MATLAB or C++ ethernet connection:**

a. Use a Cat6 ethernet cable and connect the external computer to the Neuro Omega (see *Figure 1: Neuro Omega Trolley Side View*) as follows:

   i. On the base of the Main Unit, plug in the ethernet cable to one of the open ports (see *Figure 2: Bottom of Main Unit*)

   ii. Plug the ethernet cable to Ethernet Isolator (pay attention to direction).

   iii. Plug another ethernet cable to the Matlab or C++ computer.

   iv. Plug the other ethernet cable to the Ethernet Isolator.

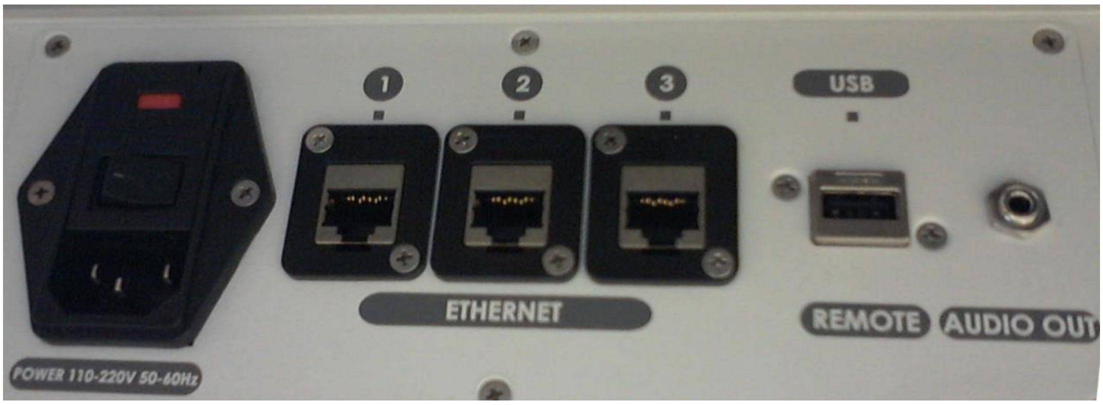


**Figure 2: Bottom of Main Unit**

## 2.3 Controlling Stimulation Paradigms through Coding

This procedure describes how run code using MATLAB to control stimulation paradigms. Running code affords more control over the paradigms and the ability to run additional signal processing needs.

Coding and running the code is performed on an external computer.

**To control stimulation paradigms through coding:**

1. Connect the external computer as described in section **2.2**.
2. With an ethernet cable, connect the external computer to the Main Unit described in section **2.2**.
3. Prepare the external computer, as described in ***2.3.1***.
4. Write the code the MATLAB functions as commands, using **2.3.2** as a guide.

### 2.3.1 Preparing the External Computer

This procedure describes how to prepare the external computer in order to use the MATLAB tool and connect to the Neuro Omega system.

To prepare the external computer for using MATLAB:

1. Install MATLAB Tool by running the supplied setup file and following the on screen instructions.
2. Start MATLAB, as follows:

   b. Open MATLAB.

> **Note:**
>
> - With Windows 7, you may need to run MATLAB as Administrator, or change the user settings to lower administrative control.

   c. Set the working directory path in MATLAB to the installed MATLAB Tool Directory, for example, as follows:

```
C:\Program Files(x86)\AlphaOmega\AO_MATLABTool
```

> **Note:** If you have MATLAB 2014a and you have the visual distribution library 2010 for 64bit then the next step you don't have to go over them and start with step

d. Set up the compiler and compile the MEX file, as follow:

1. In the MATLAB command window, type `mex -setup`, and then press **ENTER**.

2. A MATLAB message appears in the command window:

- **Would you like mex to locate installed compilers [y]/n?]**

3. Press n.

- MATLAB suggests a list of all supported compilers.

4. Select a version of Microsoft Visual, such as Microsoft Visual C++ 2008 or 2010.

> **Note:** If you do not have the compiler on your PC, you need to install it before continuing (express mode is downloaded for free).

5. Continue the procedure for choosing the compiler by answering the questions in the wizard. For the path validation, if the path is correct, answer **y** to all questions.

6. Make sure that compiler is existing using the following command:

- cc = mex.getCompilerConfigurations()

  if there is no compiler follow the troubleshooting guideline.

7. Compile the MEX files as follows:

- run the following in the MATLAB command window:

  for 32bits:

  ```
  mex MexFileEthernetStandAlone.cpp
  Include\ethernetStandAlone.lib
  ```

  for 64bits:

  ```
  mex MexFileEthernetStandAlone.cpp
  Include\ethernetStandAloneX64.lib
  ```

- The following results:

i. The **MexFileEthernetStandAlone.cpp** file is compiled.

ii. A MEX file is created, called MexFileEthernetStandAlone.mexw32.

iii. Installation concludes

iv. If the compile fails, see Troubleshooting Section 5

e. Test the installation by doing one of the following:

- In the MATLAB command window, type `AO_IsConnected`, and then press ENTER.

If no compilation error appears, which is usually indicated by red colored messages, installation was successful.

Else see Troubleshooting Section 5

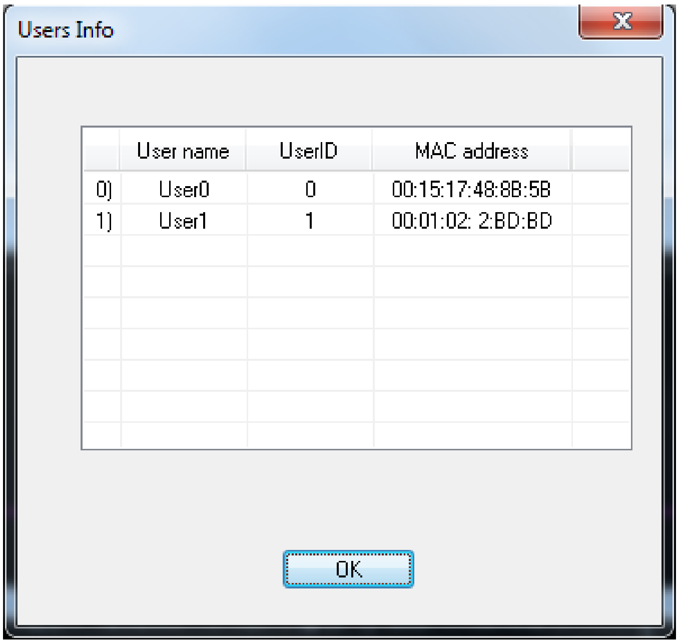
### 2.3.2 MATLAB Functions and Usage

Complete syntax of each MATLAB function is provided in ***Table 4***, as well as syntax, descriptions and examples.

Provides a list of function return cases.

**Table 4: MATLAB Functions**

| Function | Function syntax and example |
|---|---|
| AO_DefaultStartConnection | **Syntax:**<br>`[Result] = AO_DefaultStartConnection(DspMac)`<br>**Function:**<br>Used to connect MATLAB to Neuro Omega system<br>**Result:**<br>Function return is an integer, 0 = no function errors, other number indicate function error (see **6**)<br>**Function parameters:**<br>■ **DspMac**: String of 6 hex values. This is the mac address of the Neuro Omga system<br>It is preferable to ensure connection was done successfully by calling the function AO_IsConnected<br>**Example:**<br>`DSPMac='00:21:ba:07:ab:9e';`<br>`retStartConnection=AO_DefaultStartConnection(DSPMac);`<br>Add the following code to insure proper connection:<br>`for j=1:100,`<br>` pause(1);`<br>` ret=AO_IsConnected;`<br>` if ret==1`<br>` 'The System is Connected'`<br>` break;`<br>` end`<br>`end`<br>After a successful connection, the PcMac address will appear in the menu, **Help > User Info**, as illustrated in *Figure 3*. |

<table>
<tr><th>Function</th><th>Function syntax and example</th></tr>
<tr><td></td><td><br>Users Info<br><table><tr><th></th><th>User name</th><th>UserID</th><th>MAC address</th></tr><tr><td>0)</td><td>User0</td><td>0</td><td>00:15:17:48:8B:5B</td></tr><tr><td>1)</td><td>User1</td><td>1</td><td>00:01:02: 2:BD:BD</td></tr></table> <br>Figure 3: MAC Addresses Including MATLAB Computer</td></tr>
<tr><td>AO_IsConnected</td><td>Syntax:<br>[Results] = AO_IsConnected()<br>Function:<br>Checks if MATLAB is connected to Neuro Omega<br>Result:<br>Returns 1 if the system is connected, otherwise returns 0<br>Example:<br>for j=1:100,<br> pause(1);<br> ret=AO_IsConnected;<br> if ret==1<br> 'The System is Connected'<br> break;<br> end<br>end</td></tr>
<tr><td>AO_CloseConnection</td><td>Syntax:<br>[Result] = AO_CloseConnection()<br>Function:<br>Used to close connection between MATLAB and the Neuro Omega system<br>Result:</td></tr>
</table>

| Function | Function syntax and example |
|---|---|
|  | Function returns an integer, 0 = no function errors, other number indicate function error (see **6**)<br>**Example:**<br>`Result=AO_CloseConnection();`<br>`if (Result==0)`<br>` display('Connection closed successfully');`<br>`else`<br>` display('Connection close error');`<br>`end` |
| AO_AddBufferingChannel | **Syntax:**<br>`[Result] = AO_AddBufferingChannel(ChannelID,BufferSizemSec)`<br>**Function:**<br>Used to gather data for the channel defined in ChannelID<br>**Result:**<br>Returns an integer, 0 = no function errors, other number indicate function error (see ***6***)<br>**Function parameters:**<br>■ **ChannelID**: The channel ID we want to gather data for<br>■ **BufferSizemSec**: The size of the buffer in mSec.<br>**Notes:**<br>• The function stores the data using First In First Out (FIFO) mechanism.<br>• The data value is A\D value including gain.<br>**Example:**<br>` ChannelID=10256; % set the channel number`<br>`BufferSizemSec = 10000; % set the size of the buffer in mSec`<br>`AO_AddBufferingChannel(ChannelID, BufferSizemSec)% start gathering data for channel 10256` |
| AO_GetAlignedData | **Syntax:**<br>`[Result,pData,DataCapture,TS_FirstSample]= AO_GetAlignedData(ChannelIdArr,ChannelCount)`<br>**Function:**<br>Used to get aligned data for several channels<br>**Result:** |

| Function | Function syntax and example |
|---|---|
| | Returns an integer, 0 = no function errors, other number indicate function error (see **6**)<br>**Function parameters:**<br>▪ **ChannelIdArr**: Array of channels which we need to get data for, all channels must have the same sampling rate<br>▪ **ChannelCount**: Number of channels listed in ChannelIdArr<br>▪ **pData**: Array of data samples of all listed channels, be arranged in single row<br>▪ **DataCapture**: The amount of the useable data captured in the pData array<br>▪ **TS_FirstSample**:The timestamp of the first sample for each channel<br>**Notes:**<br>• In order to get data you need to use AO_AddBufferingChannel first.<br>• In order to get real time data u need to clear the buffered data first using AO_ClearChannelData function before using both commands AO_GetAlignedData, AO_GetChannelData otherwise u will get stored data<br>• pData will contain samples of data for all channels ,the number of valid samples in this array is DataCapture so make sure that you only get DataCapture samples. Hence, number of samples for each channel is DataCapture divided by ChannelCount.<br>• In pData samples are arranged in a single array for all channels, starting with samples of the first channel listed in the ChannelIdArr, followed by other channels consecutively and in the same order.<br>• The data value is A\D including gain<br>**Example:**<br>`ChannelIdArr=[10000,10001,10002];`<br>`ChannelCount=3;`<br>`[Result,pData,DataCapture,TS_FirstSample] =`<br>`AO_GetAlignedData(ChannelIdArr,ChannelCount);% get aligned data from channels:10000,10001,10002 save them in the array pData, the alignment is done by time stamp TS_FirstSample` |
| AO_GetChannelData | **Syntax:**<br>`[Result,pData,DataCapture] = AO_GetChannelData(ChannelId)`<br>**Function:** |

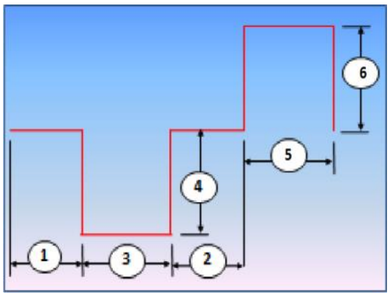
| Function | Function syntax and example |
|---|---|
|  | Used to get data for the specified channel<br>**Result:**<br>Returns an integer, 0 = no function errors, other number indicate function error (see ***6***)<br>**Function parameters:**<br>■ **ChannelId**: The channel id which we want to get data for<br>■ **pData**: Array of data<br>■ **DataCapture**: The amount of the useable data in the array<br>**Note**: The pData will contain a block of data in in the following format, for example:<br>`byte 1-2 : SizeOFtheBlock in words (1 word =2Byte) including the samples in this block`<br>`byte 3 BlockType (in our case alwayes will be 'd' or 100)`<br>`byte 4 Not used`<br>`byte 5-6 ChannelNumber the id of the channel this block belongs to`<br>`byte 7 Unit number ,this value valid only for segmented data`<br>`byte 8 Not used`<br>`byte 9-12 TimeStamp of the first sample of the block you will have to reorder them [byte10 byte9 byte12 byte11]`<br>`byte 13-14 OverFlowCount the over flow of the time stamp – Future use`<br>`byte 15-16 First sample`<br>`byte 17-18 Second sample`<br>In order to calculate the number of samples in this channel, do the following:<br>`HeaderSize=14bytes`<br>`HeadrSizeWord=14bytes/2`<br>`samplescount=SizeOFtheBlock-HeaderSizeWord`<br>` = (SizeOFtheBlock-14)/2`<br>**Example:**<br>`[Result,pData,DataCapture]=AO_GetChannelData(10256);` |

| Function | Function syntax and example |
|---|---|
| AO_ClearChannelData | **Syntax:**<br>`[Result] = AO_ClearChannelData()`<br>**Function:**<br>Used to clear buffered data by command AO_AddBufferingChannel<br>**Result:**<br>Returns an integer, 0 = no function errors, other number indicate function error (see ***6***)<br>**Note**: In order to get real time data you need to clear the buffered data first using AO_ClearChannelData function before using both commands AO_GetAlignedData, AO_GetChannelData otherwise you will get stored data.<br>**Example:**<br>`AO_ClearChannelData()` |
| AO_GetNextBlock | **Syntax:**<br>`[Result,arraydata,realDataSizeWords]= AO_GetNextBlock(sizeOfArrayWords)`<br>**Function:**<br>Used to get the next new block data, the data should be parsed using StreamFormat.h file<br>**Result:**<br>Returns an integer, 0 = no function errors, other number indicate function error (see *6*)<br>**Function parameters:**<br>■ **sizeOfArrayWords**: The max size of data the array can contain<br>■ **arraydata**: Pointer to an array to hold the new data ,the data contain stream format in order to parse the data you need some Knowledge in our stream Format<br>■ **realDataSizeWords**: The count of the data copied to the array data<br>**Note**: StreamFormat.h file is saved in the include directory<br>**Example:**<br>`realDataSizeWords=zeros(1,1);`<br>`[res,arraydata,realDataSizeWords]=AO_GetNextBlock(50000);` |
| AO_SendBlock | **Syntax:**<br>`[Result] = AO_SendBlock(ArrayData)`<br>**Function:** |

| Function | Function syntax and example |
|---|---|
|  | Used to send stream format data to the Neuro Omega system<br>**Result:**<br>Return is an integer, 0 = no function errors, other number indicate function error (see ***6***)<br>**Function parameters:**<br>■ **ArrayData**: Contain the data which will be sent to Neuro Omega system<br>**Notes:**<br>• This function for advanced users only.<br>• Stream format is explained in StreamFormat.h file<br>**Example:**<br>`ArrayData=[7,1,2,3,4,5,6];`<br>`AO_SendBlock(ArrayData)` |
| AO_StartSave | **Syntax:**<br>`[Result] = AO_StartSave();`<br>**Function:**<br>Used to start saving mpx file by the Neuro Omega system<br>**Result:**<br>Returns an integer, 0 = no function errors, other number indicate function error (see ***6***)<br>**Notes:**<br>• The mpx file saved will contain the channels listed on the Data logging Options in the Neuro Omega<br>• The filename and the saving path could be set before saving using MATLAB commands: AO_SetSaveFileName, and AO_SetSavePath.<br>• Or by the parameters defined in the data logging(default)<br>• When saving start the saving button in the Neuro Omega GUI turns to red. See Neuro Omega Manual<br>**Example:**<br>`[Result] = AO_StartSave()% start saving on the Neuro Omeg` |
| AO_SetSaveFileName | **Syntax:**<br>`[Result] = AO_SetSaveFileName(FileName)`<br>**Function:** |

| Function | Function syntax and example |
|---|---|
| | Used to set mpx file name saved by Neuro Omega system<br>**Result:**<br>Function returns an integer, 0 = no function errors, other number Indicate function error (see ***6***)<br>**Function parameters:**<br>■ **FileName**: Contains the file name.<br>**Note**: File name must be less than 30 chars.<br>**Example:**<br>`fileName='TestFile';`<br>`AO_SetSaveFileName(fileName)%set the file name as TestFile`<br>`AO_StartSave()% start saving, the file name will be testFile` |
| AO_SetSavePath | **Syntax:**<br>[Result] = AO_SetSavePath(Path)<br>**Function:**<br>Used to set the path of the directory to save in the mpx file.<br>**Result:**<br>Returns an integer, 0 = no function errors, other number indicates function error (see ***6***)<br>**Function parameters:**<br>■ **Path**: Contain the path of the directory for saving the files<br>**Example:**<br>`path='c:\logging_data\' ;%the path of the directory to save in`<br>`AO_SetSavePath(path)%set the path of the saving to 'c:\logging_data\'`<br>`AO_StartSave()%start saving, the file will be saved at 'c:\logging_data\'` |
| AO_StopSave | **Syntax:**<br>[Result] = AO_StopSave()<br>**Function:**<br>Used to stop saving by Neuro Omega system<br>**Result:**<br>Returns an integer, 0 = no function errors, other number indicate function error (see ***6***)<br>**Example:**<br>`fileName='TestFile’;` |

| Function | Function syntax and example |
|---|---|
| | `path='c:\logging_data\'; ;%`<br>`AO_SetSaveFileName(fileName);`<br>`AO_SetSavePath(path);`<br>`AO_StartSave();%`<br>`pause(100) ;`<br>`AO_StopSave();`<br>Saving will be done for 100 sec, and the mpx file name is TestFile in the **c:\logging_data\**. |
| AO_SendDout | **Syntax:**<br>`[Result] = AO_SendDout(mask, value);`<br>**Function:**<br>Sends output to port number 11701<br>**Result:**<br>Returns an integer, 0 = no function errors, other number indicates function error (see ***6***)<br>**Function parameters**:<br>■ **Mask**: is an 8 bit hex number input as a string. This variable masks the value, any 1 bit number changes the corresponding bit to the number in value. A 0 bit will leave the port unchanged<br>■ **Value**: the value to insert the digital channel. The value can be any number between 0 and 2^8-1<br>**Notes:** Only lower case characters can be used in the mask<br>**Example:**<br>`mask='0xFF'; %the mask (11111111)`<br>`value=0; %the value (00000000)`<br>`Result = AO_SendDOut(mask,value); %Initialize all bits of port 11701 to 0`<br>`mask='0x05'; %the mask (00000101)`<br>`value=3; %the value (00000011)`<br>`Result = AO_SendDOut(mask,value); %set port 11701`<br>`%====> The output of the bits on port 11701 will be '0000 0001'`<br>`Mask=00000101`<br>`Value=00000011`<br>`Port= 00000001` |
| AO_StartDigitalStimulation | **Syntax:** |

| Function | Function syntax and example |
|---|---|
|  | `[Result] = AO_ StartDigitalStimulation(StimChannel, FirstPhaseDelay_mS, FirstPhaseAmpl_mA, FirstPhaseWidth_mS, SecondPhaseDelay_mS, SecondPhaseAmpl_mA, SecondPhaseWidth_mS, Freq_hZ, Duration_sec, ReturnChannel);`<br>**Function:**<br>Function used to set the parameters and start stimulation using Neuro Omega system for the specified StimChannel<br>**Resuls:**<br>Returns an integer, 0 = no function errors, other number indicate function error (see ***6***)<br>**Function parameters:**<br>See *Figure 4* for an illustration of the stimulation parameters.<br><br> <br>**Figure 4: Stimulation Parameters Illustration**<br>- **StimChannel**: The channel we want to start stimulation on<br>- **FirstPhaseDelay_mS**: First phase delay in mSec (1)<br>- **FirstPhaseAmpl_mA**: First phase amplitude (4)<br>- **FirstPhaseWidth_mS** : The width of the first phase (3)<br>- **SecondPhaseDelay_mS**: Second phase delay in mSec (2)<br>- **SecondPhaseAmpl_mA**: Second phase amplitude (6)<br>- **SecondPhaseWidth_mS**: Second phase width (5)<br>- **Freq_hZ**: The stimulation frequency<br>- **Duration_sec**: Duration of the stimulation after which stimulation stops<br>- **ReturnChannel**: The ID of the channel we want to return the stimulation with(set -1 for Global return)<br>**Note:** This function should be called before starting stimulation, otherwise stimulation will be done using the parameters defined in the SW GUI<br>**Example:** |

| Function | Function syntax and example |
|---|---|
| | StimChannel=10000;%the channel we want to start stimualtion in<br>FirstPhaseDelay_mS=1.1;%the delay of the first phase<br>FirstPhaseAmpl_mA=-3.5;%the amp of the first phase<br>FirstPhaseWidth_mS=0.5;%the width of the first phase<br>SecondPhaseDelay_mS=1.5;%the delay of the second phase<br>SecondPhaseAmpl_mA=1.5;%the amp of the second phase<br>SecondPhaseWidth_mS=0.2;%the width of the second phase<br>Freq_hZ=10;%the frequency of the stimulation<br>Duration_sec=30;%duration of the stimulation<br>ReturnChannel=10001;%the ID of the channel we want to return the stimulation with<br>AO_StartDigitalStimulation(StimChannel,FirstPhaseDelay_mS,FirstPhaseAmpl_mA,FirstPhaseWidth_mS,SecondPhaseDelay_mS,SecondPhaseAmpl_mA,SecondPhaseWidth_mS,Freq_hZ,Duration_sec,ReturnChannel);%set stimulation params and start stimulation |
| AO_StopStimulation | **Syntax:**<br>[Results]=AO_StopStimulation(ChannelNumber);<br>**Function:**<br>Used to stops stimulation to the stimulation source of the ChannelNumber<br>**Results:**<br>Returns an integer, 0 = no function errors, other number indicates function error (see ***6***)<br>**Function parameters:**<br>■ **ChannelNumber**: The ID of the channel used for stimulation<br>**Note:** In order to stop stimulation in all channels use: ChannelNumber= -1<br>**Example:**<br>ChannelNumber=10000;<br>AO_StopStimulation(ChannelNumber); |
| AO_LoadWaveToEmbedded | **Syntax:**<br>[Results]=AO_ LoadWaveToEmbedded (pSource, SamplesCount, downSampleFactor, waveName);<br>**Function:**<br>Used to take an analog wave and load it into the Neuro Omega system |

| Function | Function syntax and example |
|---|---|
| | **Results:**<br>Returns an integer, 0 = no function errors, other number indicates function error (see ***6***)<br>**Function parameters:**<br>■ **pSource**: array of data<br>■ **SamplesCount**: number of samples (less than 8 milion)<br>■ **downSampleFactor**: must be 2^N = 1, 2, 4, 8, 16<br>■ **waveName**: The name of the wave<br>**Example:**<br>`x=[0:0.1:180]`<br>`pSource = sin(x);`<br>`SamplesCount = 1500;`<br>`downSampleFactor = 2 ;`<br>`waveName = 'sin_wave';`<br>`[ Result ] = AO_LoadWaveToEmbedded( pSource, SamplesCount, downSampleFactor, waveName )` |
| AO_StartAnalogStimulation | **Syntax:**<br>[Results]=AO_ StartAnalogStimulation (StimChannel, waveId, Freq_Hz, Duration_sec, ReturnChannel);<br>**Function:**<br>Used to set the parameters for the analog stimulation and start stimulation on the specified channel<br>**Results:**<br>Returns an integer, 0 = no function errors, other number indicates function error (see ***6***)<br>**Function parameters:**<br>■ **StimChannel**: the channel we want to start stimulation on<br>■ **waveId**: The id of the wave<br>■ **Freq_Hz:** the frequency of the stimulation<br>■ **Duration_sec**: duration of the stimulation<br>■ **ReturnChannel**: The ID of the channel we want to return the stimulation with (set -1 for Global return)<br>**Example:**<br>`StimChannel= 10256;`<br>`waveId= 1;`<br>`Freq_hZ=10; %the frequncy of the stimulation`<br>`Duration_sec=30 ;%duration of the stimulation`<br>`ReturnChannel=-1;` |

| Function | Function syntax and example |
|---|---|
| | `[Result] = AO_StartAnalogStimulation(StimChannel,waveId,Freq_Hz , Duration_sec, ReturnChannel )` |
| AO_GetLatestTimeStamp | **Syntax:**<br>[Results] = AO_GetLatestTimeStamp()<br>**Function:**<br>Used to get the last time stamp<br>**Results:**<br>Returns an integer, 0 = no function errors, other number indicates function error (see ***6***)<br>**Examples:**<br>`[Result , lastTS] = AO_GetLatestTimeStamp( )` |
| AO_TranslateNameToID | **Syntax:**<br>[Results] = AO_TranslateNameToID( ChannelName , nameLength )<br>**Function:**<br>Used to translate the name of the channel to his ID<br>**Results:**<br>Returns an integer, 0 = no function errors, other number indicates function error (see *6*)<br>**Function parameters:**<br>■ **ChannelName:** the name of the channel<br>■ **nameLength**: the length of the name we want to translate<br>**Examples:**<br>`[Result ,channelID] = AO_TranslateNameToID('LPF_01');` |
| AO_SetChannelSaveState | **Syntax:**<br>[Results] = AO_ SetChannelSaveState(channelID,stateSave)<br>**Function:**<br>Used to Check the checkbox in the Neuro Omega Gui in v if the statesave=1 is on or unchecked if the statesave=0 is off<br>**Results:**<br>Returns an integer, 0 = no function errors, other number indicates function error (see ***6***)<br>**Function parameters:** |

| Function | Function syntax and example |
|---|---|
| | ■ **channelID:** the id of the channel<br>■ **stateSave:** 1 is on, 0 is off<br>**Examples:**<br>`for statesave =TRUE we will check the check box in v`<br>`channelID=10256`<br>`stateSave = 1;`<br>`[ Result ] = AO_SetChannelSaveState( channelID ,`<br>`stateSave)`<br>`for statesave =FALSE we will check the check box in`<br>`v`<br>`channelID=10256`<br>`stateSave = 0;`<br>`[ Result ] = AO_SetChannelSaveState( channelID ,`<br>`stateSave)` |
| AO_SendDigitalData | **Syntax:**<br>[Results] = AO_ SendDigitalData (DigitalChannelNumber, mask, value)<br>**Function:**<br>Used to sends digital data for specific Internal port ID<br>**Results:**<br>Returns an integer, 0 = no function errors, other number indicates function error (see ***6***)<br>**Function parameters:**<br>■ **DigitalChannelNumber**: the ID of the Internal port<br>■ **Mask**: is an 8 bit hex number input as a string. This variable masks the value, any 1 bit number changes the corresponding bit to the number in value. A 0 bit will leave the port unchanged<br>■ **Value**: the value to insert the digital channel. The value can be any number between 0 and 2^8-1<br>**Notes:** Only lower case characters can be used in the mask<br>**Examples:**<br>`DigtalChannelNumber=11230 ;channel ID`<br>`mask='0x00';  %the mask`<br>`value=0;      %the value`<br>`Result =`<br>`AO_SendDigitalData(DigtalChannelNumber,mask,value);`<br>`%Initialize port 11230 to 0`<br>`mask='0x05';  %the mask`<br>`value=3;      %the value`<br>`Result =`<br>`AO_SendDigitalData(DigtalChannelNumber,mask,value);`<br>`%set port 11230`<br>` %====> The output of the bits on port 11230 will be`<br>`'0000 0001'` |

| Function | Function syntax and example |
|---|---|
| | |
| AO_GetDriveDepth | **Syntax:**<br>[Results, Depth] = AO_ GetDriveDepth ()<br>**Function:**<br>Used to get the drive position<br>**Results:**<br>Returns an integer, 0 = no function errors, other number indicates function error (see ***6***)<br>**Function parameters**:<br>■ **Depth:** the depth of the drive in um (microMeter**)**<br>**Examples:**<br>`[Result , Depth] =  AO_GetDriveDepth( );` |
| AO_SetThreshold | **Syntax:**<br>[Results] = AO_ SetThreshold (ChannelID, ThresholdValue_uVolt, Direction)<br>**Function:**<br>Used to set the thresh hold (level line) of a channel<br>**Results:**<br>Returns an integer, 0 = no function errors, other number indicates function error (see ***6***)<br>**Function parameters**:<br>■ **channelID:** contain the chanel id can be only SPIKE LFP OR segmented of the same electrode<br>■ **ThreshHoldValue_uVolt:** the value of the level line in microVolts<br>■ **Direction** : the detection 1->down 2->up<br>**Examples:**<br>`ChannelID = 10256;`<br>`ThresholdValue = 100;`<br>`Direction =  1;`<br>`[Result] =  AO_SetThreshold( ChannelID , ThresholdValue , Direction );` |
| AO_SendTextEvent | **Syntax:**<br>[Results] = AO_ SendTextEvent (text)<br>**Function:**<br>Used to send text to the mpx file with the current time stamp<br>**Results:** |

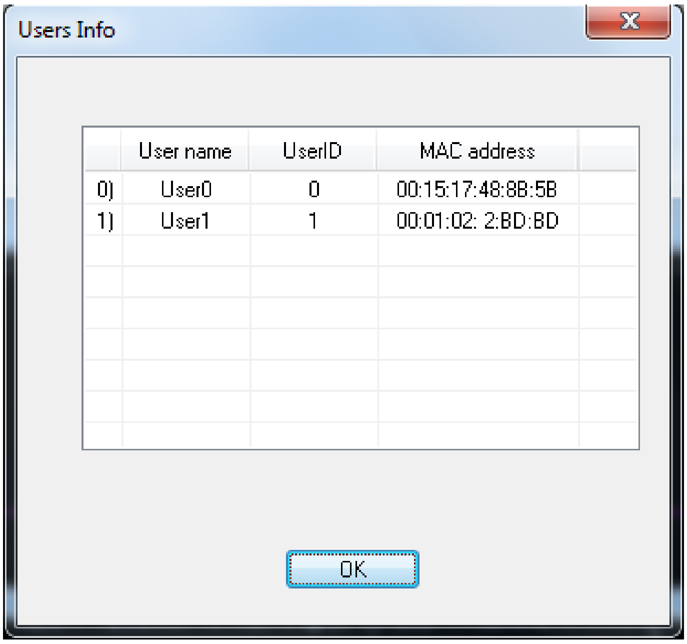
| Function | Function syntax and example |
|---|---|
| | Returns an integer, 0 = no function errors, other number indicates function error (see *6*)<br>**Function parameters:**<br>■ **Text:** array of chars<br>**Examples:**<br>`text = 'the text is in the mpx file';`<br>`[ Result ] = AO_SendTextEvent( text );` |
| AO_CheckConnectionQuality | **Syntax:**<br>[Results qualityType, qualityPercent ] = AO_CheckConnectionQuality()<br>**Function:**<br>Used to check the quality of the connection of the system if its poor, medium or high<br>**Results:**<br>Returns an integer, 0 = no function errors, other number indicates function error (see *6*)<br>**Function parameters:**<br>■ **qualityType:** poor, medium or high<br>■ **qualityPercent:** the rate of the quality connection in percent<br>**Examples:**<br>`[ Result, qualityType,  qualityPercent ] = AO_CheckConnectionQuality( );` |
| AO_GetAllChannels | **Syntax:**<br>[Results ,channelsData ] = AO_ GetAllChannels (ChannelCount)<br>**Function:**<br>Used to return all the channels with their name and id<br>**Results:**<br>Returns an integer, 0 = no function errors, other number indicates function error (see *6*)<br>**Function parameters:**<br>■ **ChannelCount:** number of channels the user want to see<br>■ **channelsData:** a struct that include all the channels with two fields: name and id<br>**Examples:**<br>`ChannelCount=220;`<br>`[ Result , channelsData ] = AO_GetAllChannels(ChannelCount )` |
| AO_GetCutOffFC | **Syntax:** |

| Function | Function syntax and example |
|---|---|
|  | [Results, FCLP , FCHP ] = AO_GetCutOffFC( ChannelID )<br>**Function:**<br>Used to return the cut-off frequency of the low and high pass filter<br>**Results:**<br>Returns an integer, 0 = no function errors, other number indicates function error (see ***6***)<br>**Function parameters:**<br>■ **channelID:** the id of the channel<br>■ **FCLP:** will contain the cut-off frequency of the low pass filter<br>■ **FCHP:** will contain the cut-off frequency of the high pass filter<br>**Examples:**<br>`ChannelID = 10256;`<br>`[ Result, FCLP , FCHP] = AO_GetCutOffFC( ChannelID )` |
| AO_SetChannelName | **Syntax:**<br>[Results] = AO_SetChannelName( ChannelID, newName )<br>**Function:**<br>Used to set a new name for the specified channel<br>**Results:**<br>Returns an integer, 0 = no function errors, other number indicates function error (see ***6***)<br>**Function parameters:**<br>■ **channelID:** the id of the channel<br>■ **newName:** array of chars with the new name for the channel<br>**Examples:**<br>`ChannelId = 10258 ;`<br>`newName = 'left Side';`<br>`[Result] = AO_SetChannelName(ChannelId , newName);` |
| AO_GetChannelSaveState | **Syntax:**<br>[Results, status] = AO_ GetChannelSaveState ( ChannelId)<br>**Function:**<br>Used to get the save status of the specified channel<br>**Results:**<br>Returns an integer, 0 = no function errors, other number indicates function error (see ***6***)<br>**Function parameters:**<br>■ **channelId:** the id of the channel<br>■ **status:** 1 if the save state is true, 0 if the save state is false |

| Function | Function syntax and example |
|---|---|
|  | **Examples:**<br>`ChannelId = 10256;`<br>`[ Result , status ] = AO_GetChannelSaveState(`<br>`ChannelId )` |
| AO_GetStopMotorTS | **Syntax:**<br>[Results, StopMotorTS] = AO_ GetStopMotorTS ()<br>**Function:**<br>Used to get the last time stamp when the motor stopped moving<br>**Results:**<br>Returns an integer, 0 = no function errors, other number indicates function error (see ***6***)<br>**Function parameters:**<br>■ **StopMotorTS:** the last time stamp when the motor stopped<br>**Examples:**<br>`[Result , StopMotorTS] = AO_GetStopMotorTS()` |
| AO_GetChannelsInformation | **Syntax:**<br>information = AO_ GetChannelsInformation ()<br>**Function:**<br>Used to return struct that contain all the information about the channels: name, id , LP frequency, HP frequency and save state<br>**Results:**<br>Returns an integer, 0 = no function errors, other number indicates function error (see ***6***)<br>**Function parameters:**<br>■ **information:** struct of information about the channels have 5 fields: name , id , LP frequency, HP frequency and save state<br>**Examples:**<br>`information = AO_ GetChannelsInformation ()` |

### 2.3.3 C++ Functions and Usage

| Function | Function syntax and example |
|---|---|
| DefaultStartConnection | **Syntax:**<br>`int DefaultStartConnection(MAC_ADDR*core_macAdd , AOParseFunction ) ;`<br>**Function:**<br>Used to connect C++ to Neuro Omega system<br>**Result**:<br>Function return is an integer, 0 = no function errors, other number indicate function error (see ***6***)<br>**Function parameters:**<br>■ **DspMac**: String of 6 hex values. This is the mac address of the Neuro Omga system<br>■ **AOParseFunction**: pointer to function which will be called when new data received from the embedded.<br>It is preferable to ensure connection was done successfully by calling the function isConnected<br>**Example:**<br>`MAC_ADDR dsp;`<br>`dsp.addr[0]=0xbc;`<br>`dsp.addr[1]=0x6a;`<br>`dsp.addr[2]=0x29;`<br>`dsp.addr[3]=0xe1;`<br>`dsp.addr[4]=0x49;`<br>`dsp.addr[5]=0xbf;`<br>`retStartConnection=DefaultStartConnection(&dsp , 0);`<br>Add the following code to insure proper connection:<br>`while(isConnected()==FALSE){` |

<table>
<tr><th>Function</th><th>Function syntax and example</th></tr>
<tr><td></td><td>        AOSLEEP_MSEC(1);<br>}<br>             }<br>printf("\n Connect = %d \n" , connect);<br><br>After a successful connection, the PcMac address will appear in the menu, **Help** > **User Info**, as illustrated in below.<br><br>Users Info<br><table><tr><th></th><th>User name</th><th>UserID</th><th>MAC address</th></tr><tr><td>0)</td><td>User0</td><td>0</td><td>00:15:17:48:8B:5B</td></tr><tr><td>1)</td><td>User1</td><td>1</td><td>00:01:02: 2:BD:BD</td></tr></table><br>**Figure 5: MAC Addresses Including MATLAB Computer**</td></tr>
<tr><td>isConnected</td><td>**Syntax:**<br>int isConnected();<br>**Function:**<br>Checks if C++  is connected to Neuro Omega<br>**Result:**<br>Returns 1 if the system is connected, otherwise returns 0<br>**Example:**<br>while(isConnected()==FALSE){<br>        AOSLEEP_MSEC(1);<br>}<br>             }<br>printf("\n Connect = %d \n" , connect);</td></tr>
<tr><td>CloseConnection</td><td>**Syntax:**<br>int CloseConnection()<br>**Function:**</td></tr>
</table>

| Function | Function syntax and example |
|---|---|
|  | Used to close connection between C++ and the Neuro Omega system<br>**Result:**<br>Function returns an integer, 0 = no function errors, other number indicate function error (see ***6***)<br>**Example:**<br>`Result=CloseConnection();`<br>`if (Result==0)`<br>` printf("\n Connection closed successfully\n");`<br>`else`<br>` printf("\n Connection close error\n");` |
| AddBufferingChannel | **Syntax:**<br>`int AddBufferingChannel(int ChannelID,int BufferSizemSec)`<br>**Function:**<br>Used to gather data for the channel defined in ChannelID<br>**Result:**<br>Returns an integer, 0 = no function errors, other number indicate function error (see ***6***)<br>**Function parameters:**<br>■ **ChannelID**: The channel ID we want to gather data for<br>■ **BufferSizemSec**: The size of the buffer in mSec.<br>**Notes:**<br>• The function stores the data using First In First Out (FIFO) mechanism.<br>• The data value is A\D value including gain.<br>**Example:**<br>`ChannelID=10256; // set the channel number`<br>`BufferSizemSec = 10000; // set the size of the buffer in mSec`<br>`AddBufferingChannel(ChannelID, BufferSizemSec) //start gathering data for channel 10256` |
| GetAlignedData | **Syntax:**<br>`int GetAlignedData(int16* pArray, int ArraySize, int* actualData, int arrChannel,int sizearrChannels, ULONG* TS_Begin);`<br>**Function:** |

| Function | Function syntax and example |
|---|---|
|  | Used to get aligned data for several channels<br>**Result:**<br>Returns an integer, 0 = no function errors, other number indicate function error (see ***6***)<br>**Function parameters:**<br>■ **pArray: this array must be allocatd by the user and the function will insert the data into it**<br>■ **ArraySize: the size of the array in words**<br>■ **actualData: the actual data of the amount of data we inserted into the pArray**<br>■ **arrChannel: contain the list of channel we want to collect data for**<br>■ **sizearrChannels: the count of the channel**<br>■ **TS_Begin: the timestamp of the first sample**<br>**Notes:**<br>■ In order to get data you need to use the **AddBufferingChannel** first<br>■ note that the function gets the data in FIFO , so at the beginning you get the data **stored** by the buffering then u start getting a real time data<br>■ the data in the array will be sorted like the channels ,e.g. if the channel are 10000,10001,10002 then the data will be ,data for channel 10000,data for channel 10001,data for channel 10002. the amount of data for each channel Must be the same == actualData/sizearrChannels<br>**Example:**<br>`int16* pArray = new int16[100000];`<br>`int ArraySize = 10000;`<br>`int actualData =0;`<br>`int arrChannel[]={10000,10001,10002};`<br>`int sizearrChannels = 3;`<br>`ULONG TS_Begin = 0;`<br>`AddBufferChannel(10000,10000);`<br>`AddBufferChannel(10001,10000);`<br>`AddBufferChannel(10002,10000);`<br>`AOSLEEP_MSEC(10000);`<br>`GetAlignedData( pArray, ArraySize, &actualData, arrChannel, sizearrChannels, TS_Begin); //get aligned data from channels:10000,10001,10002 save them in the array pData, the alignment is done by time stamp TS_Begin`<br>`printf("\n %d \n" , ActualData);` |

| Function | Function syntax and example |
|---|---|
|  | `printf("\n %d \n" , TS_Begin);` |
| GetChannelData | **Syntax:**<br>`int GetChannelData(int ChannelsId,int16* pData,int ArrSizeWords,int *DataCapture);`<br>**Function:**<br>Used to get data for the specified channel<br>**Result:**<br>Returns an integer, 0 = no function errors, other number indicate function error (see ***6***)<br>**Function parameters:**<br>■ **ChannelId**: The channel id which we want to get data for<br>■ **pData**: Array of data<br>■ **ArrSize**: the size of the pData<br>■ **DataCapture**: The amount of the useable data in the array<br>**Note**: The pData will contain a block of data in in the following format, for example:<br>`byte 1-2 : SizeOFtheBlock in words (1 word =2Byte) including the samples in this block`<br>`byte 3 BlockType (in our case alwayes will be 'd' or 100)`<br>`byte 4 Not used`<br>`byte 5-6 ChannelNumber the id of the channel this block belongs to`<br>`byte 7 Unit number ,this value valid only for segmented data`<br>`byte 8 Not used`<br>`byte 9-12 TimeStamp of the first sample of the block you will have to reorder them [byte10 byte9 byte12 byte11]`<br>`byte 13-14 OverFlowCount the over flow of the time stamp - Future use`<br>`byte 15-16 First sample`<br>`byte 17-18 Second sample`<br>In order to calculate the number of samples in this channel, do the following:<br>`HeaderSize=14bytes`<br>`HeadrSizeWord=14bytes/2`<br>`samplescount=SizeOFtheBlock-HeaderSizeWord`<br>` = (SizeOFtheBlock-14)/2` |

| Function | Function syntax and example |
|---|---|
|  | **Example:**<br>`int ChannelsId=10256;`<br>`int16* pData=new int16[100000];`<br>`int ArrSizeWords = 10000;`<br>`int DataCapture=0;`<br>`AddBufferChannel(10256,10000);`<br>`GetChannelData(ChannelsId, pData, ArrSizeWords, &DataCapture);` |
| ClearBuffers | **Syntax:**<br>`Void ClearBuffers();`<br>**Function:**<br>Used to clear all the data from the buffers<br>**Example:**<br>`ClearBuffers ();` |
| GetNextBlock | **Syntax:**<br>`void GetNextBlock(int16 * arraydata,int sizeOfArrayWords,int* realDataSizeWords)`<br>**Function:**<br>Used to get the next new block data, the data should be parsed using StreamFormat.h file<br>**Function parameters:**<br>■ **arraydata**: Pointer to an array to hold the new data ,the data contain stream format in order to parse the data you need some Knowledge in our stream Format<br>■ **sizeOfArrayWords**: The max size of data the array can contain<br>■ **realDataSizeWords**: The count of the data copied to the array data<br>**Note**: StreamFormat.h file is saved in the include directory<br>**Example:**<br>`int16 * arraydata = new int16[45000];`<br>`int sizeOfArrayWords = 45000;`<br>`int realDataSizeWords = 0;`<br>`GetNextBlock(arraydata, sizeOfArrayWords, &realDataSizeWords);` |
| AO_SendBlock | **Syntax:** |

| Function | Function syntax and example |
|---|---|
|  | `int SendBlock(void* streamBlock);`<br>**Function:**<br>Used to send stream format data to the Neuro Omega system<br>**Result:**<br>Return is an integer, 0 = no function errors, other number indicate function error (see ***6***)<br>**Function parameters:**<br>■ **streamBlock**: Contain the block of data which will be sent to the Neuro Omega system<br>**Notes:**<br>• This function for advanced users only.<br>• Stream format is explained in StreamFormat.h file |
| StartSave | **Syntax:**<br>`int StartSave();`<br>**Function:**<br>Used to start saving mpx file by the Neuro Omega system<br>**Result:**<br>Returns an integer, 0 = no function errors, other number indicate function error (see ***6***)<br>**Notes:**<br>• The mpx file saved will contain the channels listed on the Data logging Options in the S Neuro Omega<br>• The filename and the saving path could be set before saving using MATLAB commands: SetSaveFileName, and SetSavePath.<br>• Or by the parameters defined in the data logging(default)<br>• When saving start the saving button in the Neuro Omega GUI turns to red. See Neuro Omega Manual<br>**Example:**<br>`StartSave() //start saving on the Neuro Omeg` |
| SetSaveFileName | **Syntax:**<br>`int SetSaveFileName(cChar* fileName, int size);`<br>**Function:**<br>Used to set mpx file name saved by Neuro Omega system |

| Function | Function syntax and example |
|---|---|
| | **Result:**<br>Function returns an integer, 0 = no function errors, other number Indicate function error (see ***6***)<br>**Function parameters:**<br>■ **fileName**: Contains the file name.<br>■ **size**: the size of the name of the file<br>**Note**: File name must be less than 30 chars.<br>**Example:**<br>`cChar fileName[50];`<br>`strncpy(fileName , "TestFile ",50);`<br>`SetSaveFileName(fileName , 10);`<br>`StartSave(); //start saving, the file name will be testFile` |
| SetSavePath | **Syntax:**<br>`int SetSavePath(cChar* Path , int size);`<br>**Function:**<br>Used to set the path of the directory to save in the mpx file.<br>**Result:**<br>Returns an integer, 0 = no function errors, other number indicates function error (see ***6***)<br>**Function parameters:**<br>■ **Path**: Contain the path of the directory for saving the files<br>■ **size**: the size of the path name<br>**Example:**<br>` strncpy (path , " c:\logging_data\ ", 50); //the path of the directory to save in`<br>`SetSavePath(path , 50);`<br>`StartSave(); //start saving, the file will be saved at 'c:\logging_data\'` |
| StopSave | **Syntax:**<br>`int StopSave();`<br>**Function:**<br>Used to stop saving by Neuro Omega system<br>**Result:**<br>Returns an integer, 0 = no function errors, other number indicate function error (see ***6***) |

| Function | Function syntax and example |
|---|---|
| | **Example:**<br>`StopSave();` |
| SendDout | **Syntax:**<br>`int SendDout(mask, value);`<br>**Function:**<br>Sends output to port number 11701<br>**Result:**<br>Returns an integer, 0 = no function errors, other number indicates function error (see ***6***)<br>**Function parameters:**<br>■ **Mask:** is an 8 bit hex number input as a string. This variable masks the value, any 1 bit number changes the corresponding bit to the number in value.  A 0 bit will leave the port unchanged<br>■ **Value:** the value to insert the digital channel. The value can be any number between 0 and 2^8-1<br>**Example:**<br>`mask = 1;`<br>`value1 = 1;`<br>`value2 = 0;`<br>`SendDout(mask, value1); //up`<br>`AOSLEEP_MSEC(10000);`<br>`SendDout(mask, value2); //down` |
| StartDigitalStimulation | **Syntax:**<br>`int StartDigitalStimulation(StimChannel, FirstPhaseDelay_mS, FirstPhaseAmpl_mA, FirstPhaseWidth_mS, SecondPhaseDelay_mS, SecondPhaseAmpl_mA, SecondPhaseWidth_mS, Freq_hZ, Duration_sec, ReturnChannel);`<br>**Function:**<br>Function used to set the parameters and start stimulation using Neuro Omega system for the specified StimChannel<br>**Resuls:**<br>Returns an integer, 0 = no function errors, other number indicate function error (see ***6***)<br>**Function parameters:**<br>See below for an illustration of the stimulation parameters. |

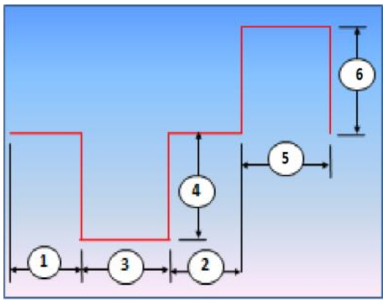


**Figure 6: Stimulation Parameters Illustration**

- **StimChannel**: The channel we want to start stimulation on
- **FirstPhaseDelay_mS**: First phase delay in mSec (1)
- **FirstPhaseAmpl_mA**: First phase amplitude (4)
- **FirstPhaseWidth_mS** : The width of the first phase (3)
- **SecondPhaseDelay_mS**: Second phase delay in mSec (2)
- **SecondPhaseAmpl_mA**: Second phase amplitude (6)
- **SecondPhaseWidth_mS**: Second phase width (5)
- **Freq_hZ**: The stimulation frequency
- **Duration_sec**: Duration of the stimulation after which stimulation stops
- **ReturnChannel**: The ID of the channel we want to return the stimulation with(set -1 for Global return)

**Note:** This function should be called before starting stimulation, otherwise stimulation will be done using the parameters defined in the SW GUI

**Example:**

```
StimChannel=10000;%the channel we want to start
stimualtion in
FirstPhaseDelay_mS=1.1;
FirstPhaseAmpl_mA=-3.5;
FirstPhaseWidth_mS=0.5;
SecondPhaseDelay_mS=1.5;
SecondPhaseAmpl_mA=1.5;
SecondPhaseWidth_mS=0.2;
Freq_hZ=10;
Duration_sec=30;
ReturnChannel=10001;
StartDigitalStimulation(StimChannel,FirstPhaseDelay_
mS,FirstPhaseAmpl_mA,FirstPhaseWidth_mS,SecondPhaseD
elay_mS,SecondPhaseAmpl_mA,SecondPhaseWidth_mS,Freq_
hZ,Duration_sec,ReturnChannel);
```

| Function | Function syntax and example |
|---|---|
| | |
| StopStimulation | **Syntax:**<br>`int StopStimulation(int ChannelNumber);`<br>**Function:**<br>Used to stops stimulation to the stimulation source of the ChannelNumber<br>**Results:**<br>Returns an integer, 0 = no function errors, other number indicates function error (see ***6***)<br>**Function parameters:**<br>■ **ChannelNumber**: The ID of the channel used for stimulation<br>**Note:** In order to stop stimulation in all channels use: ChannelNumber= -1<br>**Example:**<br>`ChannelNumber=10000;`<br>`StopStimulation(ChannelNumber);` |
| LoadWaveToEmbedded | **Syntax:**<br>`int LoadWaveToEmbedded(short* pSource,int SamplesCount,int downSampleFactor,char* waveName)`<br>**Function:**<br>Used to take an analog wave and load it into the Neuro Omega system<br>**Results:**<br>Returns an integer, 0 = no function errors, other number indicates function error (see ***6***)<br>**Function parameters:**<br>■ **pSource**: array of data<br>■ **SamplesCount**: number of samples (less than 8 milion)<br>■ **downSampleFactor**: must be 2^N = 1, 2, 4, 8, 16<br>■ **waveName**: The name of the wave<br>**Example:**<br>`for(int i=0; i<180 ; i++)`<br>`{`<br>`    pSource[i] = sin(i);`<br>`}`<br>`SamplesCount = 150;`<br>`downSampleFactor = 2 ;` |

| Function | Function syntax and example |
|---|---|
| | `strncpy(waveName , " sin_wave " , 10);`<br>`LoadWaveToEmbedded( pSource, SamplesCount, downSampleFactor, waveName );` |
| StartAnalogStimulation | **Syntax:**<br>`int StartAnalogStimulation (StimChannel, waveId, Freq_Hz, Duration_sec, ReturnChannel);`<br>**Function:**<br>Used to set the parameters for the analog stimulation and start stimulation on the specified channel<br>**Results:**<br>Returns an integer, 0 = no function errors, other number indicates function error (see ***6***)<br>**Function parameters:**<br>■ **StimChannel**: the channel we want to start stimulation on<br>■ **waveId**: The id of the wave<br>■ **Freq_Hz:** the frequency of the stimulation<br>■ **Duration_sec**: duration of the stimulation<br>■ **ReturnChannel**: The ID of the channel we want to return the stimulation with (set -1 for Global return)<br>**Example:**<br>`StimChannel= 10256;`<br>`waveId= 1;`<br>`Freq_hZ=10;`<br>`Duration_sec=30;`<br>`ReturnChannel=-1;`<br>`StartAnalogStimulation(StimChannel,waveId,Freq_Hz, Duration_sec, ReturnChannel )` |
| GetLatestTimeStamp | **Syntax:**<br>`int GetLatestTimeStamp(ULONG* plastTS);`<br>**Function:**<br>Used to get the last time stamp<br>**Results:**<br>Returns an integer, 0 = no function errors, other number indicates function error (see ***6***)<br>**Examples:**<br>`ULONG* plastTS = 0;`<br>`GetLatestTimeStamp(&plastTS );`<br>`printf(" \n %d \n " , plastTS);` |

| Function | Function syntax and example |
|---|---|
| | |
| TranslateNameToID | **Syntax:**<br>`int TranslateNameToID( cChar* ChannelName , int nameLength , int channelID );`<br>**Function:**<br>Used to translate the name of the channel to his ID<br>**Results:**<br>Returns an integer, 0 = no function errors, other number indicates function error (see ***6***)<br>**Function parameters**:<br>■ **ChannelName:** the name of the channel<br>■ **nameLength**: the length of the name we want to translate<br>■ **channel**ID**:** the id of the channel after translation<br>**Examples:**<br>`cChar channelName[10];`<br>`int nameLength = 10;`<br>`int channelID = 0;`<br>`strncpy(channelName, "RAW 01" , 10);`<br>`TranslateNameToID(channelName, nameLength, &channelID );` |
| SetChannelSaveState | **Syntax:**<br>`int SetChannelSaveState(int channelID,BOOL BState)`<br>**Function:**<br>Used to Check the checkbox in the Neuro Omega Gui in v if the statesave=1 is on or unchecked if the statesave=0 is off<br>**Results:**<br>Returns an integer, 0 = no function errors, other number indicates function error (see ***6***)<br>**Function parameters**:<br>■ **channelID:** the id of the channel<br>■ **BState:** TRUE is on, FALSE is off<br>**Examples:**<br>`BState =TRUE`<br>`channelID=10256`<br>`SetChannelSaveState( channelID , BState)`<br><br>`channelID=10256`<br>`BState = FALSE;`<br>`SetChannelSaveState( channelID , BState)` |

| Function | Function syntax and example |
|---|---|
| AO_SendDigitalData | **Syntax:**<br>`int SendDigitalData (int DigitalChannelNumber, int mask, int value)`<br>**Function:**<br>Used to sends digital data for specific Internal port ID<br>**Results:**<br>Returns an integer, 0 = no function errors, other number indicates function error (see ***6***)<br>**Function parameters**:<br>■ **DigitalChannelNumber**: the ID of the Internal port<br>■ **Mask:** is an 8 bit hex number input as a string. This variable masks the value, any 1 bit number changes the corresponding bit to the number in value.  A 0 bit will leave the port unchanged<br>■ **Value:** the value to insert the digital channel. The value can  be any number between 0 and 2^8-1<br>**Examples:**<br>`DigitalChannelNumber = 11230;`<br>`mask = 1;`<br>`value1 = 1;`<br>`value2 = 0;`<br>`SendDout(DigitalChannelNumber ,mask, value1); //up`<br>`AOSLEEP_MSEC(10000);`<br>`SendDout(DigitalChannelNumber ,mask, value2); //down` |
| GetDriveDepth | **Syntax:**<br>`int GetDriveDepth (int* nDepth);`<br>**Function:**<br>Used to get the drive position<br>**Results:**<br>Returns an integer, 0 = no function errors, other number indicates function error (see ***6***)<br>**Function parameters**:<br>■ **nDepth:** the depth of the drive in um (microMeter**)**<br>**Examples:**<br>`int* nDepth = 0;`<br>`GetDriveDepth( &nDepth );`<br>`printf("Depth = %d\n" , nDepth);` |
| SetThreshold | **Syntax:**<br>`int SetThreshold (int channelID, int ThresholdValue_uVolt, int Direction);`<br>**Function:**<br>Used to set the thresh hold (level line) of a channel<br>**Results:** |

| Function | Function syntax and example |
|---|---|
| | Returns an integer, 0 = no function errors, other number indicates function error (see ***6***)<br>**Function parameters**:<br>■ **channelID:** contain the chanel id can be only SPIKE LFP OR segmented of the same electrode<br>■ **ThreshHoldValue_uVolt:** the value of the level line in microVolts<br>■ **Direction** : the detection 1->down 2->up<br>**Examples:**<br>`ChannelID = 10256;`<br>`ThresholdValue = 100;`<br>`Direction =  1; //down`<br>`SetThreshold(ChannelID ,ThresholdValue ,Direction );` |
| SendText | **Syntax:**<br>`int SendText (char* text, int length);`<br>**Function:**<br>Used to send text to the mpx file with the current time stamp<br>**Results:**<br>Returns an integer, 0 = no function errors, other number indicates function error (see ***6***)<br>**Function parameters**:<br>■ **Text:** array of chars<br>■ **length: the length of the text**<br>**Examples:**<br>`char text[20] = {};`<br>`strncpy(text , "the text is in the mpx file" , 20);`<br>`int length = 20;`<br>`SendText( text , length );` |
| CheckConnectionQuality | **Syntax:**<br>`int CheckConnectionQuality(int* qualityType , real32* pQualityPercent);`<br>**Function:**<br>Used to check the quality of the connection of the system if its poor, medium or high<br>**Results:**<br>Returns an integer, 0 = no function errors, other number indicates function error (see ***6***)<br>**Function parameters**:<br>■ **qualityType:** poor, medium or high<br>■ **pQualityPercent:** the rate of the quality connection in percent |

| Function | Function syntax and example |
|---|---|
| | **Examples:**<br>`int qualityType= 0;`<br>`real32 pQualityPercent = 0;`<br>`CheckConnectionQuality( &qualityType , &pQualityPercent);` |
| GetAllChannels | **Syntax:**<br>`int GetAllChannels(SInformation *pAllChannels,int32 ChannelCount)`<br>**Function:**<br>Used to return all the channels with their name and id<br>**Results:**<br>Returns an integer, 0 = no function errors, other number indicates function error (see ***6***)<br>**Function parameters:**<br>■ **pAllChannels:** a struct that include all the channels with two fields: name and id<br>■ **ChannelCount:** number of channels the user want to see<br>**Examples:**<br>`int32 ChannelCount=220;`<br>`SInformation pAllChannels[220];`<br>`GetAllChannels(SInformation *pAllChannels,int32 ChannelCount)` |
| GetCutOffFC | **Syntax:**<br>`int GetCutOffFC(int channelID,real32 *dFCLP,real32 *dFCHP)`<br>**Function:**<br>Used to return the cut-off frequency of the low and high pass filter<br>**Results:**<br>Returns an integer, 0 = no function errors, other number indicates function error (see ***6***)<br>**Function parameters:**<br>■ **channelID:** the id of the channel<br>■ **dFCLP:** will contain the cut-off frequency of the low pass filter<br>■ **dFCHP**: will contain the cut-off frequency of the high pass filter<br>**Examples:**<br>`ChannelID = 10256;`<br>`real32 dFCLP = 0;`<br>`real32 dFCHP = 0;`<br>`GetCutOffFC( ChannelID , &dFCLP , &dFCHP );`<br>`printf("LP_freq = %d\n" , dFCLP);` |

| Function | Function syntax and example |
|---|---|
|  | `printf("HP_freq = %d\n" , dFCHP);` |
| SetChannelName | **Syntax:**<br>`int SetChannelName(int channelID , cChar* newName,int NameLength)`<br>**Function:**<br>Used to set a new name for the specified channel<br>**Results:**<br>Returns an integer, 0 = no function errors, other number indicates function error (see ***6***)<br>**Function parameters:**<br>■ **channelID:** the id of the channel<br>■ **newName:** array of chars with the new name for the channel<br>■ **NameLength:** the length of the new name<br>**Examples:**<br>`int ChannelId = 10258 ;`<br>`cChar newName[10] = {};`<br>`strncpy(newName , "left Side" , 10);`<br>`SetChannelName(ChannelId , newName , 10);` |
| GetSaveStatues | **Syntax:**<br>`int GetSaveStatues( int channelId , BOOL *pSaveState)`<br>**Function:**<br>Used to get the save status of the specified channel<br>**Results:**<br>Returns an integer, 0 = no function errors, other number indicates function error (see ***6***)<br>**Function parameters:**<br>■ **channelId:** the id of the channel<br>■ **pSaveState: 1 if the save state is true, 0 if the save state is false**<br>**Examples:**<br>`int ChannelId = 10256;`<br>`BOOL pSaveState = FALSE;`<br>`GetChannelSaveState( ChannelId , &pSaveState)`<br>`printf("status = %d\n" , pSaveState);` |
| AO_GetStopMotorTS | **Syntax:**<br>`int GetStopMotorTS (uint32* pLastStopTS)`<br>**Function:**<br>Used to get the last time stamp when the motor stopped moving<br>**Results:** |

| Function | Function syntax and example |
|---|---|
| | Returns an integer, 0 = no function errors, other number indicates function error (see ***6***)<br>**Function parameters:**<br>▪ **pLastStopTS: the last time stamp when the motor stopped**<br>**Examples:**<br>`uint32 pLastStopTS =0;`<br>`GetStopMotorTS(&pLastStopTS);`<br>`printf("StopMotorTS = %d\n" , pLastStopTS);` |

**Table 6: MATLAB Function Return Cases**

| Function return | Result |
|---|---|
| 0 | No compiling error |
| 3 | Cannot do stimulation on the specified channel |
| 4 | The system is not connected |
| 5 | The Device Driver is Null |
| 6 | The name of the channel does not existed |
| 7 | Out of range |
| 8 | The channel id does not existed |
| 9 | Null parameter |
| 10 | The system is already connected |
| 11 | Index not found |
| 12 | There is no match |
| 13 | Sampling rate is not the same |
| 14 | Gap in data |
| 15 | Wrong value for the Duration parameter |

# 3 Technical Specifications

| | **Specification** | |
|---|---|---|
| **Description** | **MATLAB** | **C++** |
| *Min. System requirements* | 1Gb Ethernet, x64, Windows 7, MATLAB 2014a | |
| *Versions* | 2010-2014 | visual studio |
| *Response time (full loop) – Wait data to Stimulate* | average: 15.2435 msec<br>standard deviation: 6.2846 msec | average: 6.7136 msec<br>standard deviation: 2.6106 msec |
| *Response time (full loop) – Wait data to Digital out* | average: 14.6384 msec<br>standard deviation: 6.5194 msec | average: 5.1992 msec<br>standard deviation: 1.8028 msec |
| *Biphasic stimulation limits* | Max pulse width – 0.5ms<br>Min pulse width – 20us<br>Frequency – 300 Hz | |
| *Max total stimulation simultaneous* | 100mA | |
| *Micro stimulation output* | <100uA | |
| *Arbitrary biphasic stimulation limits* | Max pulse width – 0.5ms<br>Min pulse width – 20us<br>Max waveform length – 1000000 samples<br>Max duty cycle – | |
| *Arbitrary analog stimulation limitations* | 300Hz – 6000 Hz | |
| *Macro stimulation output* | <7mA (macro contacts)<br><15mA (ECOG, Nerve) | |

# 4 Use Case Code

## 4.1 MATLAB Use Case #1

**Explanation about the use case:**

After connecting to the Neuro Omega system, start gathering data for the segmented channel #1. Start sending the command get channel data and every time the level line is crossed send a digital trigger. At the end of the use the connection to the Neuro Omega system is closed.

```
function [] = TestingCloseLoopStimulation()

      DspMac  = 'bc:6a:29:e1:49:bf';
      value = AO_DefaultStartConnection(DspMac);

      for j=1:100
          pause(1);
          ret = AO_IsConnected;
          if ret ==1
             fprintf('connected')
              break;
          end
      end

      segChannelId = AO_TranslateNameToID( 'SEG 01' , 6);
      BufferSizemSec = 10000;

      AO_AddBufferingChannel(segChannelId,BufferSizemSec);

       [Result,pData,DataCapture] = AO_GetChannelData(segChannelId);
      while(k<1000)
          yes = 0 ;
          unit0 = pData(4);
          if (unit0 == 0 && DataCapture >0)
              yes =1 ;
              AO_SendDout('0x05' , 3);
          end

          [Result,pData,DataCapture] = AO_GetChannelData(segChannelId);
          AO_SendDout('0x05' , 0);
          k=k+1;
      end


      AO_CloseConnection();
      While(1){
            ret = AO_IsConnected;
            if ret == 0
             fprintf('disconnected')
             break;
            end
      end
```

```
end
```

## 4.2 MATLAB Use Case #2

**Explanation about this use case:**

After connecting to the Neuro Omega system, start collecting for the three SPK channels 10256, 10257 and 10258 (Micro SPK channel 1, Micro SPK channel 2 and Micro SPK channel 3). Then start saving the data on mpx file. Call the get aligned command and get the data from the three channels arranged one after another. At the end of the use case we making sure the connection to the Neuro Omega system is closed.

```
function [res,Result,pData,DataCapture,TS_FirstSample] = Testing_GetAlignedData()
      'Testing Default start connection command ';
      DspMac   = 'c8:a0:30:27:21:bf';
      value = AO_DefaultStartConnection(DspMac);

      for j=1:100
          pause(1);
          ret = AO_IsConnected;
          if ret ==1
             fprintf('connected')
              break;
          end
      end

      'gather data for the three SPK channels 10256, 10257, 10258';
      res(1)  = AO_AddBufferingChannel(10256,10000);
      res(2)  = AO_AddBufferingChannel(10257,10000);
      res(3)  = AO_AddBufferingChannel(10258,10000);

      arr_SPK=[10256,10257,10258];
      sizeArr_SPK = length(arr_SPK);

      ret = AO_StartSave(); %start save the data
          if ret >0
              'missing Saving File';

          end
      pause(10);

      [Result,pData,DataCapture,TS_FirstSample] =
      AO_GetAlignedData(arr_SPK,sizeArr_SPK);

      pause(20);

      AO_StopSave()%stop save the data

      AO_CloseConnection();
      While(1){
            ret = AO_IsConnected;
            if ret == 0
             fprintf('disconnected')
             break;
            end
      end
end
```

## 4.3 MATLAB Use Case #3

**Explanation about this use case:**

After connecting to the Neuro Omega system, start generating stimulation trains which are repeated in a burst frequency

> note: stimulation parameters need to be set according to the
> AO_StartDigitalStimulation command.

```
'Testing Default start connection command ';
DspMac  = 'bc:6a:29:e1:49:bf';
value = AO_DefaultStartConnection(DspMac);

for j=1:100
    pause(1);
    ret = AO_IsConnected;
    if ret ==1
       fprintf('connected')
        break;
    end
end

train_freq=100; % the frequency between the pulses (for example 100Hz)
number_of_pulses=3; % number of pulses in every train (for example 3
pulses)
burst_freq=10; % burst frequency between the trains (for example 10Hz)
number_of_bursts=100; % total number of bursts in a stimulation season
stim_channel_name= 'SPK 01'; % stimulation channel name as appears in
software
nameLength=length(stim_channel_name);
stim_channel=AO_TranslateNameToID( stim_channel_name , nameLength );


for i=1:number_of_bursts
    [Result] = AO_StartDigitalStimulation(stim_channel, 0, -0.09, 0.06, 0,
0.09, 0.06, train_freq, (number_of_pulses/train_freq)-
0.1*(number_of_pulses/train_freq), -1);
    pause((1/burst_freq));
end
[Result] = AO_StopStimulation(-1);
```

# 5 Troubleshooting Guideline

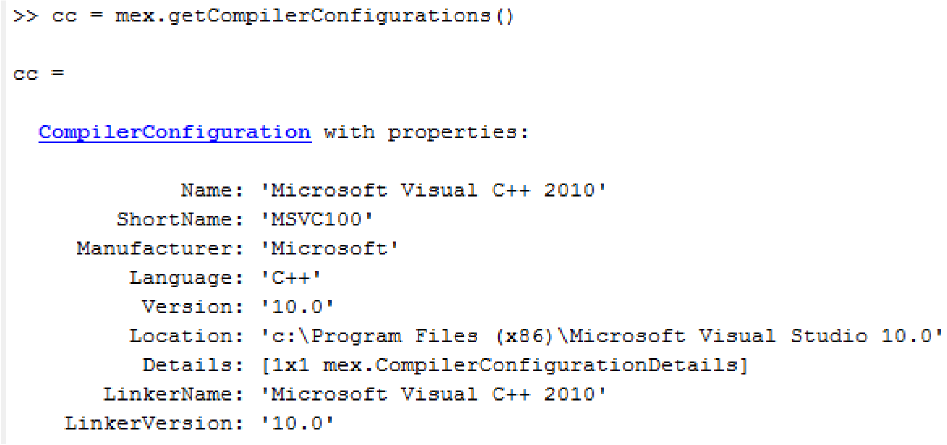
## 5.1 MEX Compiler Error

This error can occur when there the .NET framework is not the correct version or the Windows SDK is not installed correctly or at all.

To start the troubleshooting, enter the following command into the MATLAB command window:

- cc = mex.getCompilerConfigurations()

a. In case the following message appears, check if the Windows SDK and the .NET framework 4.0 are installed



```
>> cc = mex.getCompilerConfigurations()

cc =

  CompilerConfiguration with properties:

             Name: 'Microsoft Visual C++ 2010'
        ShortName: 'MSVC100'
     Manufacturer: 'Microsoft'
         Language: 'C++'
          Version: '10.0'
         Location: 'c:\Program Files (x86)\Microsoft Visual Studio 10.0'
          Details: [1x1 mex.CompilerConfigurationDetails]
       LinkerName: 'Microsoft Visual C++ 2010'
    LinkerVersion: '10.0'
```

**Figure 7: MATLAB Compiler Configuration**

i. Remove the Microsoft Windows SDK and the .Net Framework through the control panel uninstall programs.

ii. Download the .NET Framework V4.0 and above from Microsoft and install

iii. Download the Microsoft Windows SDK for Windows 7 from Microsoft and install

**Notes:**

- Make sure that the .NET Framework 4.0 or above is installed before the Microsoft Windows SDK for Windows 7
- Windows 7 ships with only .NET Framework 3.5. The MEX compilation requires .NET Framework 4.0 and above.

b. In case the message in *Figure 7: MATLAB Compiler Configuration* does not appear, there is no compiler installed on the system, do the following:

iv. MATLAB recommends installing Microsoft Visual C++ Express 2010 and the Microsoft Windows SDK for Windows 7 from Microsoft

**Warning:**

- One common error that is seen is Return Code 5100. This indicates that there were existing installs of redistributable Microsoft Visual C++ and that installation could not proceed. In this case, you need to uninstall the existing Visual C++ redistributable installations.

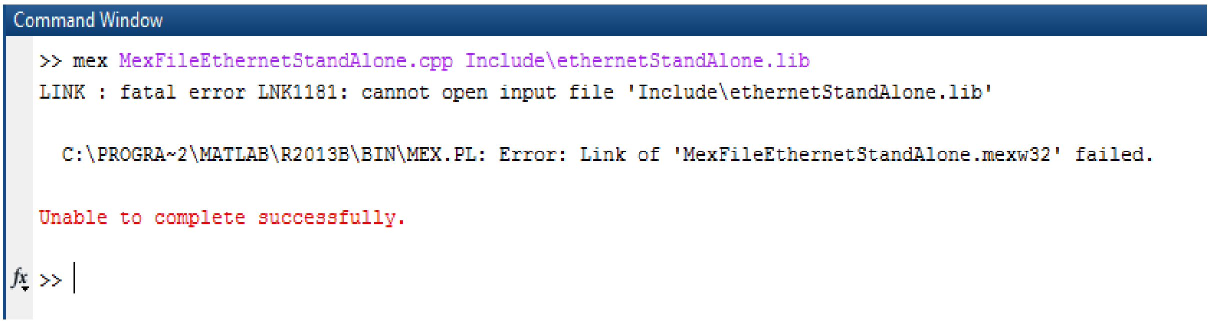
## 5.2 Missing Runtime Libraries

MATLAB will fail to load MEX-files if it cannot find all DLLs referenced by the MEX-files; the DLLs must be in the same directory as the MEX-file.

**Notes:**

- On 64-bit Windows, the MEX files require the Visual Studio runtime libraries.
- If an error occurs follow the link that suggested in the line
- The information took from: http://warpproject.org/trac/wiki/howto/MEX_Compile

In case the EthernetStandAlone.lib is missing from the include file the following message will appear:



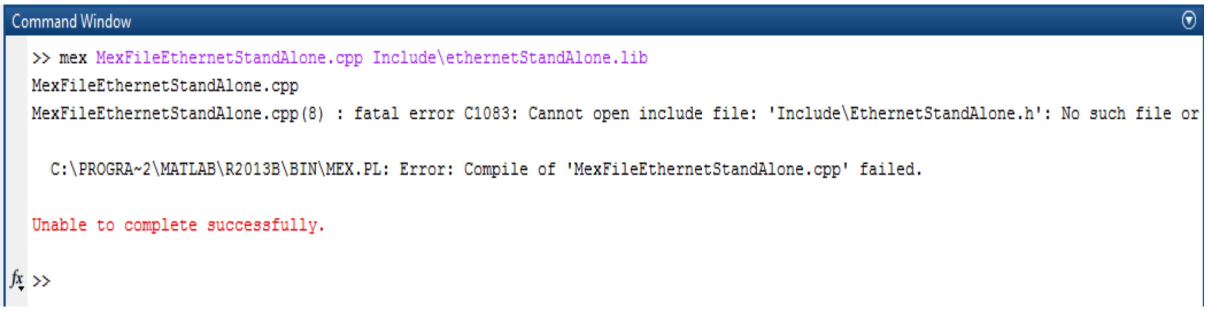
```
>> mex MexFileEthernetStandAlone.cpp Include\ethernetStandAlone.lib
LINK : fatal error LNK1181: cannot open input file 'Include\ethernetStandAlone.lib'

  C:\PROGRA~2\MATLAB\R2013B\BIN\MEX.PL: Error: Link of 'MexFileEthernetStandAlone.mexw32' failed.

Unable to complete successfully.

fx >> |
```

In case EthernetStandAlone.h is missing from the include file the following message will appear:



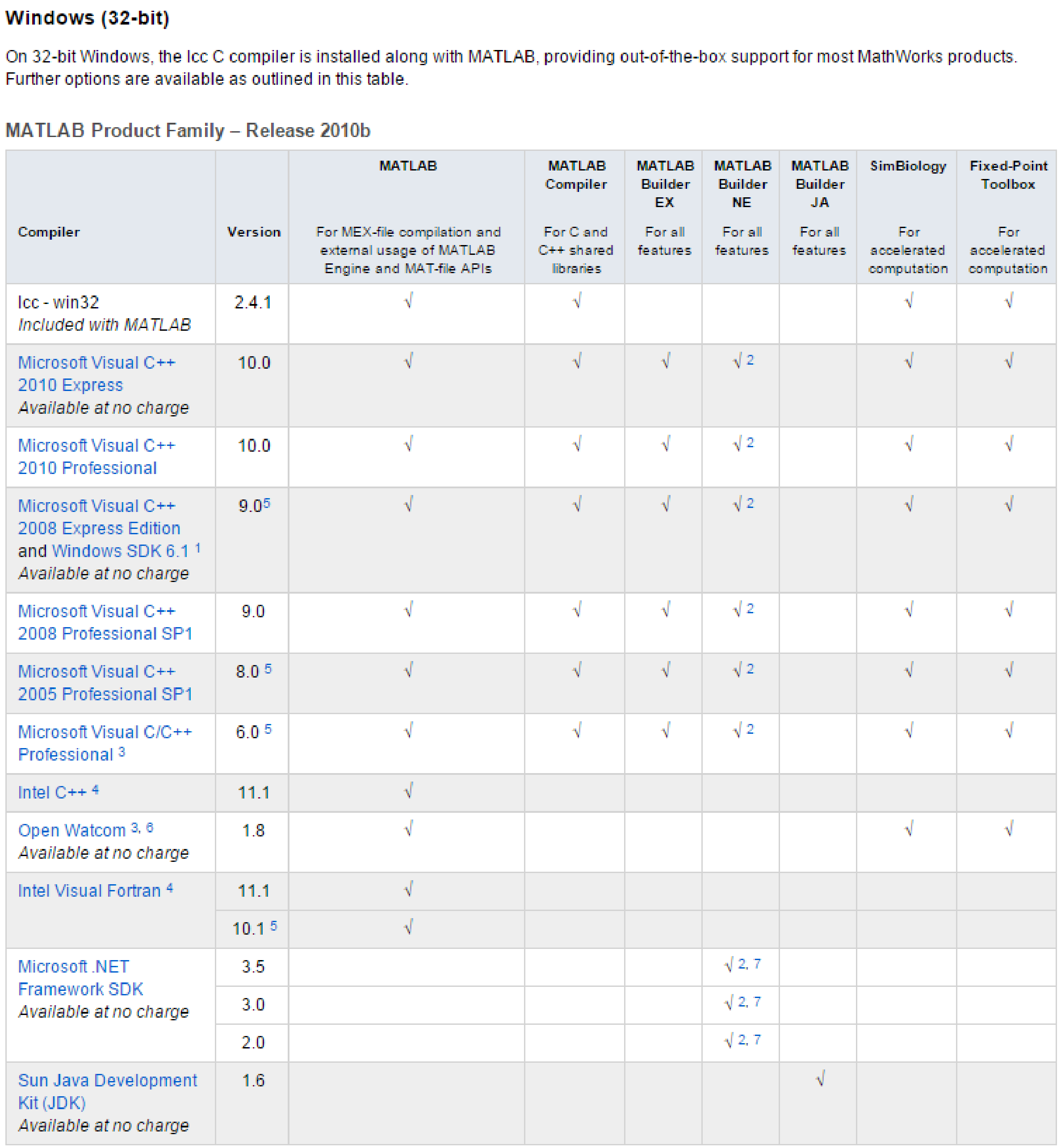
```
>> mex MexFileEthernetStandAlone.cpp Include\ethernetStandAlone.lib
MexFileEthernetStandAlone.cpp
MexFileEthernetStandAlone.cpp(8) : fatal error C1083: Cannot open include file: 'Include\EthernetStandAlone.h': No such file or

  C:\PROGRA~2\MATLAB\R2013B\BIN\MEX.PL: Error: Compile of 'MexFileEthernetStandAlone.cpp' failed.

Unable to complete successfully.

fx >>
```

## 5.3 Supported and Compatible Compilers – Release 2010B



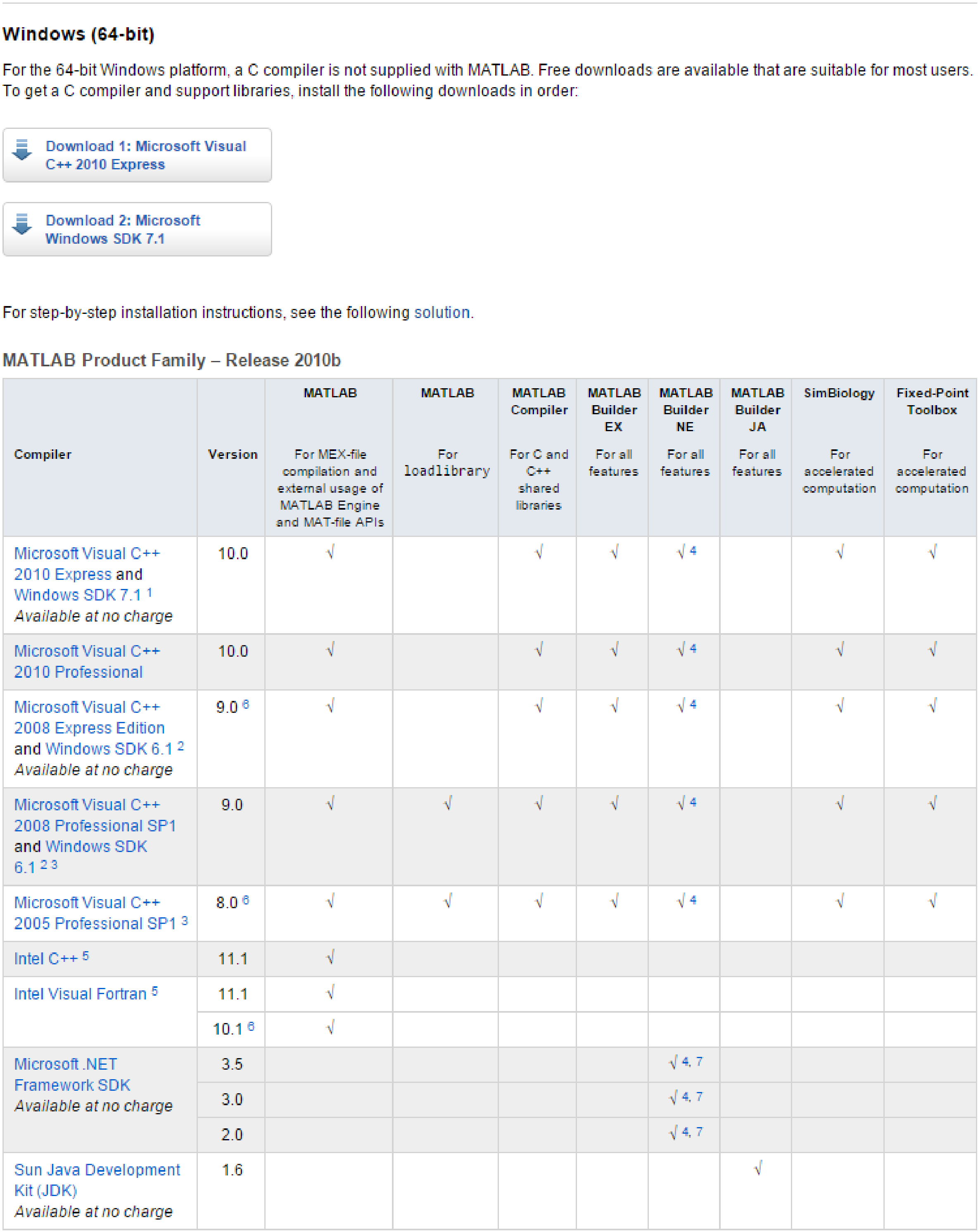
### Windows (32-bit)

On 32-bit Windows, the lcc C compiler is installed along with MATLAB, providing out-of-the-box support for most MathWorks products. Further options are available as outlined in this table.

MATLAB Product Family – Release 2010b

| Compiler | Version | MATLAB<br>For MEX-file compilation and external usage of MATLAB Engine and MAT-file APIs | MATLAB Compiler<br>For C and C++ shared libraries | MATLAB Builder EX<br>For all features | MATLAB Builder NE<br>For all features | MATLAB Builder JA<br>For all features | SimBiology<br>For accelerated computation | Fixed-Point Toolbox<br>For accelerated computation |
|---|---|---|---|---|---|---|---|---|
| lcc - win32<br>*Included with MATLAB* | 2.4.1 | √ | √ | | | | √ | √ |
| Microsoft Visual C++ 2010 Express<br>*Available at no charge* | 10.0 | √ | √ | √ | √[2] | | √ | √ |
| Microsoft Visual C++ 2010 Professional | 10.0 | √ | √ | √ | √[2] | | √ | √ |
| Microsoft Visual C++ 2008 Express Edition and Windows SDK 6.1 [1]<br>*Available at no charge* | 9.0[5] | √ | √ | √ | √[2] | | √ | √ |
| Microsoft Visual C++ 2008 Professional SP1 | 9.0 | √ | √ | √ | √[2] | | √ | √ |
| Microsoft Visual C++ 2005 Professional SP1 | 8.0 [5] | √ | √ | √ | √[2] | | √ | √ |
| Microsoft Visual C/C++ Professional [3] | 6.0 [5] | √ | √ | √ | √[2] | | √ | √ |
| Intel C++ [4] | 11.1 | √ | | | | | | |
| Open Watcom [3, 6]<br>*Available at no charge* | 1.8 | √ | | | | | √ | √ |
| Intel Visual Fortran [4] | 11.1 | √ | | | | | | |
| | 10.1 [5] | √ | | | | | | |
| Microsoft .NET Framework SDK<br>*Available at no charge* | 3.5 | | | | √[2, 7] | | | |
| | 3.0 | | | | √[2, 7] | | | |
| | 2.0 | | | | √[2, 7] | | | |
| Sun Java Development Kit (JDK)<br>*Available at no charge* | 1.6 | | | | | √ | | |

**Figure 8: Supported Compilers for Windows 32bit (taken from the MathWorks website)**



### Windows (64-bit)

For the 64-bit Windows platform, a C compiler is not supplied with MATLAB. Free downloads are available that are suitable for most users. To get a C compiler and support libraries, install the following downloads in order:

Download 1: Microsoft Visual C++ 2010 Express

Download 2: Microsoft Windows SDK 7.1

For step-by-step installation instructions, see the following solution.

MATLAB Product Family – Release 2010b

| Compiler | Version | MATLAB<br>For MEX-file compilation and external usage of MATLAB Engine and MAT-file APIs | MATLAB<br>For `loadlibrary` | MATLAB Compiler<br>For C and C++ shared libraries | MATLAB Builder EX<br>For all features | MATLAB Builder NE<br>For all features | MATLAB Builder JA<br>For all features | SimBiology<br>For accelerated computation | Fixed-Point Toolbox<br>For accelerated computation |
|---|---|---|---|---|---|---|---|---|---|
| Microsoft Visual C++ 2010 Express and Windows SDK 7.1 [1]<br>*Available at no charge* | 10.0 | √ | | √ | √ | √ [4] | | √ | √ |
| Microsoft Visual C++ 2010 Professional | 10.0 | √ | | √ | √ | √ [4] | | √ | √ |
| Microsoft Visual C++ 2008 Express Edition and Windows SDK 6.1 [2]<br>*Available at no charge* | 9.0 [6] | √ | | √ | √ | √ [4] | | √ | √ |
| Microsoft Visual C++ 2008 Professional SP1 and Windows SDK 6.1 [2] [3] | 9.0 | √ | √ | √ | √ | √ [4] | | √ | √ |
| Microsoft Visual C++ 2005 Professional SP1 [3] | 8.0 [6] | √ | √ | √ | √ | √ [4] | | √ | √ |
| Intel C++ [5] | 11.1 | √ | | | | | | | |
| Intel Visual Fortran [5] | 11.1 | √ | | | | | | | |
| | 10.1 [6] | √ | | | | | | | |
| Microsoft .NET Framework SDK<br>*Available at no charge* | 3.5 | | | | | √ [4, 7] | | | |
| | 3.0 | | | | | √ [4, 7] | | | |
| | 2.0 | | | | | √ [4, 7] | | | |
| Sun Java Development Kit (JDK)<br>*Available at no charge* | 1.6 | | | | | | √ | | |

**Figure 9: Supported Compilers for Windows 64bit (taken from the MathWorks website)**