



CORTECH
S O L U T I O N S

User Manual rev 1.2

ActiveTwo API for MATLAB®

Model: DA-AT-SWMAPI

Copyright © 2011, Cortech Solutions, Inc.

Trademarks

MATLAB is a registered trademark of The MathWorks, Inc. For MATLAB product information, please contact:

The MathWorks, Inc.
3 Apple Hill Drive, Natick, MA, 01760-2098 USA
Tel: 508-647-7000
Fax: 508-647-7101
E-mail: info@mathworks.com
Web: www.mathworks.com

Windows is a registered trademark of Microsoft Corporation. For Windows product information, please contact:

Microsoft Corporation
One Microsoft Way
Redmond, WA 98052-6399
Web: www.microsoft.com

Table of Contents

ActiveTwo API for MATLAB®	1
Model: DA-AT-SWMAPI.....	1
Copyright © 2011, Cortech Solutions, Inc.	1
1. Hardware and Software Requirements	5
1.1 Hardware Requirements	5
1.2 Software Requirements.....	5
1.3 Verify the ActiveTwo Windows® USB driver:	5
2. Installation from a CD or downloaded files	6
2.1 ActiveTwo API Setup Instructions.....	6
3. INTRODUCTION.....	8
3.1 ActiveTwo API for MATLAB® (ActiveTwo API)	8
3.2 Release Notes:	8
4. ACTIVETWO USB FRAME RATE & API PROCESSING DELAYS	9
4.1 ActiveTwo USB Frame Rate: the API time 'Period'.....	9
4.2 ActiveTwo API Frame Rate: the API time 'ScanRate'	9
USB buffer size.....	9
Data access delays.....	10
Developer's Code processing delays	10
5. MATLAB® CLASSES	12
6. QUICK START	13
7. ACTIVETWO API INITIALIZATION.....	13
7.1 Object construction	13
7.2 Hardware and Software initialization.....	14
8. ACTIVETWO OBJECT PROPERTIES.....	16
8.2 ActiveTwo Hardware/Flow Properties	17
8.3 Call Back functions.....	17
8.4 Timing properties	18
8.5 Message Notification Properties	18
8.6 Quality Control Properties.....	18
8.7 Simulator Properties.....	19
8.9 Acquisition Data Properties.....	19
9. ActiveTwo API METHODS	20
9.1 ActiveTwo API Initialization	20

9.2 ActiveTwo Class Object Deletion	20
9.3 Active2 Class Configuration	21
9.4 Active2 Class Driver	21
9.5 ActiveTwo Class SIMULATOR.....	22
9.6 Active2 Class Data Acquisition.....	23
9.7 ActiveTwo Class Inspection	24
10. ACTIVETWO API EVENTS & CALLBACK FUNCTIONS	26
10.1 ActiveTwo Class Events.....	26
11. ACTIVETWO API DEMO SAMPLES	28
12. ACTIVETWO API DAISY CHAIN MODE	28
13. ACTIVETWO API EMBEDDED FEATURES	30
13.1 Quality Control Tool	30
13.2 ActiveTwo Simulator Tool.....	33

1. Hardware and Software Requirements

1.1 Hardware Requirements

- 1.1.1. Intel or AMD based PC, minimum 1GHz CPU clock, minimum 2 GB RAM, 20GB hard drive.
- 1.1.2. One free USB port on your PC.

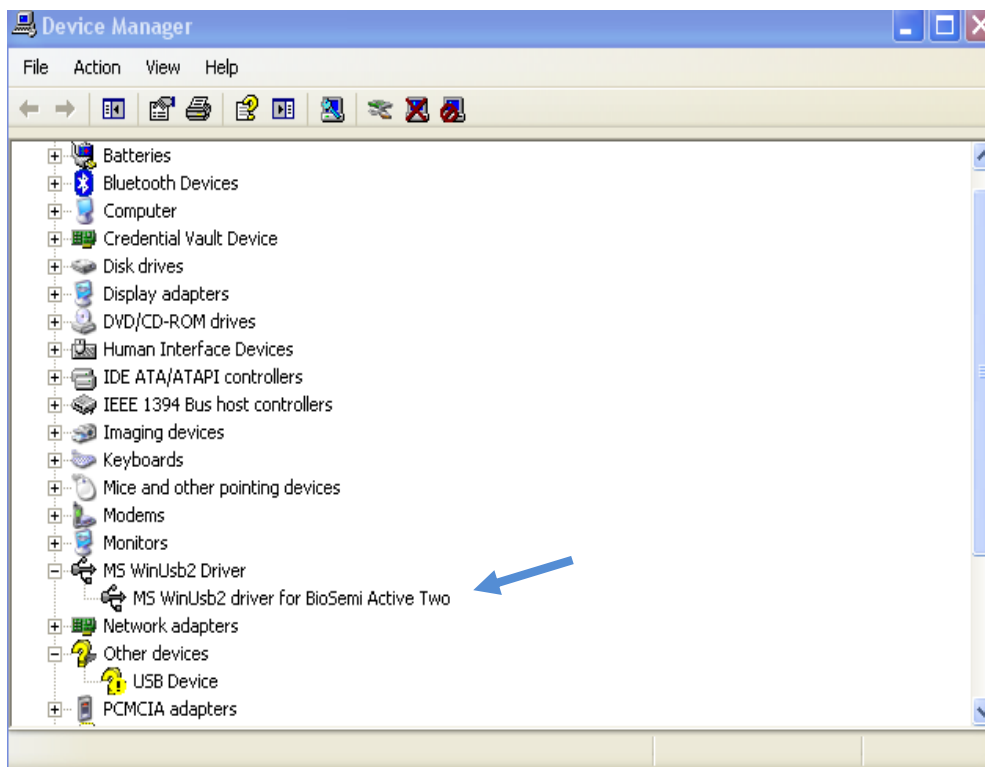
1.2 Software Requirements

- 1.2.1. Windows® (XP preferably) 32-Bit service pack 3 (SP3)
- 1.2.2. MATLAB® 2008b or earlier.

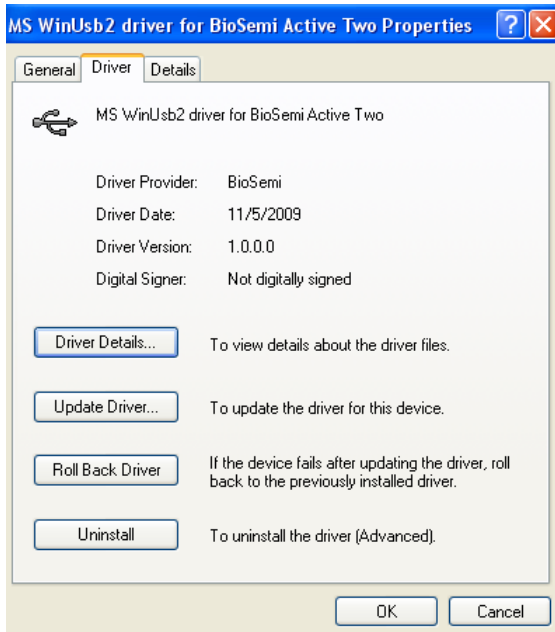
BioSemi ActiveTwo USB driver: MS WinUbs2 Driver, BioSemiWinUsb2.INF. Version 6.1.7600.16385

1.3 Verify the ActiveTwo Windows® USB driver:

- 1.3.1. Go to the Device Manager, find the MS WinUbs2 Driver, see the window below.



- 1.3.2. Right click on the MS WinUbs2 Driver and select 'Properties'. Your driver should be the same as the window below.



2. Installation from a CD or downloaded files

2.1 ActiveTwo API Setup Instructions

1. Contents of your Install CD or downloaded zip file:

Root path Files:

Licenses_CSI.dat	License file listing the products available in your license
Active2.p	ActiveTwo Matlab® API Class
Active2QuickSTART.m	This introductory script will help you to quickly operate the API
setup.p	Install program needed to begin using your API
Uninstall.p	Uninstalls all previously installed software
<i>Bin</i>	Binaries files needed to install the API
<i>AuxFiles</i>	Auxiliary files like libraries and external functions
<i>Configuration</i>	ActiveTwo Configuration files, the same ones as in ActiveView
<i>BDFfiles</i>	Sample EEG recordings to test your ActiveTwo Simulator
<i>SampleCode</i>	Plenty of sample code to help you get started

Table1 notes: text in ***Bold-Italics*** indicate directories

2. Create a directory 'ActiveTwo' preferably on either your Matlab® work directory or other folder residing on your local root directory.
3. Please copy the contents of the software CD to your newly created destination folder. If we provided a direct link for download, please download zipped file. Unzip file to the newly created destination folder.
4. Verify that you have administrator rights to ensure that you have the ability to install software. Run the program 'setup.p' program. Make sure that your personal firewall allows the 'Aladdin HASP License Manager' to access the network.

5. Set the Matlab® path with 'set path' to add the directory location of your ActiveTwo API.
 - a. Set the 'current directory' to the directory location of your ActiveTwo API.
 - b. Go to the File Menu's 'Set Path' option.
 - c. Click the 'Add with subfolders' button.
 - d. Click Save and Close.
 - e.
6. Generate a fingerprint , **fpr**, file:
 - a. Please type at the command line: Active2.Write_fpr;
Example: create a personal fingerprint fpr file with the 'Write_fpr' method.
>> Active2.Write_fpr;
Created the 'fpr' , Local_user_ID.fpr ,file in :
C:\Matlab\ActiveTwo
Please send the 'fpr' file to Alex Fernandez-Villa at Cortech Solutions Inc.
sales@cortechsolutions.com
cortechsolutions.com
 - b. The above action will generate a file named, '*Local_user_ID.fpr*'. Please send this file to us so that we can generate a provisional license Activation file for you, c2c file.
7. We will immediately generate the c2c file, which you will copy to your ActiveTwo directory. At this point you can use the provisional 30 day license while your HL key is on the way.
8. Insert your USB Hard-lock key for your ActiveTwo Matlab® API license (for licensed users only).
 - a. Windows will match the drivers installed in step 3 to your USB Hard-lock key.
 - b. Note your specific License number is printed on the USB Hard-lock Tag.

3. INTRODUCTION

3.1 ActiveTwo API for MATLAB® (ActiveTwo API)

This document describes in detail the calling conventions and unique features integrated in this API.

The ActiveTwo API is a medium level programming interface to enable a rapid and comprehensive interaction between the developer and the complex BioSemi ActiveTwo hardware. It facilitates the developer's experience by handling the driver communications and transforming the complex ActiveTwo Serial data stream into comprehensive MATLAB® class structure.

Traditionally software APIs require that the developers invest many hours of their precious time to understand setting rules, write, and debug your API setup code and workflow. Our chief goal of when contriving this product was to reduce the developer's function calls, setup time, and at the same time present the user with all the ActiveTwo information in a concise fashion. Setting up the ActiveTwo and integrating your code with the API takes seconds. There is no special interfacing code or toolbox required to transfer Bio Signal data from ActiveTwo hardware to MATLAB® code.

3.2 Release Notes:

- The Jazz eye tracker is not supported.
- The BDF file writer feature is not operational on this release.
- Speed Mode 9 is not fully supported in this release.

4. ACTIVETWO USB FRAME RATE & API PROCESSING DELAYS

4.1 ActiveTwo USB Frame Rate: the API time 'Period'.

The ActiveTwo class uses the BioSemi ActiveTwo USB2 driver version, which allows for a minimum user defined USB frame rate of 1 millisecond; we call this property the 'Period'. This MATLAB® API allows the user to set the 'Period' to ≥ 0.001 seconds; the default setting is 0.060 seconds.

For instance if you set the hardware speed mode to 4, the sampling rate is 2048 Hz. When setting the 'Period' to 0.005 seconds, the USB driver will add new data to the Buffer every 0.005 seconds. At 2048Hz, the newly arrived USB packet/frame will contain $(2048 \text{ points/second}) * 0.005 \text{ seconds} = 11$ points for every channel, see table 2.

Point	1	2	3	4	5	6	7	8	9	10	11
Timing (ms)	-5 (oldest)	-4.5	-4.0	-3.5	-3.0	-2.5	-2.0	-1.5	-1.0	-0.5	0 (Newest)

Table2: frame size illustration per channel for 0.005 second period at mode4 (2048Hz).

The frame size is 11 points long and the oldest point will have a delay of 0.005 seconds.

4.2 ActiveTwo API Frame Rate: the API time 'ScanRate'.

The 'ScanRate' is the rate at which the API fetches new data from the USB buffer; it is set to 0.200 seconds by default. But to set the API 'ScanRate' we first must understand the factors driving API processing delays. These delays solemnly depend on (1) the user defined 'BufferSize', (2) data access, and (3) the developer's processing time.

USB buffer size

The 'BufferSize' is in place to prevent data losses in the event that Windows® removes resources from the USB driver process. If this happens MATLAB® may not be able to retrieve data from USB buffer in a timely fashion and data losses may occur. To prevent this, the USB driver saves the incoming data into special memory space; typically called the USB buffer. You can use this relation, $282 * \text{sampling Rate} * \text{Delay} * 4$, to compute the USB buffer size needed. If the API is halted for 1 second, you will have to buffer data for at least 1 second or (for mode 4) $282 * 2048 * 1 * 4 = 2$ Mbytes. The 'BufferSize' default setting is 16 Mbytes or 7 seconds of data.

When deciding on a 'BufferSize' it is desirable to keep it as small as possible since the API 'BufferSize' delay is proportional to the USB buffer allocated memory. See the table 3 for delays.

Buffer Size (Mbyte)	Average Overhead Delay (ms)
2.00	2.13
4.00	4.78

8.00	7.94
16.00	14.42
32.00	32.01

Table : At a 40 ms Period, 80ms Scan-Rate

Data access delays

When accessing data from the various channel groups the API has to fetch data from the appropriate position of the USB buffer. Therefore, there will be a delay associated with the particular bulk size of the data access. The following graph (Data access delay vs. Time) depicts the increase in data access delay when gradually increasing the data access demand in the order shown.

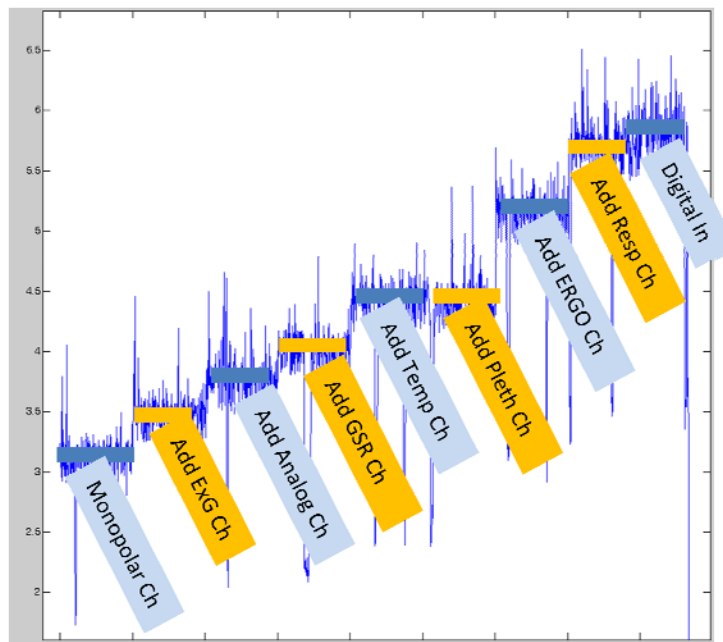


Figure 1 : Data Access delay (ms) vs. Time.

When operating in a Daisy Chain mode you need to add 2ms of delay to the values shown in figure 1.

Developer's Code processing delays

This delay depends on the amount processing time investment incurred by the developer's code. This is entirely up the user.

Since we have discussed the three sources of delay, (1) BufferSize, (2) Data access, (3) and developer's code delay, we can determine how much time we should provision for the 'ScanRate'. For instance, if our three delays amount to 0.100 seconds, we should select a 'ScanRate' of at least 0.100 seconds + small variations (

maybe 20 ms). See figure 2. Please carefully plan your timing and follow these rules, or your buffer will eventually overflow, causing the Active API to terminate the Acquisition due to loss of data.

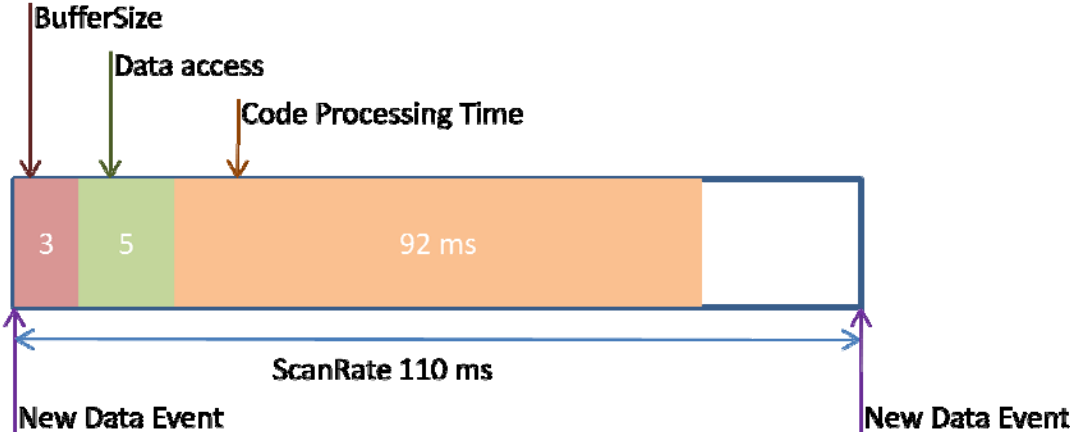


Figure 2

5. MATLAB® CLASSES

The ActiveTwo API is for the most part embodied by a MATLAB® class structure which we call the 'ActiveTwo' class. Classes typically yield their specific instantiations which we refer to as objects, so the **ActiveTwo** class yields an **ActiveTwo** type object. In this section we will briefly discuss MATLAB® Classes.

Most MATLAB® variables are indeed objects representing a class. For instance, a variable of type **int8** represents an object from the MATLAB® **int8** class. And as such, its use is bounded by the rules defined by the MATLAB® class type **int8**. For instance, an **int8** type object can only represent an integer number from -128 to 127.

Example: study a MATLAB® class object.

```
>>n = int8(40);
>> whos n;

Name      Size      Bytes      Class  Attributes
n         1x1         1          int8
```

Classes are data structures which contain fields or variables called *properties*. Classes also contain functions, called *methods*, which act on external variables (we know them as input variables) or on the class' own variables/properties. Let's inspect the properties and methods of the object **n**:

Example: list the available features of the int8 Class.

```
>> properties(n)

No properties for class int8 or no class int8.

>> methods(n)

Methods for class int8:
abs  eq  le  nnz  sign
... sum  ... minus  plus  ... tril
... rem  xor  ...
display  ldivide  ne  round
```

We see that the class type **int8** does not contain any *properties* but it does allow for many *methods* to act on the objects of type **int8**. For example, the common *methods* we might employ on a type **int8** are 'plus', 'minus', or 'abs'. In MATLAB® we make constant use of the 'colon' *method*:

```
>> n = int8(1):int8(5)
n =
    1    2    3    4    5
>> whos n

Name      Size      Bytes      Class  Attributes
n         1x5         5          int8
```

And the *method* 'display' is always called when we omit the ';', for example:

```
>> n
n = 1 2 3 4 5
```

6. QUICK START

Please refer to the file 'ActiveTwoQuickStart.m' script file and follow along the code and detailed comments.

7. ACTIVETWO API INITIALIZATION

The API setup can be realized in one line of code. First you must ensure that your ActiveTwo hardware is turned on and the USB cable connected to your computer. To setup the ActiveTwo you must call the function 'Active2', this is otherwise called object construction.

7.1 Object construction

The function ActiveTwo has one output variable, which is known as the object, and 14 input properties at construction. You may call the 'Active2' construction function without specifying any properties and configure them later on.

Example

```
>> a2 = Active2;
```

```
Verifying License keys.
```

```
Found 1 C.S.I License Keys.
```

```
Found a HASP-Software-key, License number: 555448472208.
```

```
...searching for valid products...
```

```
Found 1 valid C.S.I licensed product(s):
```

```
  'ActiveTwo API for MATLAB® Release 1'
```

```
You must enable/turn On the ActiveTwo to set the Digital Outputs.
```

```
Warning: Initialize the ActiveTwo at least once to determine maximum number of Monopolar Channels
```

```
Warning: Assigning the minimum amount of Monopolar Channels: Channels 1 through 8
```

```
>> whos a2;
```

Name	Size	Bytes	Class	Attributes
a2	1x1	60	Active2	

The ActiveTwo constructor checks your license and shows you the available licenses and licensed products. Upon **Active2** type object construction you may obtain some warnings to remind you that you have initialize the ActiveTwo hardware/software at least once. Other warnings let you know that you have not selected all the channels available to you.

Alternatively you may specify some properties at construction without following any particular order.

Example

```
>> a = Active2('Property1', value1 , 'Property2' , value2 , ... , 'PropertyN' , valueN);
```

Property Name	Property Description	Data Type	Default Value
BufferSize	the size of the USB Buffer in Mbytes	Numeric	4
PollPeriod	USB driver period	Double, seconds	0.040 seconds
FrameRate	API frame rate	Double, seconds	0.080 seconds
ConfigFile	BioSemi ActiveView Configuration File.	Path string	'\Default.cfg'
UserData	User defined data structure	-	[]
ExecutionMode	API Timer ExecutionMode	'fixedRate', 'fixedSpacing', or 'fixedDelay'	'fixedRate'
BusyMode	API Timer Busy Mode	'queue' or 'drop'	'queue'
Init	Initialize hardware (1)	Numeric, 0 or 1	0
NewDataFnc	Specify a New Data event callback function	function handle or string<function name>	@myNewDataCallback
SynchErrorsFnc	Specify a Synchronization Error event callback function	function handle or string<function name>	@mySyncErrCallback
BatteryLowFnc	Specify a Battery Low event callback function	function handle or string<function name>	@myBattLowCallback
CMSDRLFnc	Specify a CMS/DRL out of Range event callback function	function handle or string<function name>	@myCMSCallback
GSRFnc	Specify a GSR out of Range event callback function	function handle or string<function name>	@myGSRNOIRCallback
Sim	Initialize and load a BDF file for Simulation	Path string	"

Table 4 : ActiveTwo API construction input properties.

7.2 Hardware and Software initialization

The initialization is a critical step which must be performed at least once each time the ActiveTwo object is created. During initialization the API queries the ActiveTwo hardware and the configuration '.cfg' file for vital information such as 'Speed Mode number', 'ActiveTwo hardware version', and other software settings such as channel Labels. The hardware and software settings are transferred to the ActiveTwo object properties 'status' and 'Config'

To initialize the ActiveTwo object call the method 'init'.

Example

```
>> a2.init;
```

Driver opened.

Start up Acquisition.

Ringbuffer Initialized.

USB Handshake Enable.

```
<<< ACTIVETWO IS ON >>>
```

MK2: 1, GSR16Hz: 1, Jazz: 1, Speed Mode: 4, BatteryLow: 0
V20100224 using: MS winusb library, RAWIO, 41.22 msec stride, not sync'ing pointer calls.
Disabled USB-Handshake. DAQ Stopped!
ActiveTwo Driver closed.
<<< ACTIVE2 IS OFF >>>
<<< ACTIVE2 IS NOW INITIALIZED >>>

After initializing you may inspect the 'Status' and 'Config' Properties.

Example: Status Data Structure.

>> a2.Status

```
ans =  
    Mk2: 1  
    GSR16Hz: 1  
    JazzRec: 1  
    SpeedMode: 4  
    BatteryLow: 0  
    CMSinRangeArray: 1  
    CMSinRange: 1
```

The 'Status' property informs us about the hardware changes on the ActiveTwo.

Example: Config Data sctructure.

>> a2.Config

ans =

```
Initialize: 0  
Warning: ""  
MotherBoard: 12  
ElecGain: 0  
RespSwitch: 0  
ChanLabels: [600x20 char]  
TouLabels: [50x20 char]  
AuxLabels: [50x20 char]  
JazzLabels: [50x20 char]  
BoxLabels: [50x20 char]  
TrigLabel: 'Status'  
FileLimit: 999999  
RecTimeLimit: 9999999  
Decimation: 0  
Buffer: 2  
Channels: 4  
AddTP: 1  
Reference: 0  
JazzOn: 0  
SaveSubset: 0
```

```

SaveTouchProofs: 1
SaveSensors: 1
SaveJazz: 1
SaveAnas: 1
SaveBox: 15
SavePath: 'C:\BDFdata\Testdata.bdf'
MonChannels: 256 // comment: Monopolar channels (EEG).
ADCCchannels: 282 // comment: total number of channels.
JazzControl: 0
DecimationFactor: 1
JazzChannels: 0
SamplingRate: 2048 // comment: the Sampling Rate in Hz units.
AuxChannels: 8 // comment: sensor channels.
AnaChannels: 0 // comment: Analog channels.
BipChannels: 8 // comment: ExG channels.

```

The 'Config' data structure holds information transferred from the .cfg configuration file. In addition it contains important hardware settings such as: Sampling Rate and channel distribution; see the above comments in green font.

8. ACTIVETWO OBJECT PROPERTIES

The API is implemented into a MATLAB® class which encapsulates vital ActiveTwo data into object properties/variables. To access these properties you must use the dot ('.') operator. In addition, note that all the properties are case sensitive.

Example: access the ActiveTwo's battery percentage charge available.

```
>> a2.BatteryLevel
ans =
```

```
45.0207
```

The **Active2** class properties are arranged the following fashion.

8.1 ActiveTwo API General Properties

Property Name	Property Description	Data Type	Default Value
Active2Dll	ActiveTwo driver installed	string	Labview_DLL Ver6.1.7600.16385 (WinUSB2driver.inf)
ver	Product version	string	ActiveTwo API for MATLAB® version x
hasp_status	License key status message	string	'HASP_STATUS_OK'
hasp_license_key	License key number	string	[]
LicenseStatus	License status errors	double	0

UserData	User defined data space	User defined	[]
Initialize	Initialize ActiveTwo API	Boolean	0
ConfigFile	BioSemi configuration file	File	'default.cfg'
VoltageGain	Channel scaling	double	8192
ADCbox	Daisy chain mode box selection. Options [0 1 2 3]	double	0

Table 5 : ActiveTwo API construction input properties.

8.2 ActiveTwo Hardware/Flow Properties

Property Name	Property Description	Data Type	Default Value
Status	Hardware status data structure	structure	[]
Config	Software & Hardware configuration data structure	structure	[]
BatteryLow	Battery low indicator	Boolean	0
Battery Level	Percentage of battery charge left.	Double	0
Engaged	Is the driver actively engaged	Boolean	0
DataBufferLag		Int32	0
BufferOverflow	Buffer overflow indicator	Boolean	0
FixedFrameRate	Choose a variable of fixed data buffer. The net effect is that your 'ScanRate' becomes variable	Boolean	1
SamplingRate	ADC channel sampling rate in Hz	Double	0
ADCChannels	Number of all possible ADC channels	double	0
Running	Acquisition	string	'off'
DigitalOut	Set and inspect the trigger port's digital outputs	Boolean array	Zeros[1x15]

Table 6 : ActiveTwo API hardware and flow properties.

8.3 Call Back functions

Property Name	Property Description	Data Type	Default Value
A2StartCallback	Function called at acquisition start up.	Function handle	@myActiveTwoStartCallback
A2StopCallback	Function called when acquisition ends.	Function handle	@myActiveTwoStopCallback
NewDataCallback	Function called the event that new data arrives.	Function handle	@myNewDataCallback
SynchErrorsCallback	Function called when synchronization errors occur.	Function handle	@mySyncErrCallback
BatteryLowCallback	Function called when the battery is low.	Function handle	@myBattLowCallback
CMSCallback	Function called when the CMS/DRL is out of range.	Function handle	@myCMSCallback
GSRCallback	Function called when the GSR is out of range.	Function handle	@myGSRNOIRCallback

Table 7 : ActiveTwo API callback function properties.

8.4 Timing properties

Property Name	Property Description	Data Type	Default Value
Period	USB driver frame rate in seconds	double	0.040
ScanRate	API frame rate in seconds	double	0.080
StartDelay	Delay	double	0
ExecutionMode	Execution mode of the API timer	string	'fixedRate'
BusyMode	Busy mode of the API timer	string	'queue'
BaseRate	<i>Read only</i> , minimum set Period	double	0.001

Table 8 : ActiveTwo API timing properties.

8.5 Message Notification Properties

Property Name	Property Description	Data Type	Default Value
CMSCtrl	CMS/DRL warning message control	boolean	1
GSR1Ctrl	GSR1 out of range warning message control	boolean	1
GSR2Ctrl:	GSR2 out of range warning message control	boolean	1
USBAccessNotification	USB Buffer delay access warning	boolean	0

Table 9 : ActiveTwo API message notification properties.

8.6 Quality Control Properties

Property Name	Property Description	Data Type	Default Value
QualityCheckEn	Enable the data Quality Check feature	boolean	0
QualityCheckFreq	Set the frequency of Quality Check	Double,seconds	4
QCSlowArtTimeWindow	Set the slow artifact time window.	Double, seconds	0.040
QCFlatChannels	Channel indexes with low amplitude or no activity	Vector double	[]
QCHighOffsets	Channel indexes with abnormally large DC offsets	Vector double	[]
QCNoise60Hz	Channel indexes with excessive power line noise	Vector double	[]
QCArtifactSlow	Channel indexes wandering indicative of a bad electrode	Vector double	[]
QCArtifactFast	Channel indexes exhibiting large and fast transitions indicative of a bad electrode	Vector double	[]
QCFFTDataBuffer	Buffered data for FFT analysis	Double,array	[]

QCSlowArtDataBuffer	Buffered data for temporal analysis	Double,array	[]
QCDCOffsets	Monopolar Channel mean Offset value	Vector double	[]

Table 10 : ActiveTwo API quality check properties.

8.7 Simulator Properties

Property Name	Property Description	Data Type	Default Value
Sim	Enable the simulator mode	boolean	0
SimFile	BDF file path to be simulated	Path string	''
SimPoint	Index or time point in seconds of the bdf file	Int32 or Double	[]
RewindEn	Play the file backwards	boolean	0
bdf	BDF file header	Structure	[]

Table 11: ActiveTwo API simulator properties.

8.9 Acquisition Data Properties

The following data acquisition properties are arrays that can be indexed as per a regular Matlab matrix array. The rows represent channels and columns are the data points.

Property Name	Property Description	Data Type	Default Value
MonData	Monopolar channel data	Array double	[]
ExgData	Touchproof channel data	Array double	[]
AnaData	Analog channel data	Array double	[]
GSR1	GSR sensor 1 data	Array double	[]
GSR2	GSR sensor 2 data	Array double	[]
ERGO1	ERGO sensor 1 data	Array double	[]
ERGO2	ERGO sensor 2 data	Array double	[]
Resp	Respiration sensor data	Array double	[]
Temp	Temperature sensor data	Array double	[]
Pleth	Plethysmograph sensor data	Array double	[]
DigitalIn	Digital Input port data	Boolean array [N x 16]	Zeros(N X 16)
StatusCh	Status Channel data	Array Int32	0
MonChnSelc	Monopolar channel selection vector	Vector double	[1:8]
ExChnSelc	Touchproof channel selection vector	Vector double	[1:8]
AnaChnSelc	channel selection vector	Vector double	[1:8]

Table 12: ActiveTwo data acquisition properties.

9. ActiveTwo API METHODS

Class methods are functions available to the user which may act on the **Active2** class, on input arguments, or a combination of both. To call these methods use the dot (‘.’) operator such as with the object properties. In addition, note that all methods are case sensitive.

Example: initialize the ActiveTwo API.

```
>> a2.init;
```

9.1 ActiveTwo API Initialization

Active2

Syntax

```
a2 = Active2
```

```
a2 = Active2('property1','value1', ... , 'propertyN','valueN');
```

Please see the section ## for detailed information for input properties. The output ‘a2’ for the ActiveTwo method is the specific ActiveTwo type object.

Example: create an Active2 type object with default settings.

```
>> a2 = Active2;
```

9.2 ActiveTwo Class Object Deletion

delete

To properly delete an existing **Active2** type object you must not use the clear commands since the **Active2** class contains object instances which do not vanish upon a clear command. Because of this the **Active2** class has special ‘clearing/deletion’ methods.

Syntax

```
del
```

```
delete
```

Example: delete an existing ActiveTwo type object. Three options are shown.

```
>> a2.del
```

```
>> a2.delete
```

```
>>delete( a2 )
```

```
>>
```

NOTE: MATLAB® will leave remnants of a destroyed object in the workspace. Please make sure to follow up with the ‘del’ method with a ‘clear’ command to completely erase any evidence of the **Active2** object.

Example: cleaning after Active2 object destruction.

```
>> clear a2;
```

```
>>
```

9.3 Active2 Class Configuration

These are methods load information onto the **Active2** class.

LoadConfig

Get user defined configuration settings from a BioSemi configuration '.cfg' file.

Syntax

LoadConfig

Example:

```
>> a2.LoadConfig;  
>>
```

Init

Initialize the **Active2** type object to load critical information from the ActiveTwo hardware and configuration file.

Syntax

init

Example:

```
>> a2.init;  
Driver opened.  
Start up Acquisition.  
Ringbuffer Initialized.  
USB Handshake Enable.  
<<< ACTIVETWO IS ON >>>  
MK2: 1, GSR16Hz: 1, Jazz: 1, Speed Mode: 4, BatteryLow: 0  
V20100224 using: MS winusb library, RAWIO, 40.34 msec stride, not sync'ing pointer calls.  
Disabled USB-Handshake. DAQ Stopped!  
ActiveTwo Driver closed.  
<<< ACTIVETWO IS OFF >>>  
<<< ACTIVETWO IS NOW INITIALIZED >>>
```

9.4 Active2 Class Driver

Use these methods to inspect and test the ActiveTwo driver. This method does not engage the data acquisition engine, it serves as a way to inspect that your hardware and software drivers are working.

On

Instruct the ActiveTwo USB driver to start up data acquisition.

Syntax

On

Example:

```
>> a2.On;  
>>
```

Off

Terminate the USB driver link. This function is particularly useful when the API does not terminate session correctly; you may use 'Off' to ensure that the USB driver terminates the session.

Syntax

```
LoadConfig
```

Example:

```
>> a2.LoadConfig;  
>>
```

DrvChk

Check on the existing driver for version and settings.

Syntax

```
DrvChk
```

Example:

```
>> a2.DrvChk;  
The Active2 Driver is loaded  
Driver & Buffer Information :  
V20100224 using: MS winusb library, RAWIO, 40.34 msec stride, not sync'ing pointer calls.
```

DrvCls

Force the USB driver link. Use DrvCls instead of Off when the USB driver is not responding, you may force terminate the link from your end.

Syntax

```
DrvCls
```

Example:

```
>> a2.DrvCls  
ActiveTwo Driver closed.
```

9.5 ActiveTwo Class SIMULATOR

LoadSim

Load the bdf file specified in the property 'SimFile' for simulation.

Syntax

```
LoadConfig
```

Example: delete

```
>> a2.LoadConfig;  
>>
```

Pause

Pause a running simulation. The API will report data and settings for current time course when paused.

Syntax

Pause

Example:

```
>> a2.pause;  
>>
```

Resume

Resume a paused simulation.

Syntax

Resume

Example:

```
>> a2.Resume;  
>>
```

Rew

Run a simulation bdf file in reverse order.

Syntax

Rew

Example:

```
>> a2.Rew;  
>>
```

9.6 Active2 Class Data Acquisition

start

Start a data Acquisition session.

Syntax

start

Example:

```
>> a2.start;
```

stop

Stop a data Acquisition session.

Syntax

stop

Example:

```
>> a2.stop;
```

9.7 ActiveTwo Class Inspection

disp

Display the contents of the **Active2** type object.

Syntax

```
disp(a2)
a2 is the ActiveTwo type object.
```

Example:

```
>> disp (a2)
```

get

Retrieve all **Active2** type object properties in a data structure format.

Syntax

```
x = get( a2)
x in this case is a data structure containing a snap shot of all the properties of the ActiveTwo type object a2.
```

Example: delete an existing Active2 type object. Three options are shown.

```
>>x = get(a2);
```

```
>>whos
```

Name	Size	Bytes	Class	Attributes			
a	1x1	60	ActiveTwo	x	1x1	51322	struct

9.8 ActiveTwo API license methods.

Write_c2v

Produce a personal 'c2v' file. To update an existing license you need to generate and send Cortech Solutions the 'c2v' file.

Syntax

```
Active2.Write_c2v
```

This action generates a personal 'c2v' file and stores in the current Matlab path.

Example:

```
>> Active2.Write_c2v
```

Created the 'c2v' file in:

C:\Matlab\ActiveTwo

Please send the 'c2v' file to Alex Fernandez-Villa at Cortech Solutions Inc.

sales@cortechsolutions.com

cortechsolutions.com

Write_fpr

Produce a personal 'fpr' file. To active a provisional license you need to generate and send Cortech Solutions the 'fpr' file.

Syntax

Active2.Write_fpr

This action generates a personal 'fpr' file and stores in the current Matlab path.

Example:

```
>> Active2.Write_fpr
```

Created the 'fpr', Local_user_ID.fpr, file in:

C:\Matlab\ActiveTwo

Please send the 'fpr' file to Alex Fernandez-Villa at Cortech Solutions Inc.

sales@cortechsolutions.com

cortechsolutions.com

License_keys

Retrieve installed ActiveTwo Matlab API licenses.

Syntax

Active2.License_keys

The method searches for software, SL, or USB-hardlocks, HL, licenses.

Example:

```
>> Active2.License_keys
```

Searching for C.S.I License keys.

Found 1 C.S.I License Keys.

(expand the Command Window if the numbers are not visible)

type: {'HASP-SL'}

license_key: {'338448472208'}

Update_License

Update user license from a Cortech solutions v2c file.

Syntax

Active2.Update_License(< v2c file path string >)

Example:

```
>> Active2.Update_License
```

Success...Updated your license.

10. ACTIVETWO API EVENTS & CALLBACK FUNCTIONS

The **Active2** class notifies the user of critical ActiveTwo events or when accessing erroneous **Active2** class data. MATLAB® Events are special objects which are invoked by the ActiveTwo API and are intimately linked to a particular set of user defined MATLAB® functions; called callback functions. When the Active API activates/fires an event, the call back function linked to the event in question will immediately be added to the MATLAB® stack to run.

Example : view Active2 class events.

```
>> e = events(a)
e =
    'UserNewData'
    'UserSynchErrors'
    'UserBatteryLow'
    'UserCMSNOIR'
    'UserGSRNOIR'
    'UserA2Start'
    'UserA2Stop'
```

10.1 ActiveTwo Class Events

Event Name	Event Description	Callback function Property	Callback function property value
'UserNewData'	New data arrival event	NewDataCallback	@myNewDataCallback
'UserSynchErrors'	Optical receiver synchronization error event	SynchErrorsCallback	@mySyncErrCallback
'UserBatteryLow'	Battery below %20 event	BatteryLowCallback	@myBattLowCallback
'UserCMSNOIR'	CMS/DRL not in range event	CMSCallback	@myCMSCallback
'UserGSRNOIR'	GSR sensors not in range	GSRCallback	@myGSRNOIRCallback
'UserA2Start'	ActiveTwo acquisition start event	A2StartCallback	@myActiveTwoStartCallback
'UserA2Stop'	ActiveTwo acquisition stop event	A2StopCallback	@myActiveTwoStopCallback

Table 14 : ActiveTwo API Events.

The ActiveTwo API includes the callback functions listed in table 14. You may modify/edit the existing functions or create your own callback functions. To assign a particular callback function to an Event Property you must can either state the function name as a string value or use the function's function handle with '@' operator. The body of the call back function must include the object and event structures passed by the Event structure fired.

Example: edit myCMSCallback.m, and see the input structures.

The structures 'a' and 'evnt' are passed by the internal Event object:

- o 'a' is 'copy' or handle of the **Active2** object
- o 'evnt' is a handle of the evnt structure.
- o

```
function myCMSCallback(a, evnt)
end
```

You may also utilize the Events with the 'addlistener' event function. There is an example of its usage in the 'ActiveTwoQuickSTART.m' script file line 343. In this case, the function 'DisplayBuffer' executes when the event 'UserNewData' fires. If you set the 'ScanRate' to 0.100 sec, then the 'UserNewData' event fires and the 'DisplayBuffer' function executes every 0.100 sec; see below the code which links an event to a function:

```
LHdisp = addlistener(a, 'UserNewData',@DisplayBuffer);
```

11. ACTIVETWO API DEMO SAMPLES

In the SampleCode you will see all off the sample code functions to get you started.

- 1) Implementing a variable size buffer for user defined processes. Please see the function 'DisplayBuffer.m' and in the ActiveTwoQuickSTART.m' script file line 343.
- 2) Displaying data with a Timer Object. Please the functions: 'DisplayCallBack.m', 'ActiveTwoDisplayObj.m', and lines of the 'ActiveTwoQuickSTART.m' script file.

12. ACTIVETWO API DAISY CHAIN MODE

The ActiveTwo hardware system allows for up to four ActiveTwo ADC boxes to be daisy chained yielding a total of 608 channels plus the trigger port. The standard ActiveTwo ADC box serves as the master ADC box, and its speed mode is set to zero. The subsequent ADC boxes have the daisy chain physical modification and will be assigned speed modes 1, 2, or 3.

When you set the speed mode to < 4 the ActiveTwo hardware and ActiveTwo API switch to the daisy chain mode; this is done independently of the number of ADC boxes in the chain. The API registers the speed mode from the one ADC box connected to the optical receiver (via the fiber optic cable).

The ActiveTwo API is designed to access data from one ADC box at a time, but you may switch from box to box in microseconds. To do so, you may set the **Active2** class property 'ADCbox' to dictate which ADC box you wish to access data from.

In the daisy chain mode each ADCbox can record up to 128 monopolar channels, 8 touch-proof channels, and 8 sensor channels. Note that you may still access all boxes in 'real-time' (with no data loss) in the following manner.

```
Example: Access all ADC boxes on an instantaneous fashion.
a2.ADCbox = 0; % master ADC box
data = a2.Mondata;
a2.ADCbox = 1; % The API is now accessing the 2nd ADC box.
data = [data ; a2.Mondata];
a2.ADCbox = 2; % The API is now accessing the 3rd ADC box.
data = [data ; a2.Mondata];
a2.ADCbox = 3; % The API is now accessing the 4th ADC box.
data = [data ; a2.Mondata];
```


13. ACTIVETWO API EMBEDDED FEATURES

13.1 Quality Control Tool

The Quality control feature is particularly useful since it periodically evaluates the Monopolar data in terms of:

- (i) abnormal high frequency variability of data.
- (ii) high amplitude with slow frequency variation.
- (iii) small signal changes or so called Flat Channels.
- (iv) amplitude of power line noise.

You may utilize this feature to quickly inspect your data before onset of an experiment or have it periodically inspect your data. You may interface to this feature programmatically or graphically, see table 15 for the Quality Control properties.

To enable this feature it is best that you call the 'start' method beforehand:

1. Set the 'QualityCheckFreq' in seconds. This property sets the frequency at which the API will execute the Quality Check.
2. At this point ensure that you are not exhausting matlab's graphing resources. Set the 'QualityCheckEn' to true or 1 to enable it.
3. The Quality Check panel will pop up, see figure 4.

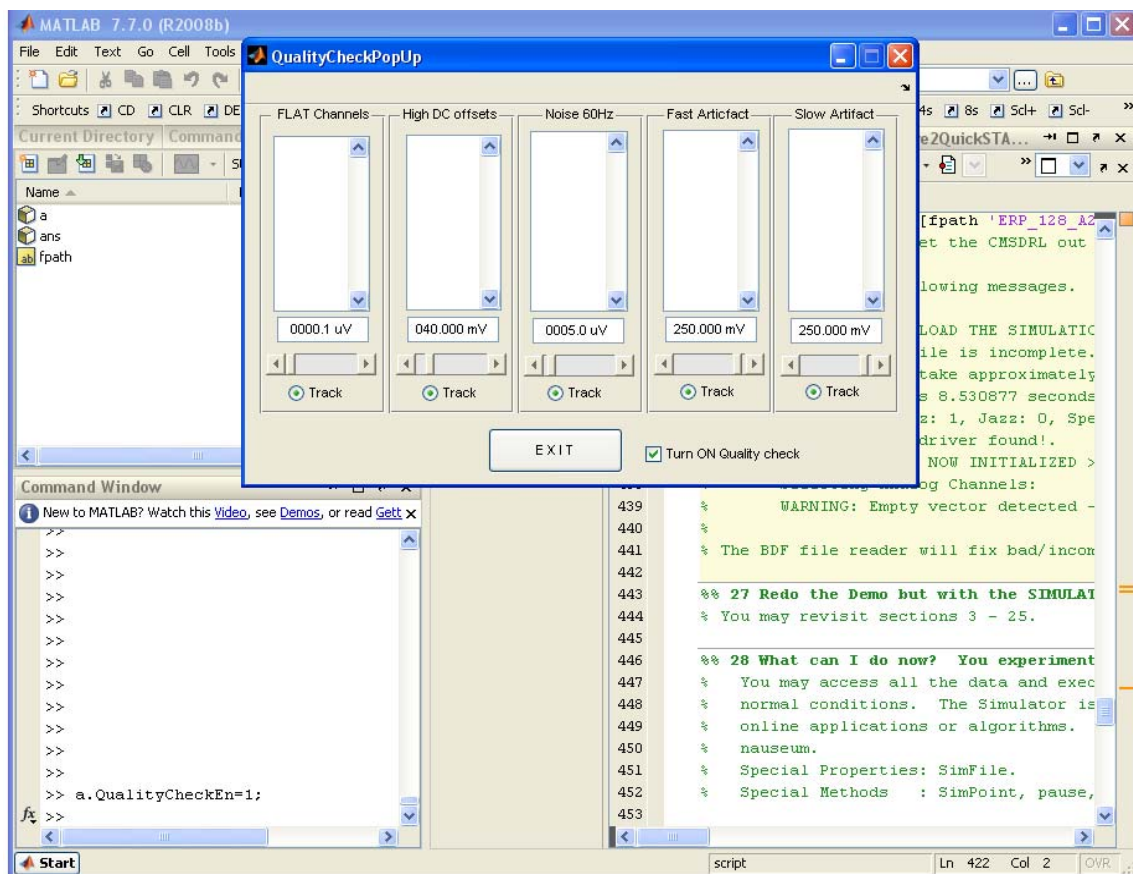


Figure 4: Quality Check Pop up window

The 'QualityCheckPopUp' window has list boxes with number boxes, sliders, and Track on radials below. The list box will show the channels by channel labels affected under the category shown above the list boxes. You may uncheck the 'Track' radial under any option you wish to not track. Use the sliders and number boxes to change the threshold which is shown on the number box.

Flat Channels

When the amplitude of channels fall below the set threshold, they are considered to be flat and will appear on this list.

High DC Offsets

When the channels' DC offset are larger than the set threshold, they are at high dc offset and will appear on this list.

Noise 60Hz

When the channels' 60 Hz power-line noise is above the set threshold, they higher than normal noise and will appear on this list.

Fast Artifact

When the amplitude of the channels rise above the threshold, the electrode might be damaged and will appear on this list.

Slow Artifact

When the amplitude of the channels rise above the threshold, the channels are wondering and will appear on this list.

To discontinue the Quality Check feature uncheck the box 'Turn ON Quality Check' and click exit, or you may do so programmatically by setting the 'QualityCheckEn' property to boolean false.

Example:

Change the Noise 60hz threshold to 2.6uV to show which channels have 60Hz noise amplitudes above 3uV. Change the DC offset threshold to 15mV. See figure 5.

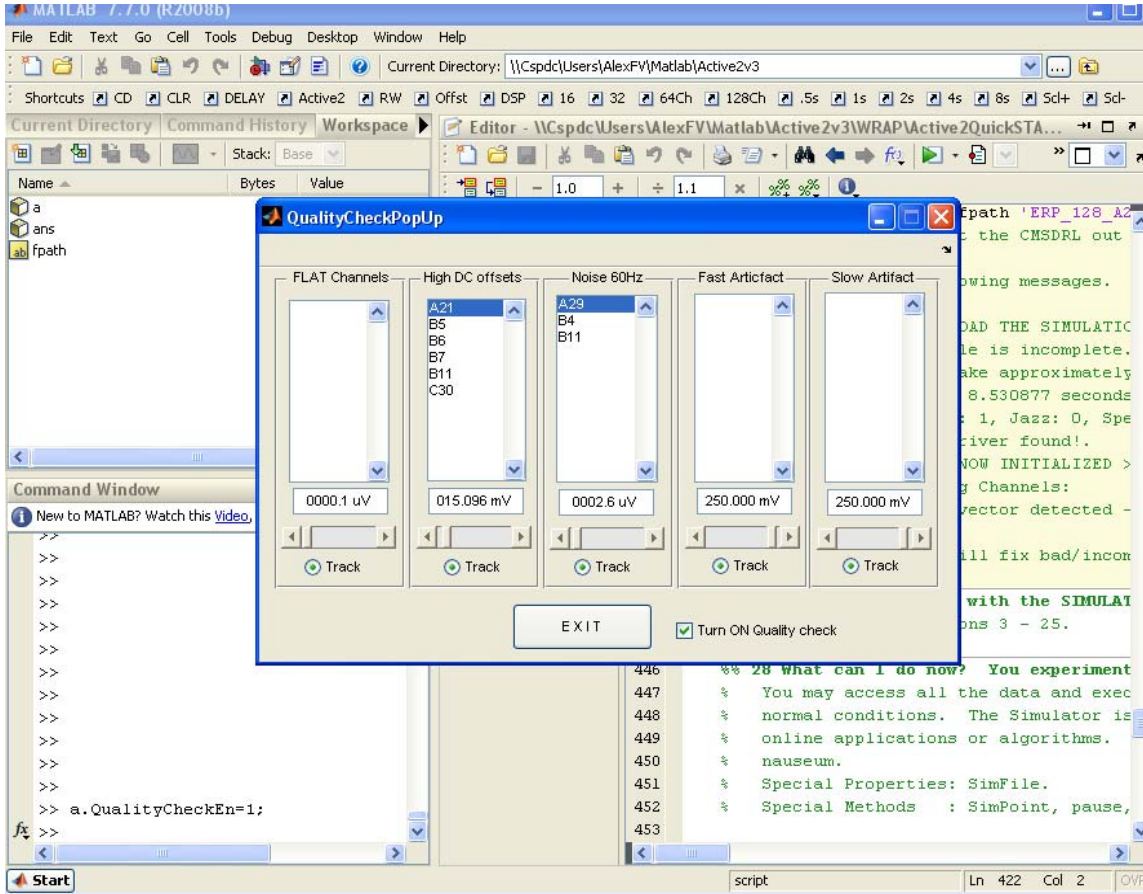


Figure 5: Quality Check Pop up window

Example: Inspect the DC offset and Noise 60Hz properties affected by the threshold changes above. These properties hold the channel indexes matching the channel labels shown in the Quality Check Pop Up window.

>> **a2.QCHighOffsets'**

ans =

21 37 38 39 43 94

>> **a2.QCNoise60Hz**

ans =

29 36 43

Property Name	Property Description	Data Type	Default Value
QualityCheckEn	Enable the data Quality Check feature	boolean	0
QualityCheckFreq	Set the frequency of Quality Check	Double, seconds	4
QCSlowArtTimeWindow	Set the slow artifact time window.	Double, seconds	0.040
QCFlatChannels	Channel indexes with low amplitude or no activity	Vector double	[]
QCHighOffsets	Channel indexes with abnormally large DC offsets	Vector double	[]
QCNoise60Hz	Channel indexes with excessive power line noise	Vector double	[]
QCArtifactSlow	Channel indexes wandering indicative of a bad electrode	Vector double	[]
QCArtifactFast	Channel indexes exhibiting large and fast transitions indicative of a bad electrode	Vector double	[]
QCFFTDDataBuffer	Buffered data for FFT analysis	Double, array	[]
QCSlowArtDataBuffer	Buffered data for temporal analysis	Double, array	[]

Table 15: list of the Quality Check feature properties.

13.2 ActiveTwo Simulator Tool

The ActiveTwo API Simulator tool is an indispensable feature for the laboratories who wish to realize their online analysis. The Simulator tool loads any BDF file and streams it directly into the API as if you were recording the experiment for a second time. Note that the ActiveTwo hardware is not connected to the PC anymore.

Now you can prototype and perfect your real-time/online analysis code ad nauseam. You may run the code through the particular point in time where your code may crash or where you would like to extract a specific feature.

To create an **Active2** type object in the Simulator mode, you must specify a BDF file at the onset of construction.

Example : create a Simulator mode ActiveTwo type object.

```
>> a2 = Active2 ('Sim', fpath);
```

Verifying License keys.

Found 1 C.S.I License Keys.

Found a HASP-Software-key, License number: 338448472208.

...searching for valid products...

Found 1 valid C.S.I licensed product(s):

'ActiveTwo API for MATLAB® Release 1'

ATTEMPTING TO LOAD THE SIMULATION FILE:

C:\MATLAB\R2007b\work\ActiveTwo\ActiveTwoClassImplementation\ActiveTwo\BDFfiles\ERP_128_A24_B8_B11_noisy&drifting.bdf.

Warning: The file is incomplete. New sample size, 81408 (318 seconds).

The file will take approximately 1.954863e+000 seconds to load.

Elapsed time is 2.646934 seconds.

MK2: 1, GSR16Hz: 1, Jazz: 0, Speed Mode: 4, BatteryLow: 0

<<< ACTIVETWO IS NOW INITIALIZED >>>

Selecting Analog Channels:

WARNING: Empty vector detected - Please Select Channels.

Acquisition in the Simulator mode

The Simulator behaves much like the ActiveTwo, except that you may only set the 'ScanRate' and not the 'Period'.

The 'start' method, will commence streaming data from the beginning of the file. You may also change the point at which to start the simulation. To do so, modify the 'SimPoint' property. You may enter the specific time point you where are interested in running the Simulation from. Enter this time point whether in the Simulation is halted or already running, either in terms of an index integer value or in terms of experimental time.

Example: commence simulation at 4.3 seconds after experiment onset.

```
>> a2.SimPoint = 4.3;
```

```
>>
```

Example: commence simulation at the 1000th point of the experiment.

```
>> a2.SimPoint = uint32(1000);
```

```
>>
```

At this moment, the ActiveTwo API is filled with all the information relating the time point specified. But 'SimPoint' must advance itself by one frame to address the next frame of data. The frame size is dictated by the 'ScanRate' and the 'SamplingRate'. To understand this concept read 'SimPoint' to inspect the point where the file is currently halted.

Example: load the simulation at 2 seconds. Then read 'SimPoint'.

```
>> a2.SimPoint = 2;
```

```
>> a2.SimPoint
```

```
ans =
```

```
94.7474 563.0073 2.1992
```

'SimPoint' reports a vector of three double numbers:

The first number is the percentage of file remaining. The second number is the file index where the Simulation will continue if the 'resume' method is executed. The third number is the file time-point version of the 2nd number. The 'ScanRate' is set to 0.200 seconds, the third number is then 2.00 secs (the time entered) + 0.200 secs (the scanrate) = 2.2.

Pause and Resume

Use 'pause/resume' to simply 'pause/resume' the Simulation. Use 'start/stop' much like pause/resume except that the 'start' method runs the simulation from the beginning of the BDF file.

Change the Simulation File

You may change the simulation file while already in the Simulator mode. To do so, change the file/path held by the 'SimFile' property and assert the 'LoadSim' property. After asserting 'LoadSim' the ActiveTwo API will automatically load the new BDF file.