B-Alert®

Programmer's Manual

Advanced
Brain Monitoring
advancedbrainmonitoring.com

Contents

1. B-Alert Software Architecture 4
2. B-Alert SDK Programming Interface 15
3. B-Alert Matlab® Programming Interface 66
4. B-Alert Client Programming Interface 94
5. B-Alert Server Programming Interface 120
6. B-Alert Maintenance Programming Interface 147
7. ABM Multi-Channel External Sync Unit (MC-ESU) Programming Interface 152

1. B-Alert Software Architecture

1. B-Alert Software Architecture

1.1 B-Alert Software Modules 7

1.2 Third Party Software Integration Options 8

1.3. B-Alert SDK State Machine 9

1.4 B-Alert SDK States 10

1.5 Event Synchronization and Extraction 11

1.5.1 Background 11

1.5.2 Multi-Channel External Synchronization Unit (MC-ESU) 11

1.5.3 Native SDK Integration 12

1.5.4 Datastreaming 13

1.5.5 Clock Server 13

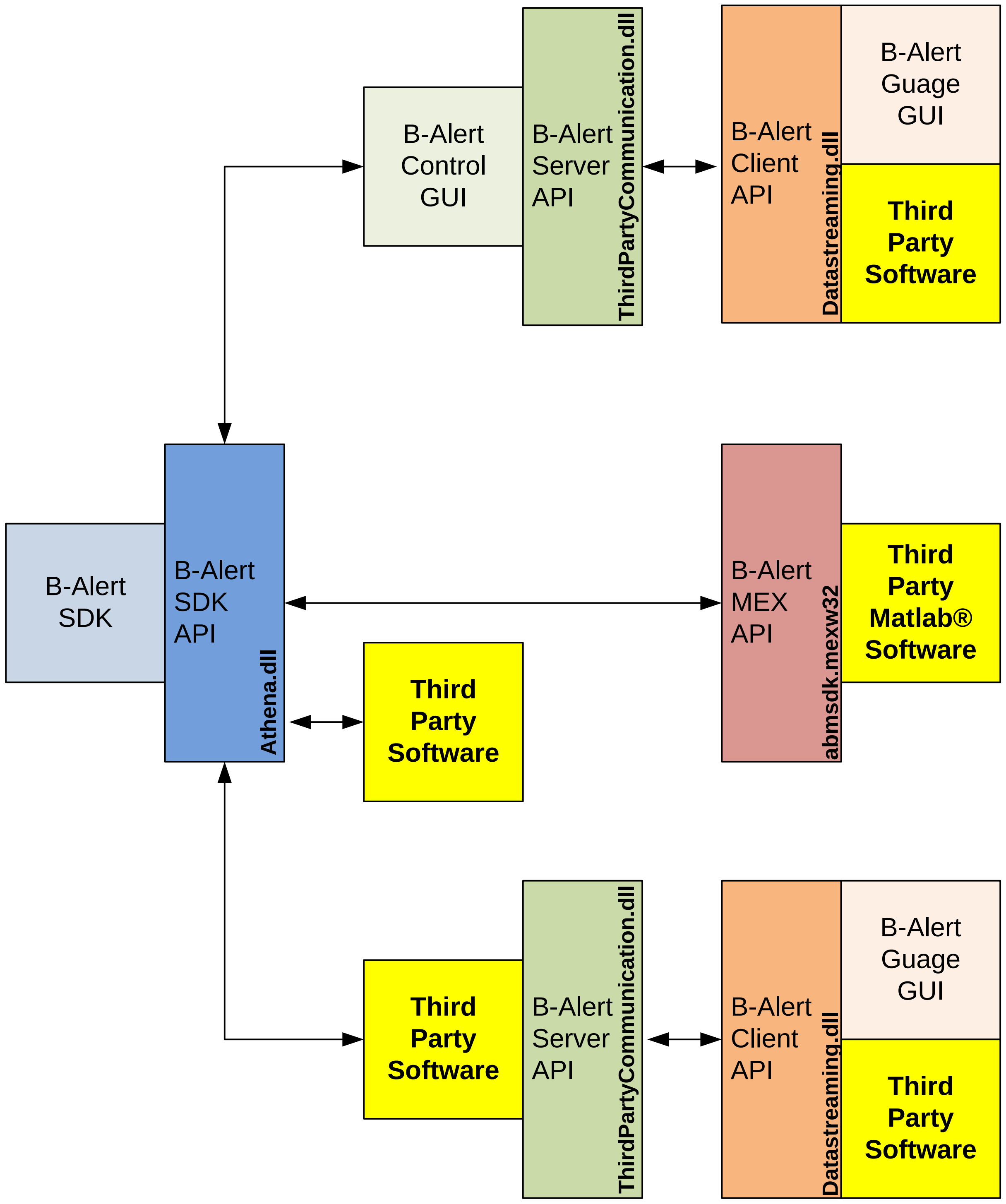


Figure1: B-Alert Software Architecture

1.1 B-Alert Software Modules

The B-Alert software has a layered architecture and is designed to be modular and portable. The modules are designed to function independently and interact with each other through proprietary ABM protocol over client-server architecture. Elaborate wrapper-functions are provided through the Application Programming Interface of each module such that all the messy details of the protocol are hidden from the programmer. The major modules are:

1. **B-Alert SDK:** The SDK is the core module of the B-Alert software that incorporates the bulk of the functionality. The sub-modules in the SDK are written in C++ and partitioned into libraries. All features in the SDK are accessible through well defined input-output Application Programming Interfaces (APIs). The SDK also incorporates the B-Alert Alertness and Memory Profiler (AMP) and B-Alert Batch computation module.

2. **B-Alert Control GUI:** The module serves as the link between the SDK and the B-Alert data presentation module (B-Alert Gauge GUI). It collects data from the SDK and forwards it to data presentation client(s) via the TCP/IP network. The module includes a backend TCP server (enclosed in ABM_ThirdPartyCommunication.dll) and a front-end GUI to select the control functionalities of the B-Alert software (such as, connect device, configure device, acquisition, test impedance, run AMP, generate report, invoke offline playback etc).

3. **B-Alert Gauge GUI:** The module incorporates the data presentation parts of the B-Alert software. It is written in C#, under .NET 3.5 framework and uses National Instruments library to present data in a user friendly format. The module includes a TCP/IP client (enclosed in ABM_Datastreaming.dlll) that interacts with the B-Alert Control GUI (server) to get raw and processed data from the SDK.

1.2 Third Party Software Integration Options

Third party applications can interface with the B-Alert Software at various levels (see Figure 1).

1. **B-Alert SDK API:** The B-Alert SDK can be interfaced standalone, without any other B-Alert modules, by incorporating the associated dlls and header files in the third party application. Obviously, in this configuration the third party application must provide the user interface to interact with the SDK and present data.

2. **B-Alert MEX API:** Using abmsdk.mex32 interface, the third party application can interact with the B-Alert SDK in a Matlab® environment.

3**. B-Alert Client API:** Third party applications can bypass the B-Alert data presentation module (B-Alert Gauge GUI) and interact directly with the B-Alert data server (B-Alert Control GUI) via TCP/IP network. Multiple such clients are supported simultaneously, if necessary. The Application Programming Interface is provided by ABM_Datastreaming.dll.

4. **B-Alert Server API:** For transparent control over data acquisition, third party applications can bypass the B-Alert Control GUI and serve as the link between the B-Alert SDK and the B-Alert data presentation module (B-Alert Gauge GUI). The Application Programming Interface is provided through the ABM_ThirdPartyCommunication.dll. In this configuration the third party application must provide the user interface to start, stop and configure sessions with ABM devices.

1.3. B-Alert SDK State Machine

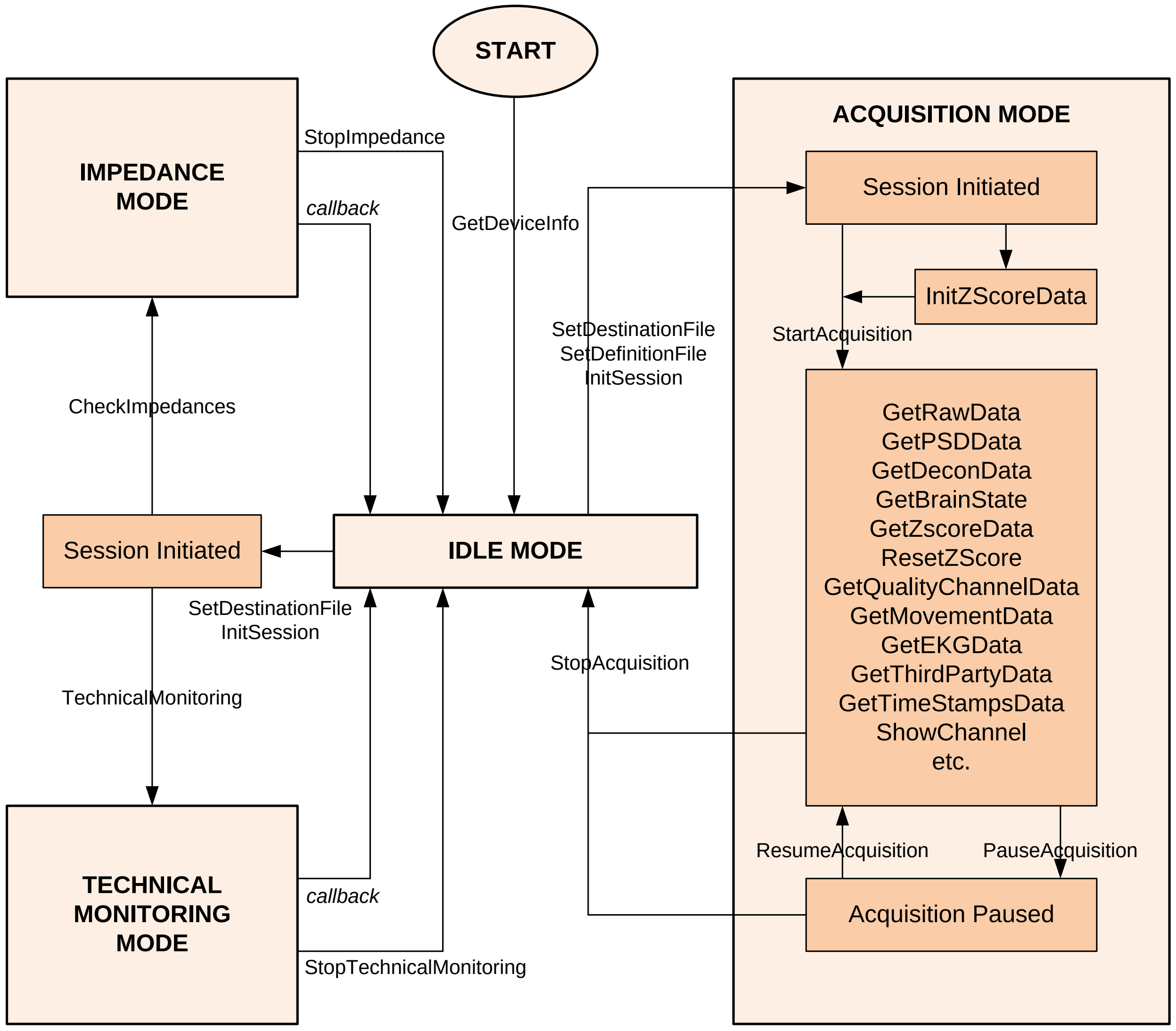


Figure2: B-Alert SDK State Machine

1.4 B-Alert SDK States

The B-Alert SDK follows defined states (Figure 2) and all APIs are expected to be called according to the rules of the state machine. Any violation in this regard could result in wrong output values and/or erroneous behavior.

1**. Start:** It is assumed that the ABM device is connected and synced to the ABM receiver (in case of wireless configuration) prior to the Start state. Syncing is a one-time procedure and in most cases the ABM device is delivered in a synced state with the ABM receiver. If syncing has to be re-established, it is highly recommended to use the B-Alert Control GUI rather than the B-Alert Maintenance APIs listed in this document. The device connection status can be checked using GetDeviceInfo API.

2. **Idle:** After the device is connected and the communication port is opened and active, the SDK remains in Idle-state until InitSession API is invoked.

3. **Impedance:** Impedance evaluation must be performed when data is NOT acquired from the device. It is usually invoked at the beginning of a study session, before data acquisition. In case impedance has to be checked during a session, data acquisition must be stopped before invoking impedance monitoring. In addition, a new session must always be initiated (using InitSession API) before calling Impedance APIs. The impedance computation takes several seconds (~5 sec / channel), thus it is designed to be non-blocking. The user can however exit by calling StopImpedance API if necessary. Exiting before completion could result in erroneous/incomplete impedance values in many channels. The SDK switches to Idle-state after impedance computation is completed.

4. **Technical Monitoring:** Similar to Impedance Monitoring, data acquisition must be stopped before invoking Technical Monitoring and a new session must be initiated (InitSession) before calling the tech monitoring APIs. Technical Monitoring is a computationally intensive process and may continue for several seconds in excess to the time requested in the API. The state is non-blocking and the user can exit anytime (by calling StopTechnicalMonitoring API), but the interruption could result in erroneous outputs. The SDK switches to Idle-state after Technical Monitoring is completed.

5. **Acquisition:** The majority of the APIs in the SDK are accessible in the Acquisition-state. A new session has to be successfully initiated before entering Acquisition-state. The parameters for Z-Score computation are usually initialized before starting acquisition. The data acquisition starts with StartAcquisition API and various other APIs can be used to access raw as-well-as processed data. The acquisition can also be temporarily paused and resumed without initiating a new session. SDK switches to Idle-state at the end of the acquisition (StopAcquisition API).

1.5 Event Synchronization and Extraction

1.5.1 Background

The B-Alert software provides a multitude of options for third party developers to synchronize, extract and analyze external events with respect to the physiological signals and coginitive metrices generated by the B-Alert system. There are several issues that impact the implementation choice, the most important being the synchronization accuracy between the physiological and test environment. Because discrete events of interest (i.e., presentation of a stimulus, onset of a task, response to a presentation, or a fixation related event) occur randomly while data is acquired rather than temporally timed to the exact start of a one-second epoch, event related analyses are only possible when the presentation/ recognition of information are synchronized with the EEG recordings.For example, Event related potential (ERP) analysis is a procedure commonly applied to the raw data associated with onset or response to an event (i.e., P300). The specificity of off-line ERP analyses is impacted by how closely and consistently the events and EEG are synchronized. The more embedded the synchronization approach, the less likely the operating system(s) and/or network protocols contribute to alignment noise. Four synchronization solutions are described below.

1.5.2 Multi-Channel External Synchronization Unit (MC-ESU)

The ESU is hardware solution designed by ABM to provide the most accurate means to provide third party synchronization with minimal software development effort (Figure-3). The ESU is designed to obtain and time stamp up to four serial and one parallel port inputs from third party applications in combination with the digitized signals from the Sensor Headset. For this deployment, the third party modifies its software application to transmit a time stamp or other marker to the ESU via their serial or parallel port and simultaneously save their time stamp in their data file. The third party time stamp is then synchronized with the sensor headset data as both arrive at the ESU and saved in the EEG file. In this mode, the only variability in an exact synchronization is attributed to the Windows operating system on the third party's computer which is transmitting their time stamp. Tests suggest that this delay could range between 0 and 30 milliseconds depending on CPU processing load and how the third party implemented their time stamp (i.e., parallel or serial port and embedded software priority assigned to the time stamp transfer routine). More details about programming the ESU can be found in Section-7.

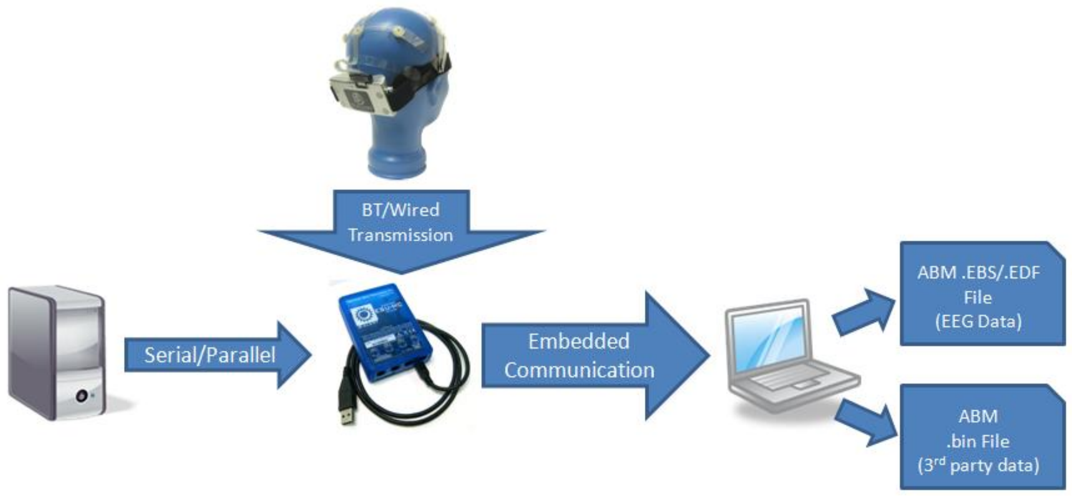


Figure3: Synchronizing via MC-ESU

1.5.3 Native SDK Integration

This configuration is most effective for real-time applications, especially involving closed-loop feedback between the stimulsus and physiological meterices. The B-Alert SDK provides an extensive application programming interface that can be used to integrate with third party softwares (see section-2). The SDK operations are called from the third party software and both applications must run on the same computer. The B-Alert real time algorithms require significant resources (both RAM and CPU and if the third party software is also resource demanding, then a computer must be selected which is adequately powered. Given the software applications are independent, multi-threading and other innovations that would otherwise improve resource load is not easily implemented. When the resource demands from both applications exceed the current technology, separate computers are requested and a means to synchronize the data for off-line analysis must be employed.

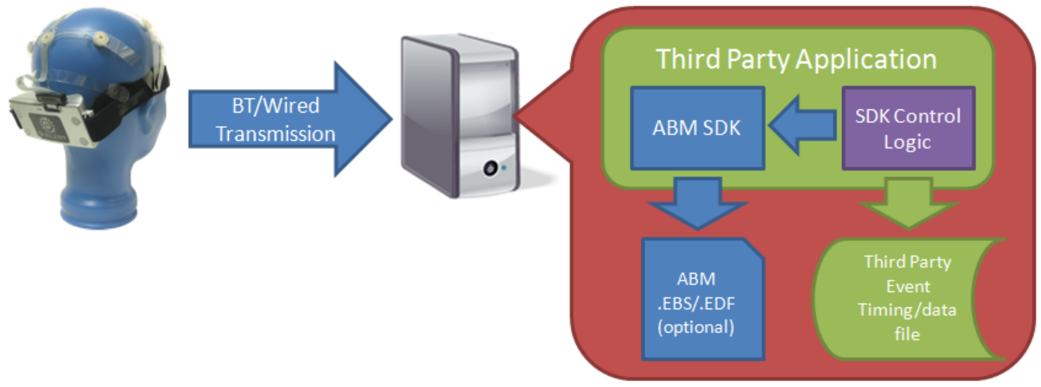


Figure4: Synchronizing via SDK for Real-Time Applications

1.5.4 Datastreaming

For resource intensive third-party applications, two sepearte computers connected over the nework can be used. The data streaming solution requires the third party to implement a dynamic link library (DLL) client to acquire data provided by ABM and save it their data file (Figure 5). Two options are provide for data streaming. The first option passes only time stamp information from the headset via UDP/IP. UDP/IP is preferable to TCP/IP for synchronization because it reduce the influence of delays attributed to network communication and traffic to the extent practical for this synchronization approach. The limitation of UDP/IP is that only the time stamp can be passed without risk of lost data. The second data streaming option and the least accurate method for synchronization uses a TCP/IP protocol. Data streaming "get" commands allow the third party client to select raw or decontaminated data, power spectra density values for each channel and/or the B-Alert results for transfer. These data are placed in a buffer which holds up to 60 seconds of data.

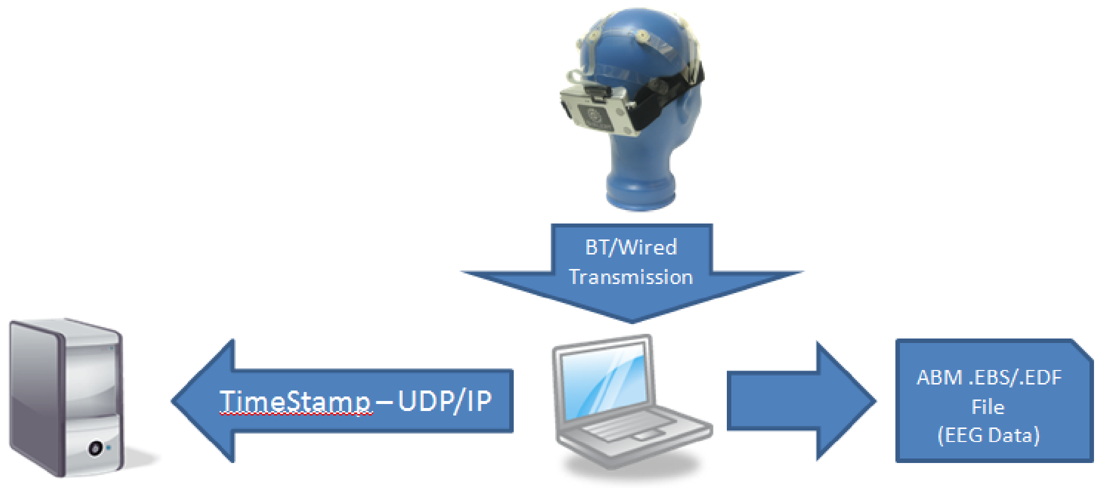


Figure4: Synchronizing via the network using Datastreaming

1.5.5 Clock Server

Clock Server synchronization is used in ABM's Teaming Platform. The system clock server provides a means to integrate a common time stamp across multiple computers (Figure 5). This approach should be used to synchronize simultaneously acquired data with multiple headsets and across event scenarios. The most common application for this is during group dynamics assessment or training when participants are provided unique event scenarios. In order to optimize the accuracy of the time stamp, the computer hosting the system clock server should be limited to minimal computational and graphic demands. The time stamp obtained from the clock server should be incorporated into the third party's data file so the data can be subsequently aligned for off-line analyses.

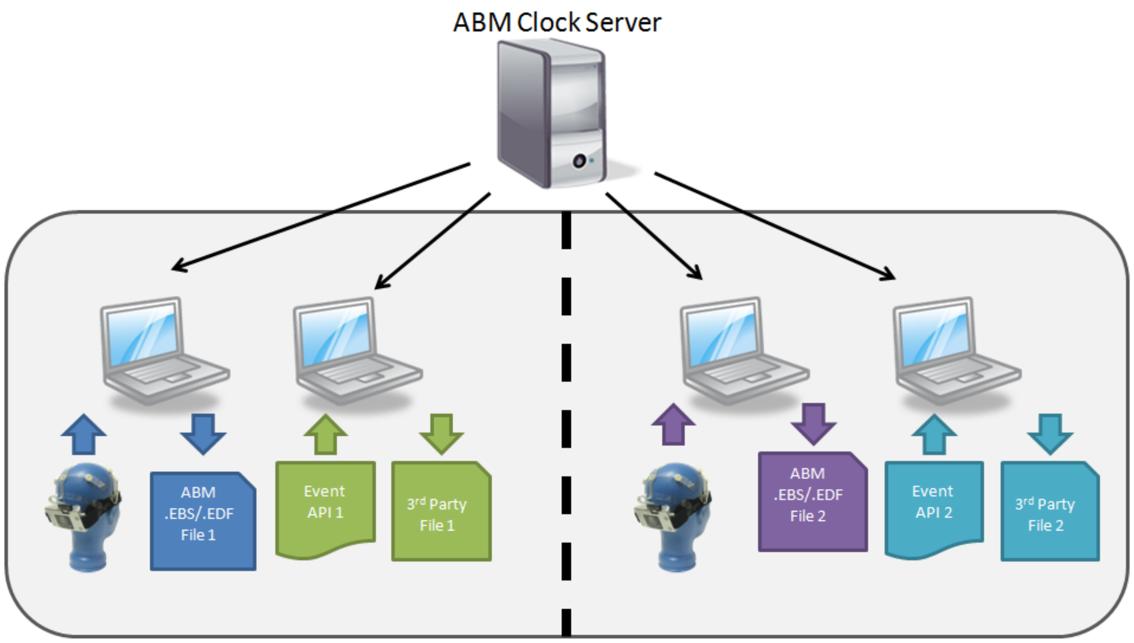


Figure5: Synchronizing via the Clock Server in ABMs Teaming Platform

2. B-Alert SDK Programming Interface

2. B-Alert SDK Programming Interface

2.1 Programming Notes 18

2.2. Validation 19
2.2.1 Scope 19
2.2.2 Procedure 19
2.3 Application Programming Interface 21
GetDeviceInfo 21
SetDestinationFile 22
SetDefinitionFile 23
InitSession 24
GetChannelMapInfo 26
CheckImpedances 28
StopImpedance 29
TechnicalMonitoring 30
StopTechnicalMonitoring 31
StartAcquisition 31
PauseAcquisition 32
ResumeAcquisition 33
StopAcquisition 33
GetRawData 34
GetFilteredData 35
GetDeconData 36
GetPSDData 37
GetPSDDataraw 38
GetBandOverallPSDData 39
GetBandOverallPSDRawData 40
GetPSDBandwidthData 41
GetPSDBandwidthRawData 42
GetBrainState 43
InitZScoreData 44
GetZScoreData 45
ResetZScoreData 46
GetQualityChannelData 46

GetMovementData 47
GetTiltRawData 48
GetTiltAnglesData 49
GetEKGData 50
GetThirdPartyData 51
GetTimeStampsStreamData 52
GetCurrentSDKMode 53
ShowChannel 54
AlignChannels 55
SetArtifactsCallbackFuncs 55
RegisterCallbackImpedanceElectrodeFinished 56
RegisterCallbackImpedanceElectrodeFinishedA 57
RegisterCallbackDataArrived 58
RegisterCallbackOnStatusInfo 59
RegisterCallbackOnError 60
RegisterCallbackMissedBlocks 61
GetClassMu 62
GetClassMidline 63
GetClassPrefrontal 64
2.4. SDK Error / Warning Codes 65
2.4.1 Errors 65
2.4.2 Warnings 66

2.1 Programming Notes

1. Files: All the files necessary for integrating with the SDK can be found in ABM\EEG\SDK folder of the installation. The necessary files and their functionalities are as follows:

i) Dll files – Even though the SDK has a modular architecture, the dlls cannot be separated currently. All the dlls (in SDK/bin folder) have to be included and must in the same path as the exe file. However, only ABM_Athena.dll needs to be loaded in the third-party program. All other dlls (in the same path) will be referenced by ABM_Athena.dll.

ii) Lib files – the lib files corresponding to the dll files can be found in SDK\lib folder

iii) Header files – All the header files are essential, but only ABM_Athena.h needs to be referenced in the third-party code.

iv) Other files – It is highly recommended that the third-party application reference the path to all the files in SDK/bin folder. Some of the other essential files are: 1) xml – the initialization parameters of the SDK are specified in xml files, 2) txt – many computation coefficients are specified in txt files, etc.

2. Building the project: Depending on the application, there are two ways to build the integrated-project:

i) Run-Time Dynamic linking: Windows functions such as LoadLibrary & GetProcAddress can be used to get a handle and address of the dll. In this option the dlls are loaded during runtime and the lib-files are not necessary at the time of building the code.

ii) Load-Time Dynamic linking: In this type of programming the lib files have to be included in the project at the time of building the code. The lib files may be included manually (through IDE) or via compiler directives such as *#pragma comment.*

3. Windows.h: The client application must define *windows.h* header file, unless it is internally referenced (such as in MFC based windows applications).

4. Init files: Initialization parameters for the SDK are specified in xml files (we don't recommend editing them), and they are accessed every time the SDK is executed. The path to the xml files must be referenced in the third-party application (or use default path as the exe). However, for JAVA and MATLAB applications the xml files have to be copied to the exe-engine folder (which is not necessarily the same as the exe folder). This inconvenience will be rectified in future releases by referencing the registry.

5. Reference program: Sample client programs (written in C++ and C#) are included in the installation for reference (SDK/SampleClients).

2.2. Validation

2.2.1 Scope

This section briefly describes the recommended validation procedures using simulated hardware, after the third party software is integrated with the B-Alert SDK. Note that some features such as impedance measurement, technical monitoring, etc need real hardware and cannot be validated using simulated hardware.

2.2.2 Procedure

EBS Remote Player is a utility application that can be found in the installation (ABM\Utilities\Remote Player) and can be used to simulate ABM hardware. The application reads data from ABM data files (EBS or EDF format) and simulates the corresponding ABM device (X4, X10, X24) based on the type of data file used. The steps to setup the EBS Remote Player are described below:

1. **Wiring the connection:** Both B-Alert SDK and EBS Remote Player communicate via Virtual COM ports (registered by USB-to-Serial Converters). The following cables are essential in order to setup the connection: i) USB-to-Serial cable (2) – two USB-to-Serial cables must be used in order to open two Virtual COM ports in the computer. One of each will be used by the Remote player and the SDK. ii) Null modem cable (1) – a null-modem cable must be used to bridge the two USB-to-Serial cables (see Figure 6). It is recommended that both B-Alert SDK and the EBS Remote player be executed in the same computer.

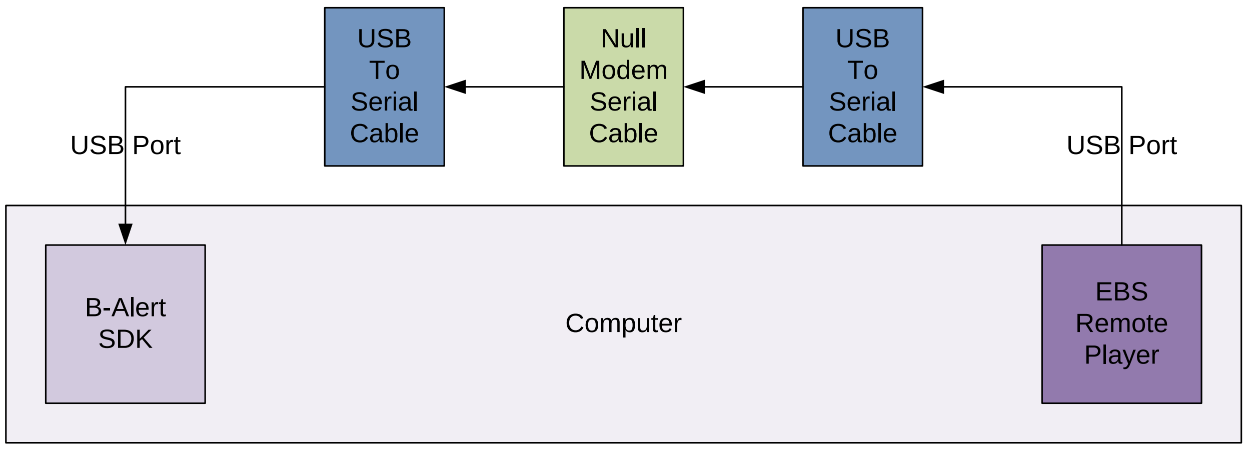


Figure 6: Connecting EBS Remote Player to B-Alert SDK

2. EBS Remote Player Interface

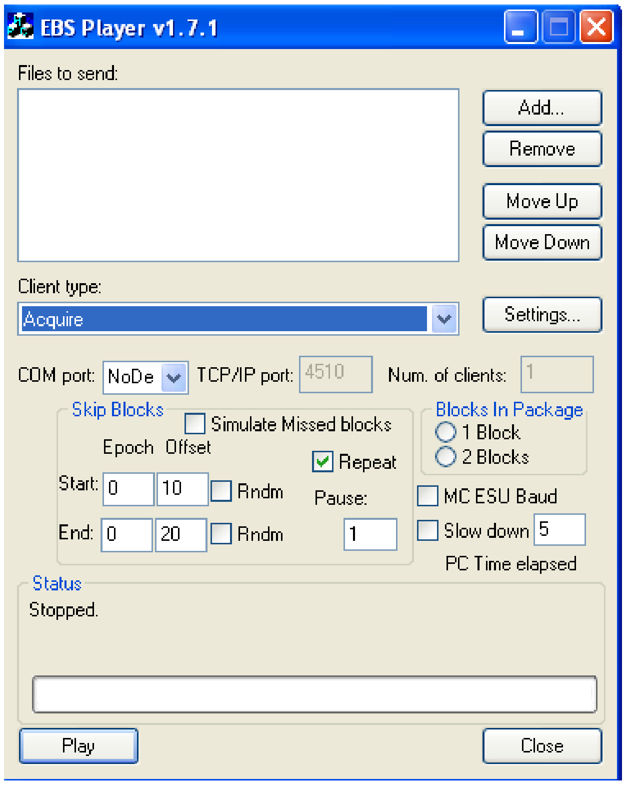


Figure 7: EBS Remote Player Interface

- Select ABM EBS or EDF file using add button. Gold standard data files for ABM devices can be found in ABM\EEG\Data folder (1830_1=X10, 1830_2=X24, 1830_3=X4 B-Alert, 1830_4=X4 APPT). Multiple EBS/EDF files can be selected or the same file can be selected multiple times in order to increase the duration of the session. The application will play all files in a sequential order without interruption.
- Select COM port (Virtual port established by the USB-to-Serial converter)
- Select Blocks In package = 2 Blocks (for X10,X4), 1 Block (for X24)
- Click Play. Monitor the progress bar for status of the data transmission.
- Start B-Alert SDK. The SDK will use the second Virtual COM port established by the USB-to-Serial converter.
- Note: The default sampling of ABM devices is 256 samples/second. The EBS Remote Player mimics the sampling rate using windows timers which may vary depending on the processing capability of the computer. In faster computers the timing can be adjusted using the “Slow down” feature. Note that the SDK may behave abnormally if the sampling rate is too slow or too fast. In most cases “Slow down” will not be necessary.

3. **Output files**
   The final step in the validation involves comparing the output from the SDK with the gold standard files provided in the installation. The simulated hardware cannot entirely replicate all the features of the real-hardware accurately, thus slight variations in the data outputs should be expected. Nevertheless it should still maintain a high level of similarity with almost all identical values. Gold standard files are provided for all devices and can be found in ABM\EEG\Data folder (1830_1=X10, 1830_2=X24, 1830_3=X4 B-Alert, 1830_4=X4 APPT).

2.3 Application Programming Interface

GetDeviceInfo

Description:
Checks whether ABM device is properly connected to ABM receiver. Returns information about the connected device.

```
Format:
_DEVICE_INFO* GetDeviceInfo(_DEVICE_INFO* pDeviceInfo = NULL);
```

Input Arguments:

1. Type: _DEVICE_INFO*
   Obsolete, kept for backward compatibility

Output Arguments:

1. Type: Pointer to _DEVICE_INFO

```
typedef struct _DEVICE_INFO {
        char chDeviceName[256];          //Device Serial/Type
        int    nCommPort;                       //COM Port Number
        int    nECGPos;                          //Position of EKG Channel(0=1st
channel)
        int    nNumberOfChannel;           //Number of Channels
        int    nESUType;                      //Type of Receiver (Single-
Channel=1/Multi-Channel=0)
        int    nTymestampType;             //Type of Timestamp (System time=1
/ ESU time=0)
        int    nDeviceHandle;                  //Reserved
        char chDeviceID[MAX_PATH];   //Reserved
}
```

PseudoCode:

```
_DEVICE_INFO* pInfo = GetDeviceInfo();
If(pInfo == NULL)
    //failed
else
   //success
```

Notes:

It is HIGHLY RECOMMENDED that this function be called as the first step in the initialization process in order to check the device state.

SetDestinationFile

Description:
Informs SDK the path and name of the data file(s).

Format:

```
Int SetDestinationFile(char* pDestinationFile)
```

Input Arguments:

1. Type: char*
   pDestinationFile: Full (absolute) path to destination file

Output Arguments:

1. Type: int
   ```
   TRUE          //Path was successfully set
   FALSE         //Failed
   ```

PseudoCode:

```
If (SetDestinationFile( "C:₩₩ABM₩₩EEGTest₩₩SDK₩₩TestSDKCpp₩₩Output
Files₩₩123456789.ebs" ))
  //success
else
  //failed
```

Notes:

The ebs files are usually named based on session #(4-digit), group #(1-digit), iteration # (1-digit), task # (2-digit), task iteration #(1-digit).

The SDK also creates many comma separated variable (csv) files as-well-as binary (bin) files in the same path. If the destination path is not set, none of the output files will be created.

SetDefinitionFile

Description:
Informs SDK the path and name of the definition file.

Format:
```
Int SetDefinitionFile(char* pDefinitionFile)
```

Input Arguments:
1. Type: char*
   pDefinitionFile: Full (absolute) path to definition file

Output Arguments:
1. Type: int
```
   TRUE          //Path was successfully set
   FALSE         //Failed
```

PseudoCode:
```
If (SetDefinitionFile( “C:₩₩ABM₩₩EEGTest₩₩Data₩₩8509_1₩₩8509_M7003500_A1.def” ))
  //success
else
  //failed
```

Notes:

The Definition file contains individualized information on each subject and is necessary for the computation of B-Alert classifications (Session Types: ABM_SESSION_BSTATE & ABM_SESSION_WORKLOAD). It is very important to use the correct definition file when performing classifications – otherwise results will not be valid.

The Definition file is created using baseline sessions. Graphical interface to create the definition file is not available through the SDK but can be invoked using B-Alert software (recommended) or through a utility called BAlert-Batch invoked through the Windows command line.

The definition file is mandatory for Session Types: ABM_SESSION_BSTATE & ABM_SESSION_WORKLOAD

InitSession

Description:
Initializes SDK to begin a NEW session.

Format:

```
Int InitSession (int nDeviceType,int nSessionType,int nSelectedDeviceHandle,
                        BOOL bPlayEBS)
```

Input Arguments:

1. Type: int
   nDeviceType : informs SDK which device configuration to use
   Values:

```
ABM_DEVICE_X24Qeeg [0],
ABM_DEVICE_X24Standard [1],
ABM_DEVICE_X10Standard [2],
ABM_DEVICE_X4BAlert [3],
ABM_DEVICE_X4APPT [4],
```

2. Type: int
   nSessionType: informs SDK which session configuration to use
   Values:

```
             ABM_SESSION_RAW [0]            //For RAW & RAW-PSD data
             ABM_SESSION_DECON [1]        //Additional DECON, DECON-PSD data
             ABM_SESSION_BSTATE [2]       //Additional Brain State
Classification data
             ABM_SESSION_WORKLOAD [3] //Additional B-Alert Workload data
```

3. Type: int

nSelectedDeviceHandle
Value: -1 //Reserved

4. Type: bool
   bPlayEBS
   Value: 0 //Reserved

Output Arguments:

1. Type: int

```
INIT_SESSION_OK                                   //Session successfully
initiated
INIT _SESSION_NO                                  //Session initiation
failed
ID_WRONG_SEQUENCY_OF_COMMAND   //command was ignored
```

PseudoCode:

```
If (InitSession (ABM_DEVICE_X10Standard, ABM_SESSION_RAW, -1, 0) == INIT_SESSION_OK)
  //success
else
  //failed
```

Notes:

When using session type = ABM_SESSION_BSTATE or ABM_SESSION_WORKLOAD, SetDefinitionFile should be invoked and definition file should be selected previously.

GetPacketChannelNmbInfo

Description:
Gets the number of channels represented in the following APIs : GetRawData, GetDeconData, GetPSDData, GetPSDDataraw, GetQualityChannelData,

Format:

```
int GetPacketChannelNmbInfo(int& nRawPacketChannelsNmb, int&
nDeconPacketChannelsNmb, int& nPSDPacketChannelsNmb, int&
nRawPSDPacketChannelNmb, int& nQualityPacketChannelNmb);
```

Input Arguments:

1. Type: &int
   nRawPacketChannelsNmb – channels represented in GetRawData
2. Type: &int
   nDeconPacketChannelsNmb – channels represented in GetDeconData
3. Type: &int
   nPSDPacketChannelsNmb – channels represented in GetPSDData
4. Type: &int
   nRawPSDPacketChannelNmb – channels represented in GetPSDDataraw
5. Type: &int

nQualityPacketChannelNmb – channels represented in GetQualityChannelData

Output Arguments:

2. Type: int
   SUCCESS / FAILED

PseudoCode:

Notes:

This API is accessible only AFTER InitSession and BEFORE StartAcquisition.

The following values are returned by the API, depending on the type of the device connected.

Device	GetRawData	GetDeconData	GetQualityChannelData	GetPSDDataraw	GetPSDData
X4 APPT	4	4	3	3	3
X4 B-Alert	4	4	2	2	2
X10 Std	10	10	9	9	9
X24 Std	24	24	20	20	20

GetChannelMapInfo

Description:

Gets information about the channel map of the devices. It includes number of channels, channel names, availability of auxilarity channels, session type etc.

Format:

```
int GetChannelMapInfo (_CHANNELMAP_INFO & stChannelMapInfo)
```

Input Arguments:

1. Type: _CHANNELMAP_INFO
   nDeviceTypeCode: (0=X10Std, 1=X4APPT, 2=X4BALERT, 3=X24STD
   nSize: Number of analog channels in device (X4=5, X10=10, X24=24)
   _EEGCHANNELS_INFO: Structure with fields –
      cChName[24][20] – 20 bytes long channel names for max 24 channels
      bChUsed[24] – channels used in the device (1=used, 0=unused)
      bChUsedInQualityData[24] – channels used in technical monitoring
      BOOL bChCanBeDecontaminated[MAX_NUM_EEGCHANNELS];

```
                BOOL     bIsChEEG[MAX_NUM_EEGCHANNELS];

    _ELECTRODES_INFO –
            int    nNumElectrodes;
            int    nStabilization;
            int    nAgregationsSamples;
            int    nCurrentType;
            char   cElName[MAX_NUM_ELECTRODE][MAX_LENGTH_CHANNEL_NAME];
            //[in]the name of electrode max 20 characters
            int    nElCommand[MAX_NUM_ELECTRODE];
            // Impedance command to be sent for this electrode to be measured
            int    nElChannel[MAX_NUM_ELECTRODE];
            // EEG channel to be used when measuring electrode
            int    nElReferentialElectrode[MAX_NUM_ELECTRODE];
            // Electrode to be used when substracting ref el. (-1 for none)
    _AUXDATA_INFO – structure with fields
                bIred – Optical channel (1=available, 0=absent)
                bred – Optical channel (1=available, 0=absent)
                bTilt – accelerometer channel (1=available, 0=absent)
                bEcgIndex – index of ECG channel
                bMic – microphone channel (1=available, 0=absent)
                bHaptic – haptics channel (1=available, 0=absent)
    -HARDWARE_INFO – structure with fields
                int nBatteryMax; //millivolts
                int nBatteryMin; //millivolts
                int nTiltLinearTransformA;
                int nTiltLinearTransformB;
    _SESSIONTYPES_INFO – structure with fields
                 bDecon – artifact decontamination (1=enabled, 0=disabled)
                 bBalert – BAlert calssification (1=enabled, 0=disabled)
                 bWorkload – Workload classification (1=enabled, 0=disabled)
```

Output Arguments:

2. Type: int
   SUCCESS / FAILED

PseudoCode:

Notes:

This API is accessible only AFTER InitSession and BEFORE StartAcquisition.

CheckImpedances

Description:
Checks and reports impedance values of all EEG channels.

Format:
```
int  CheckImpedances ( _stdcall* callback(ELECTRODE*, int&), _DEVICE_INFO*
pDeviceInfo=NULL );
```

Input Arguments:
1. Type: Pointer to callback function
   callback(ELECTRODE* pEl, int& nCount): impedance checking is time consuming and takes approx 5 sec / channel, hence the callback function is non-blocking and will be automatically invoked by the SDK.
   Parameters of the callback are:
   i) Type: Pointer to ELECTRODE*
   ```
     Typedef struct ELECTRODE {
         char chName[20]; //channel name
         bool bImpedance; //status (1=impedance is good & <45kOhm, 0=bad)
         float fImpedanceValue; //value, 99999 if unable to calculate OR invalid value
     }
   ```
   ii)Type: Address to int
   nCount: Number of electrodes

2. Type: Pointer to _DEVICE_INFO
   NULL //Reserved

Output Arguments:
1. Type: int
   ```
   IMP_STARTED_OK                      // CheckImpedances started
   IMP_STARTED_NO                      // failed
   ID_WRONG_SEQUENCY_OF_COMMAND        // command was ignored
   ```

PseudoCode:
```
static void _stdcall callback(ELECTRODE *pEl, int& nCount) {}
int ret;
if(ret= CheckImpedances(callback,NULL))
   //success
```

Notes:

CheckImpedances function must be called only after a new session is initiated using InitSession and when data acquisition is not running prallely.

Impedance computation takes approx 5sec/channel and CheckImpedance function reports the values through a non-blocking callback only after the computation of all the channels. In order to get responses from individual channels please use RegisterCallbackImpedanceElectrodeFinished.

StopImpedance

Description:
Halts impedance computation.

Format:
```
int  StopImpedance ()
```

Input Arguments:
None

Output Arguments:
1. Type: int
```
IMP_STOPPED_OK                        //stop succeeded
IMP_STOPPED_NO                        //failed
ID_WRONG_SEQUENCY_OF_COMMAND          // command was ignored
```

PseudoCode:
```
int ret;
if(ret= StopImpedance())
   //success
else
   //failed
```

Notes:

StopImpedance function is rarely used and is usually called if it appears that the self initiated callback function registered by CheckImpedances has not returned for a long time. A timeout value of 3 minutes is recommended.

The impedance values may be compromised if StopImpedance is called prematurely.

TechnicalMonitoring

Description:
Performs quality check of all EEG channels for a specified time.

Format:
```
int  TechnicalMonitoring ( _stdcall* callback(CHANNEL_INFO*, int&), int
nSeconds, _DEVICE_INFO* pDeviceInfo = NULL);
```

Input Arguments:

1. Type: Pointer to callback function
   callback(CHANNEL_INFO* pCh, int& nCh): TechnicalMonitoring will take several seconds to complete, hence the callback is designed to be non-blocking.
   Parameters of the callback are:
   i) Type: Pointer to CHANNEL_INFO*
   ```
     Typedef struct CHANNEL_INFO {
         char chName[20]; //channel name
         bool bTechnicalInfo; //status (1=signal is good, 0=bad)
     }
   ```
   ii)Type: Address to int
   nCh: Number of channels that was checked

2. Type: Address of int
   nSeconds: duration of quality check in seconds. Minimum time = 15 sec.
3. Type: _DEVICE_INFO pointer
   Obsolete

Output Arguments:

2. Type: int

TM_STARTED_OK	*//Technical monitoring started*
TM_STARTED_NO	*// failed*
ID_WRONG_SEQUENCY_OF_COMMAND	*// Command was ignored*

PseudoCode:
```
static void _stdcall callback(CHANNEL_INFO *pCh, int& nCount) {}
int ret;
if(ret= TechnicalMonitoring(callback,15))
   //success
```

Notes:

TechnicalMonitoring function must be called only after a new session is initiated using InitSession and when data acquisition is not running prallely.

StopTechnicalMonitoring

Description:
Halts technical monitoring.

Format:
```
int  StopTechnicalMonitoring();
```

Input Arguments:
None

Output Arguments:
1. Type: int
```
TM_STOPPED_OK                        //stop succeeded
TM_STOPPED_NO                        //failed
ID_WRONG_SEQUENCY_OF_COMMAND         // command was ignored
```

PseudoCode:
```
int ret;
if(ret= StopTechnicalMonitoring())
   //success
else
   //failed
```

Notes:
StopTechnicalMoniotoring is usually called in cases when the technical moniotoring is non-reponsive after a timeout-duration. The quality report could be compromised if StopTechnicalMonitoring is called prematurely.

StartAcquisition

Description:
Sends START command to ABM device to begin acquisition of data. The SDK will create and start filling the output data files after this function is executed successfully.

Format:
```
Int StartAcquisition();
```

Input Arguments:
None

Output Arguments:

1. Type: int

```
ACQ_STARTED_OK                                    //Acquisition started
ACQ_STARTED_NO                                    //Acquisition failed
ID_WRONG_SEQUENCY_OF_COMMAND  // command was ignored
```

PseudoCode:

```
If (StartAcquisition() == ACQ_STARTED_OK)
  //success
else
  //failed
```

Notes:

PauseAcquisition

Description:
Sends PAUSE command to the ABM device and temporarily stops acquisition of data. The client program should call ResumeAcquisition in order to restart data acquisition.

Format:

```
Int PauseAcquisition();
```

Input Arguments:
None

Output Arguments:

1. Type: int

```
ACQ_PAUSED_OK                                        //Acquisition paused
ACQ_PAUSED_NO                                       //Failed
ID_WRONG_SEQUENCY_OF_COMMAND  // command was ignored
```

PseudoCode:

```
If (PauseAcquisition() == ACQ_PAUSED_OK)
  //success
else
  //failed
```

Notes:

The elapsed time in the data files (Epoch & Offset, computed linearly from the number of data samples obtained from the device) will NOT reflect the pause state. However, the absolute time (Hour, Minute, Second, MilliSecond) will indicate the interruption in data acquisition.

ResumeAcquisition

Description:
Sends RESUME command to ABM device and restarts data acquisition.

Format:
```
Int ResumeAcquisition();
```

Input Arguments:
None

Output Arguments:

1. Type: int
```
   ACQ_RESUMED_OK                                    //Acquisition resumed
   ACQ_RESUMED_NO                                    //Failed
   ID_WRONG_SEQUENCY_OF_COMMAND  // command was ignored
```

PseudoCode:
```
If (ResumeAcquisition() == ACQ_RESUMED_OK)
  //success
else
  //failed
```

Notes:

StopAcquisition

Description:
Sends STOP command to ABM device and stops acquisition of data. This function resets the parameters in the SDK, thus the client program should initiate a new session prior to re-

invoking StartAcquisition after this function is called.

Format:

```
Int StopAcquisition();
```

Input Arguments:

None

Output Arguments:

1. Type: int
   ```
   ACQ_STOPPED_OK                                //Acquisition stopped
   ACQ_STOPPED_NO                                //Failed
   ID_WRONG_SEQUENCY_OF_COMMAND  // command was ignored
   ```

PseudoCode:

```
If (StopAcquisition() == ACQ_STOPPED_OK)
  //success
else
  //failed
```

Notes:

StopAcquisition is usually called at the end of the session. Thus if additional data has to be collected after StopAcquisition is called, a new session has to be initiated. This involves calling SetDestinationFile, SetDefinitionFile(for applicable session types), and InitSession prior to repeating StartAcquisition.

For temporary halts in data acquisition use PauseAcquisition.

GetRawData

Description:

Gets raw data samples from the SDK.

Format:

```
float*  GetRawData ( int& nCount )
```

Input Arguments:

1. Type: Address to int
   nCount: updated with the number of samples returned in the output argument

Output Arguments:

1. Type: float*
   Pointer to array of float values containing raw data samples. The size of the return array = (nChannel+6)*nCount, where, nChannel is the number of channels in the ABM device (see GetPacketChannelNmbInfo), nCount is the number of samples acquired. The number of samples will vary based on the delay between successive calls to the function.

PseudoCode:

```
int nCount;
float *pData;
pData = GetRawData(nCount);
```

Notes:

ABM X10 & X24 devices have a sampling rate of 256, thus new samples are generated every ~4ms. In X10, two samples are combined into a single packet, thus the SDK receives new samples every ~8ms. In X24 a new packet is received every 4ms. Successive calls to GetRawData should have an appropriate intermediate delay, if new data samples are unavailable, NULL value will be returned.

Example structure of the output argument (ABM X10 device):
pData[0]=Epoch, pData[1]=Offset //Epoch & Offset of First sample
pData[2]=Hour, pData[3]=Min, pData[4]=Sec, pData[5]=MiliSec // SDK timestamp
pData[6]=Sample1_Channel1…
pData[15] = Sample1_Channel10
pData[16]=Epoch, pData[17]=Offset //Epoch & Offset of Second sample
pData[18]=Hour, pData[19]=Min, pData[20]=Sec, pData[21]=MiliSec // SDK timestamp
pData[22]=Sample2_Channel1…

X10 Channel mapping: EKG[1], POz[2], Fz[3], Cz[4], C3[5], C4[6], F3[7], F4[8], P3[9], P4[10]

X24 Channel mapping: F3[1], F1[2], Fz[3], F2[4], F4[5], C3[6], C1[7], Cz[8], C2[9], C4[10], CPz[11], P3[12], P1[13], Pz[14], P2[15], P4[16], POz[17], O1[18], Oz[19], O2[20], EKG[21], AUX1[22], AUX2[23], AUX3[24]

GetFilteredData

Description:
Gets filtered physiological data samples (EEG & EKG) from the SDK.

Format:

```
float*  GetFilteredData ( int& nCount )
```

Input Arguments:

1. Type: Address to int
   nCount: updated with the number of epochs returned in the output argument

Output Arguments:

1. Type: float*
   Pointer to array of float values containing filtered data samples. The size of the return array = (nChannel+6)*nCount, where, nChannel is the number of channels in the ABM device (see GetPacketChannelNmbInfo), nCount is the number of samples. The number of samples will vary based on the delay between successive calls to the function.

PseudoCode:

```
int nCount;
float *pData;
pData = GetRawData(nCount);
```

Notes:

Structure of the output argument - Same as GetRawData.

The following filters will be applied on the raw data.

1. 50hz Notch
2. 60hz Notch
3. 100hz Notch
4. 120hz Notch
5. 0.05hz High Pass
6. Median filter (order:256)

GetDeconData

Description:

Gets artifacts decontaminated data samples from the SDK.

Format:

```
float*  GetDeconData ( int& nCount )
```

Input Arguments:

1. Type: Address to int
   nCount: updated with the number of epochs returned in the output argument

Output Arguments:

1. Type: float*
   Pointer to array of float values containing decontaminated data samples. The size of the return array = (nChannel+6)*nCount, where, nChannel is the number of channels in the ABM device (see GetPacketChannelNmbInfo), nCount is the number of samples. The number of samples may vary based on the delay between t successive calls to the function.

PseudoCode:

```
int nCount;
float *pData;
pData = GetDeconData(nCount);
```

Notes:

ABM's proprietary artifact algorithm detects, extracts and compensates 5 types of artifacts: (1) Eye blink, (2) Saturation, (3) Excursion, (4) Spike, (5) EMG. The algorithm is computationally intensive and may lag behind raw data by a few seconds. However, the time-stamps are duplicated from the raw-data and thus marinated accuracetely for analysis.

Strutuctre or output parameter & sampling interval – refer GetRawData

The decontamination algorithms are exlusive to EEG channels, the EKG channel (and AUX channels in X24), even though reported by this function, are not decontaminated.

The detected artifacts are stored in output files for offline refrence and analysis.

Artifacts can be individually informed as-well by registering callback functions in SDK. (Refer SetArtifactsCallbackFuncs).

GetPSDData

Description:
Gets Power Spectral Densities (PSD) of 1Hz to 128Hz from PROCESSED data, for all EEG channels. The raw data is processed by artifact decontamination, smoothening using Kaiser window and averaging across 3-second overlays.

Format:

```
float*  GetPSDData ( int& nEpoch )
```

Input Arguments:

1. Type: Address to int
   nEpoch: updated with the number of epochs returned in the output argument

Output Arguments:

1. Type: float*
   Pointer to array of float values containing PSD values from each channel. The size of the return array = (128*nChannel+5)*nEpoch, where, nChannel is the number of channels in the ABM device (see GetPacketChannelNmbInfo), nEpoch is the number of epochs. The number of epochs will vary based on the delay between successive calls to the function.

PseudoCode:

```
int nEpoch;
float *pData;
pData = GetPSDData(nEpoch);
```

Notes:

Power Spectral Densities are averaged across 3 epoch overlays and reported every epoch (second). Even though the PSDs have an epoch based sampling interval, delays in the artifact detection and decontamination pipeline also effects the reporting of dependant PSD computation. If no data is available, NULL will be returned. If more than a threshold number of data samples are contaminated and cannot be salvaged, the PSDs are reported as -99999 (invalid).

Example structure of the output argument (ABM X10 device):
pData[0]=Epoch, //Epoch 1
pData[1]=Hour, pData[2]=Min, pData[3]=Sec, pData[4]=MilliSec // SDK timestamp
pData[5]=PSDHz1_Epoch1_Channel1,pData[6]=PSDHz2_Epoch1_Channel1, …

PSD is computed only for EEG channels; EKG, AUX channels etc will not be reported by this function.

GetPSDDataraw

Description:

Gets Power Spectral Densities (PSD) of1Hz to 128Hz from RAW data, for all EEG channels.

Format:

```
float*  GetPSDDataraw ( int& nEpoch )
```

Input Arguments:

1. Type: Address to int
   nEpoch: updated with the number of epochs returned in the output argument

Output Arguments:

1. Type: float*
   Pointer to array of float values containing raw PSD values from each channel. The size of the return array = (128*nChannel+5)*nEpoch, where, nChannel is the number of channels in the ABM device (see GetPacketChannelNmbInfo), nEpoch is the number of epochs. The number of epochs may vary with the delay between two calls to the function.

PseudoCode:

```
int nEpoch;
float *pData;
pData = GetPSDDataraw(nEpoch);
```

Notes:

GetPSDDataraw differs from GetPSDData in that - PSD is computed on RAW EEG samples. The artifacts are not decontaminated and Kaiser window smoothening is not applied. Averaging across 3-second overlays is however retained.

Output variable structure & Sampling - refer GetPSDData.

GetBandOverallPSDData

Description:

Gets Power Spectral Densities (PSD) of specified bands averaged across all EEG channels. The PSDs are computed from PROCESSED EEG data. The band PSDs are computed for two sets of channels: Referential & Differential. Differential PSDS are used in the classification algorithms and are computed from referential channels that are digitally subtracted to create differential channels (such as, FzPOz, CzPOz, FzC3, C4C4 and F3Cz).

Format:

```
float*  GetBandOverallPSDData( int& nCountPackages, int& nSize )
```

Input Arguments:

1. Type: Address to int
   nCountPackages: updated with the number of packages returned in the output argument
2. Type: Address to int
   nSize: updated with the number of data bytes in each package

Output Arguments:

1. Type: float*
   Pointer to array of float values containing PSD values of specified bands for the two groups. The size of the return array = (2*nBands+5)*nEpoch, where, nBands is the number of bands, nEpoch is the number of epochs.

PseudoCode:

```
int nEpoch, nSize;
float *pData;
pData = GetBandOverallPSDData(nCountPackages, nSize);
```

Notes:

Sampling is similar to GetPSDData.

The default bands are: (1) Delata 1-2Hz, (2) ThetaSlow 2-5Hz, (3) ThetaFast 5-7Hz, (4) ThetaTotal 3-7Hz, (5) AlphaSlow 8-10Hz, (6) AlphaFast 10-12Hz, (7) AlphaTotal 8-12Hz, (8) Beta 13-29Hz, (9) Sigma 30-40Hz.

Example structure of the output argument (ABM X10 device):
pData[0]=Epoch, //Epoch 1
pData[1]=Hour, pData[2]=Min, pData[3]=Sec, pData[4]=MilliSec // SDK timestamp
pData[5]=Band1_Ref, pData[6]=Band2_Ref… // Raw referential signals
pData[13]=Band1_Diff …   // Raw differential signals

GetBandOverallPSDRawData

Description:
Gets Power Spectral Densities (PSD) of specified bands (derived from RAW EEG data) averaged across all EEG channels.

Format:

```
float*  GetBandOverallPSDRawData( int& nCountPackages, int& nSize )
```

Input Arguments:

1. Type: Address to int
   nCountPackages: updated with the number of epochs returned in the output argument
2. Type: Address to int
   nSize: updated with the number of data bytes in each epoch

Output Arguments:

1. Type: float*
   Pointer to array of float values containing Raw PSD values of specified bands for each of the four groups. The size of the return array = (2*nBands+5)*nEpoch, where, nBands is the number of bands, nEpoch is the number of epochs. The number of epochs may vary with the delay between two calls to the function.

PseudoCode:

```
int nCountPackages, nSize;
float *pData;
pData = GetBandOverallPSDRawData(nCountPackages, nSize);
```

Notes:

For details such as output variable structure, sampling, band definitions – refer GetBandOverallPSDData

GetPSDBandwidthData

Description:

Gets Power Spectral Densities (PSD) of specified bands derived from PROCESSED EEG data for each EEG channel individually. Bands are reported for two groups: Referential & Differential, where differential signals are derived by digitally subtracting the referential signals.

Format:

```
float*  GetPSDBandwidthData( int& nCountPackages, int& nSize )
```

Input Arguments:

1. Type: Address to int
   nCountPackages: updated with the number of epochs returned in the output argument
2. Type: Address to int
   nSize: updated with the number of data bytes in each epoch

Output Arguments:

1. Type: float*
   Pointer to array of float values containing PSD values of specified bands for each channel. The size of the return array = nSize*nEpoch, where, nSize is the number bytes returned per epoch and nEpoch is the number of epochs. The number of epochs may vary with the delay between two calls to the function.

PseudoCode:

```
int nCountPackages, nSize;
float *pData;
pData = GetPSDBandwidthData(nCountPackages, nSize);
```

Notes:

Sampling, band definitions, etc are similar to GetBandOverallPSDData.

Example structure of the output argument (ABM X10 device, 9 channels of EEG):
pData[0]=Epoch, //Epoch 1
pData[1]=Hour, pData[2]=Min, pData[3]=Sec, pData[4]=MiliSec // SDK timestamp
pData[5]=Band1_Refchannel1, pData[6]=Band2_Refchannel1… //referential channels

GetPSDBandwidthRawData

Description:
Gets Power Spectral Densities (PSD) of specified bands derived from RAW EEG data for each EEG channel individually. Bands are reported for two groups: Referential & Differential, where differential signals are derived by digitally subtracting the referential signals.

Format:

```
float*  GetPSDBandwidthRawData( int& nCountPackages, int& nSize )
```

Input Arguments:

1. Type: Address to int
   nCountPackages: updated with the number of epochs returned in the output argument
2. Type: Address to int
   nSize: updated with the number of data bytes in each epoch

Output Arguments:

1. Type: float*
   Pointer to array of float values containing PSD values of specified bands for each channel. The size of the return array = nSize*nEpoch, where, nSize is the number bytes returned per epoch and nEpoch is the number of epochs. The number of epochs may vary with the delay between two calls to the function.

PseudoCode:

```
int nCountPackages, nSize;
float *pData;
pData = GetPSDBandwidthRawData(nCountPackages, nSize);
```

Notes:

Output variable, sampling, band definitions, etc are similar to GetPSDBandwidthData.

GetBrainState

Description:
Gets B-Alert Engagement and Workload classification indices.

Format:

```
_BRAIN_STATE*  GetBrainState( int& nEpoch )
```

Input Arguments:

1. Type: Address to int
   nEpoch: updated with the number of epochs returned in the output argument

Output Arguments:

1. Type: Pointer to _BRAIN_STATE

```
Typedef struct _BRAIN_STATE {
    float fEpoch;
    float fABMSDKTimeStampHour;
    float fABMSDKTimeStampMinute;
    float fABMSDKTimeStampSecond;
    float fABMSDKTimeStampMilsecond;
    float fClassificationEstimate;
    float fHighEngagementEstimate;
    float fLowEngagementEstimate;
    float fDistractionEstimate;
    float fDrowsyEstimate;
    float fWorkloadFBDS;
    float fWorkloadBDS;
    float fWorkloadAverage;
}
```

PseudoCode:

```
int nEpoch;
_BRAIN_STATE* pBrainState = NULL;
pBrainState = GetBrainState(nEpoch);
```

Notes:

Classification algorithms are at the final stages of the computational pipeline and thus they suffer from the acculmulated delays in artifact decontamination and PSD computation. Even though the output variable can be sampled every second, in some cases, during excessive noise in the input signal, classifications ouputs can be delayed. When no data is avialble due to delays, NULL will be reported and in cases where the EEG samples are contaminated and cannot be salvaged, -99999 (invalid) is reported.

InitZScoreData

Description:
Initializes parameters for requesting z-scores.

Format:

```
int  InitZScoreData( char* sZscoreSourceList);
```

Input Arguments:

1. Type: Pointer to string
   sZscoreSourceList: string with symbolic names of z-score data sources separated by commas: Classifications (class_higheng, class_loweng, class_distr, class_sleeponset, wl_ave), Heart rate (HeartRate), PSD values (OvRaw<Ref|Diff>_<band_index>, for example OvRawRef_0 = overall raw PSD from all referential channels in band 0. Band index is the number followed in PSDSettings.xml), Bandwidth PSD values (<channel_name>_<band_index>, for example, po_3 = bandwidth raw PSD from all referential channels in band 3 in channel PO.

Output Arguments:

1. Type: int
   ZS_LIB_SUCCESS //ZScore initialization succeeded
   ZS_LIB_FAILED //failed

PseudoCode:

```
int ret;
if(ret = InitZScoreData( “” ))
   //success
else
  //failed
```

Notes:

InitZScoreData must be called before startacquisition function.

Example initialization string: “class_higheng,class_loweng,class_distr, class_sleeponset,wl_ave,HeartRate,OvRawRef_0,OvRawDif_1,po_3,fzpo_6,c3_7"

GetZScoreData

Description:
Gets computed Z-score values.

Format:

```
float* GetZScoreData(int& nCountPackages, int& nSize);
```

Input Arguments:

1. Type: Address to int
   nCountPackages: updated with the number of epochs returned in the output argument
2. Type: Address to int
   nSize: updated with the number of data bytes in each epoch

Output Arguments:

1. Type: float*
   Pointer to array of float values containing Z-Score values of sources specified using InitZScoreData function. bands for each channel. The size of the return array = nSize*nEpoch, where, nSize is the number bytes returned per epoch and nEpoch is the number of epochs. The number of epochs may vary with the delay between two calls to the function.

PseudoCode:

```
int nCountPackages, nSize;
float *pData;
pData = GetZScoreData(nCountPackages, nSize);
```

Notes:

InitZScore function must be called prior to calling GetZScoreData.

Sampling interval of the output variable – per epoch. In case no data is available (due to delays), NULL will be reported.

ResetZScoreData

Description:
Resets parameters for requesting z-scores

Format:
```
int  ResetZScoreData( char* sNewListOfSources);
```

Input Arguments:

1. Type: Pointer to string
   sNewListOfSources: same format as InitZScoreData function input argument.

Output Arguments:

1. Type: int
   ZS_RESET_SUCCESS //ZScore reset succeeded
   ZS_RESET_FAILED   //failed

PseudoCode:
```
int ret;
if(ret = ResetZScoreData( “” ))
   //success
else
  //failed
```

Notes:

GetQualityChannelData

Description:
Gets % of 1) overall data quality across all channels combined, 2) EMG & Other artifacts in each individual channel.

Format:

```
float*  GetQualityChannelData ( int& nCount )
```

Input Arguments:

1. Type: Address to int
   nCount: updated with the number of epochs returned in the output argument

Output Arguments:

1. Type: float*
   Pointer to array of float values containing overall data quality and the % of EMG and artifacts in each individual channel. The size of the return array = (2*nChannel+6+1)*nEpoch, where, nChannel is the number of EEG channels in the device (see GetPacketChannelNmbInfo), nEpoch is the number of epochs. The number of epochs may vary based on the delay between two calls to the function.

PseudoCode:

```
int nCount;
float *pData;
pData = GetQualityChannelData(nCount);
```

Notes:

Sampling interval of the output variable – per epoch. If no data is available, NULL will be reported.

Example structure of the output argument (ABM X10 device):
pData[0]=Epoch, pData[1]=Offset //Epoch & Offset of First sample
pData[2]=Hour, pData[3]=Min, pData[4]=Sec, pData[5]=MiliSec // SDK timestamp
pData[6]=Overall quality across channels (% of good data),
pData[7]= % of data contaminated with EMG in EEG channel # 1
pData[8]= % of data contaminated with other artifacts in EEG channel # 1
pData[9]= % of data contaminated with EMG in EEG channel # 2
pData[10]= % of data contaminated with other artifacts in EEG channel # 2
….

GetMovementData

Description:
Gets movement-value and movement-level computed from the the 3-axis accelerometer data transmitted from ABM devices.

Format:

```
float*  GetMovementData ( int& nEpoch )
```

Input Arguments:

1. Type: Address to int
   nEpoch: updated with the number of epochs returned in the output argument

Output Arguments:

1. Type: float*
   Pointer to array of float values containing movement-value and movement-level in each epoch. The size of the return array = (2+5)*nEpoch, where nEpoch is the number of epochs. The number of epochs may vary with the delay between two calls to the function.

PseudoCode:

```
int nEpoch;
float *pData;
pData = GetMovementData(nEpoch);
```

Notes:

Movement value is reported as the sum of the change in the two dominanant movement angles derived from raw 3-axis accelerometer tilt data. Movement scale (reported as a value between 0 and 5) is computed from the change in movement angles, based on a proprietory ABM algorithm.

Sampling interval of the output variable – per epoch.

Example structure of the output argument (ABM X10 device):
pData[0]=Epoch, //epoch 1
pData[1]=Hour, pData[2]=Min, pData[3]=Sec, pData[4]=MiliSec // SDK timestamp
pData[5]=Mvmt_value, //movement value
pData[6]=Mvmt_level,   // movement level

GetTiltRawData

Description:
Gets raw x,y,z tilt values from the 3-axis accelerometer in ABM devices.

Format:

```
float*  GetTiltRawData ( int& nCount)
```

Input Arguments:

1. Type: Address to int

nCount: updated with the number of samples

Output Arguments:

1. Type: float*
   Pointer to array of float values containing raw x,y,z tilt values. The size of the return array = (6+3)*nCount, where nCount is the number of samples. The number of samples may vary based on the delay between two calls to the function.

PseudoCode:

Notes:

The accelerometer data has a sampling rate of 128hz (X4,X10) and 256hz (X24). If no data is available, NULL will be returned.

Example structure of the output argument (ABM X10 device):
pData[0]=Epoch, pData[1]=Offset
pData[2]=Hour, pData[3]=Min, pData[4]=Sec, pData[5]=MiliSec // SDK timestamp
pData[6]=Tilt Raw X, pData[7]=Tilt Raw Y, pData[8]=Tilt Raw Z

GetTiltAnglesData

Description:
Gets x,y,z angles from the 3-axis accelerometer in ABM devices.

Format:
```
float*  GetTiltAnglesData ( int& nCount )
```

Input Arguments:

1. Type: Address to int
   nCount: updated with the number of samples

Output Arguments:

1. Type: float*
   Pointer to array of float values containing raw x,y,z angles. The size of the return array = (6+3)*nCount, where nCount is the number of samples. The number of samples may vary based on the delay between two calls to the function.

PseudoCode:

Notes:

The accelerometer data has a sampling rate of 128hz (X4,X10) and 256hz (X24). If no data is available, NULL will be returned.

Example structure of the output argument (ABM X10 device):
pData[0]=Epoch, pData[1]=Offset
pData[2]=Hour, pData[3]=Min, pData[4]=Sec, pData[5]=MiliSec // SDK timestamp
pData[6]= X Angle, pData[7]=Y Angle, pData[8]=Z Angle

GetEKGData

Description:
Gets heart rate values computed from EKG.

Format:
```
float*  GetEKGData ( int& nEpoch )
```

Input Arguments:
1. Type: Address to int
   nEpoch: updated with the number of epochs returned in the output argument

Output Arguments:
1. Type: float*
   Pointer to array of float values containing heart rate values. The size of the return array = (4+6)*nEpoch, where nEpoch is the number of epochs. The number of epochs may vary with the delay between two calls to the function.

PseudoCode:
```
int nEpoch;
float *pData;
pData = GetEKGData(nEpoch);
```

Notes:

Sampling interval of output variable – per epoch. If no data is available, NULL will be reported.

Example structure of the output argument (ABM X10 device):
pData[0]=Epoch, pData[1]=Offset //Epoch1 & Offset1
pData[2]=Hour, pData[3]=Min, pData[4]=Sec, pData[5]=MiliSec // SDK timestamp
pData[6]=Heart Rate,
pData[7]=Inter beat interval,
pData[8]=beat quality, //good=1, otherwise=0
pData[9]=packet type, // (0=sec by sec value, 1=beat by beat value, 2=Reserved, 3=GUI presentation value)

The value presented on the GUI is checked continuously for valid EKG. In case valid EKG was not detected, the output (display) is initialized to min value. Sec by sec & beat by beat value produce outputs continuously but the beat quality is marked.

GetThirdPartyData

Description:
Returns third party data acquired using ABM Multi-Channel External Sync Unit (MC-ESU).

Format:
```
unsigned char*  GetThirdPartyData(int& nSize);
```

Input Arguments:
1. Type: Address to int
   nSize: updated with the number of bytes obtained

Output Arguments:
1. Type: Pointer to unsigned char*
   Array of bytes containing third party packets (packet number = nSize). Each packet has the following format:

Flag (0x56, 0x56)	– 2 bytes
Message Counter	– 1 byte
ESU Timestamp	- 4 bytes
Packet Length	- 2 bytes
Packet Type	- 1 byte
Third Party Data	- bytes = Packet Length
Checksum	- 1 byte

PseudoCode:

```
int nSize;
unsigned char *pucTPData;
pucTPData = GetThirdPartyData(nSize);
```

Notes:

Please refer section ABM MC-ESU Programming Interface for details regarding configuring MC-ESU and sending/decoding third party data.

If no third party data is available, NULL will be returned.

GetTimeStampsStreamData

Description:
Returns timestamps for raw and processed data samples.

Format:

```
unsigned char*  GetTimeStampsStreamData(int nType);
```

Input Arguments:

1. Type: Address to int
   nType: Timestamps of the following data are supported
   TIMESTAMP_RAW (0)
   TIMESTAMP_PSD (1)
   TIMESTAMP_DECON (2)
   TIMESTAMP_CLASS(3)
   TIMESTAMP_EKG (4)

Output Arguments:

1. Type: Pointer to unsigned char*
   Array of bytes containing 4-byte timestamps each (high byte first). Note that the array may have multiple timestamps, depending on the interval between two successive calls.

PseudoCode:

```
int nType;
unsigned char *pucTSData;
pucTSData = GetTimeStampsStreamData(nType);
```

Notes:

This function must be called immediately after the data acquisition function. For example, if GetRawData returns 2 epochs of data, then GetTimeStampsStreamData when called immediately after GetRawData (with Timestamp Type = TIMESTAMP_RAW), will return two 4-bytes timestamps corresponding to the 2 epochs.

ABM devices connect to two types of ABM receivers: 1) ABM Dongle, 2)Multi-channel ESU (MC-ESU). When using ABM dongle, windows time (8-bytes format) will be returned. When using MC-ESU, 4-bytes timestamp generated in the MC-ESU (using a dedicated hardware timer) will be returned.

GetCurrentSDKMode

Description:
Returns the current operating mode of SDK

Format:
```
int  GetCurrentSDKMode();
```

Input Arguments:

None.

Output Arguments:

1. Type: int
   Integer value representing working modes of the SDK:
   SDK_WAITING_MODE (-1)
   SDK_NORMAL_MODE (0)
   SDK_IMPEDANCE_MODE (1)
   SDK_TECHNICALMON_MODE (2)

PseudoCode:
```
int nMode;
nMode = GetCurrentSDKMode();
```

Notes:

ShowChannel

Description:
Invokes a single channel graphical plot of the input data.

Format:
```
int  ShowChannel(int nChannel, DWORD rgbBackground, DWORD rgbSignal, DWORD
rbgTitle, DWORD rgbAxis, DWORD rgbGrid, DWORD rgbAxisValues, float fGain, int
nTimeScale);
```

Input Arguments:
1. Type: int
   nChannel: index of channel (0 based)
2. Type: DWORD
   rgbBackground: background color in rgb format
3. Type: DWORD
   rgbSignal: signal color in rgb format
4. Type: DWORD
   rgbTitle: title color in rgb format
5. Type: DWORD
   rgbAxis: axis color in rgb format
6. Type: DWORD
   rgbGrid: grid lines color in rgb format
7. Type: DWORD
   rgbAxisValues: axis values color in rgb format
8. Type: float
   fGain: display gain (Values allowed: 1 <= fGain <= 20)
9. Type: int
   nTimeScale: display time scale (Values allowed: 1 <= nTimeScale <= 60)

Output Arguments:
1. Type: int
   1 = success, 0 = failed

PseudoCode:
```
int ret;
if(ret = ShowChannel(0,0,65535,16777215,16777215,789516,16777215,1,10))
   //success
```

Notes:

It is recommended to pause few seconds between successive invocations of this function.

AlignChannels

Description:
Aligns the single-channel graphical plot windows generated by ShowChannel.

Format:
```
int  AlignChannels();
```

Input Arguments:

None.

Output Arguments:

1. Type: int
   1 = success, 0 = failed

PseudoCode:
```
int ret
if(ret = AlignChannels());
   //success
else
   //failed
```

Notes:

SetArtifactsCallbackFuncs

Description:
Registers callback function to inform artifacts. Five types of artifacts are reported - Eyeblink, Excursion, Saturation, Spike, EMG.

Format:
```
int  SetArtifactsCallbackFuncs ( _stdcall* callbackEB, _stdcall* callbackExc,
_stdcall* callbackSat, _stdcall* callbackSpk, _stdcall* callbackEMG);
```

Input Arguments:

1. Type: Pointer to Eye blink artifact callback function
   callbackEB(int& epSt, int& offSt, float& Sthr, float& Stmin, float& Stsec, float& Stmsec, float& Enhr, float& Enmin, float& Ensec, float& Enmsec)
2. Type: Pointer to Excursion artifact callback function
   callbackExc(int& ind, int& epSt, int& offSt, float& Sthr, float& Stmin, float& Stsec, float& Stmsec, float& Enhr, float& Enmin, float& Ensec, float& Enmsec)
3. Type: Pointer to Saturation artifact callback function
   callbackSat(int& ind, int& epSt, int& offSt, float& Sthr, float& Stmin, float& Stsec, float& Stmsec, float& Enhr, float& Enmin, float& Ensec, float& Enmsec)
4. Type: Pointer to Spike artifact callback function
   callbackSpk(int& ind, int& epSt, int& offSt, float& Sthr, float& Stmin, float& Stsec, float& Stmsec, float& Enhr, float& Enmin, float& Ensec, float& Enmsec)
5. Type: Pointer to EMG artifact callback function
   callbackEMG(int& ind, int& epSt, int& offSt, float& Sthr, float& Stmin, float& Stsec, float& Stmsec, float& Enhr, float& Enmin, float& Ensec, float& Enmsec)

   Parameters of the callback are: 1) ind: index of the channel, 2)epSt: start epoch, 3)offSt: start offset, 4)Sthr: start hour, 5)Stmin: start min, 6)Stsec: start sec, 7)Stmsec: start millisec, 8)Enhr: End hour, 9)Enmin: End minute, 10)Ensec: End second, 11)Enmsec: End millisecond.

Output Arguments:

1. Type: int
   1 = success, 0 = failed

PseudoCode:

```
if(ret= SetArtifactsCallbackFuncs(callbackEB, callbackExc, callbackSat, callbackSpk, callbackEMG))
   //success
```

Notes:

RegisterCallbackImpedanceElectrodeFinished

Description:

Registers callback function to return impedance values of each channel individually. The channels are reported with channel number.

Format:

```
int  RegisterCallbackImpedanceElectrodeFinished ( _stdcall* callback(int , float));
```

Input Arguments:

1. Type: Pointer to callback function
   callback(int nCh, float value)
   Parameters of the callback are:
   i) Type: int
   nCh: Channel index or number
   ii)Type: float
   value: Impedance value

Output Arguments:

2. Type: int
   1 = success, 0 = failed

PseudoCode:

```
static void _stdcall callback(int& nCh, float& value) {}
int ret;
if(ret= RegisterCallbackImpedanceElectrodeFinished(callback))
   //success
```

Notes:

Impedance computation takes approx 5sec/channel.

The channel index is reported as channel numbers. For example, reference channel = 0, channel 1 = 1, etc.

RegisterCallbackImpedanceElectrodeFinishedA

Description:
Registers callback function to return impedance values of each channel individually. The channels are reported with channel name.

Format:

```
int  RegisterCallbackImpedanceElectrodeFinished ( _stdcall* callback(char* , float));
```

Input Arguments:

1. Type: Pointer to callback function
   callback(char* aCh, float value)
   Parameters of the callback are:
   i) Type: Pointer to char

aCh: Channel name
ii)Type: float
value: Impedance value

Output Arguments:

1. Type: int
   1 = success, 0 = failed

PseudoCode:

Notes:

RegisterCallbackDataArrived

Description:
Registers callback function to return sample number when a new sample is acquired from the device.

Format:
```
int  RegisterCallbackDataArrived ( _stdcall* callback(int));
```

Input Arguments:

1. Type: Pointer to callback function
   callback(int nSample): Parameters of the callback are:
   i) Type: int
   nSample: sample number

Output Arguments:

1. Type: int
   1 = success, 0 = failed

PseudoCode:
```
static void _stdcall callback(int nSample) {}
int ret;
if(ret= RegisterCallbackDataArrived (callback))
   //success
```

Notes:

The data arrived callback is called for every sample and is usually used for synchronization purposes. For example it can be used to send the elapsed time (epoch and offset) and clock time (hour, min, sec, miilisec) to a third party application.

RegisterCallbackOnStatusInfo

Description:
Registers callback to return information such as: battery voltage, timestamp, missed block count, current SDK mode, and last error code. The callback is invoked at the rate of 2Hz.

Format:
```
int  RegisterCallbackOnStatusInfo ( _stdcall* callback(_STATUS_INFO*));
```

Input Arguments:

1. Type: Pointer to callback function
   callback(_STATUS_INFO*):
   ```
   typedef struct _STATUS_INFO {
     float BatteryVoltage;
     int BatteryPercentage;
     int Timestamp;
     int TotalMissedBlocks;
     int ABMSDK_Mode;
     int LastErrorCode;
     int CustomMarkA;
     int CustomMarkB;
     int CustomMarkC;
     int CustomMarkD;
      int nTotalSamplesReceived;
   }
   ```

Output Arguments:

1. Type: int
   1 = success, 0 = failed

PseudoCode:

```
static void _stdcall callback(_STATUS_INFO *sInfo) {}
int ret;
if(ret= RegisterCallbackOnStatusInfo (callback))
   //success
```

Notes:

RegisterCallbackOnError

Description:
Registers callback that returns internally generated error codes in the SDK.

Format:

```
int  RegisterCallbackOnError ( _stdcall* callback(int));
```

Input Arguments:

1. Type: Pointer to callback function
   callback(int nErrorCode):
   i) Type: Address to int
   nErrorCode: Error code

Output Arguments:

1. Type: int
   1 = success, 0 = failed

PseudoCode:

```
static void _stdcall callback(int nErrorCode) {}
int ret;
if(ret= RegisterCallbackOnError (callback))
   //success
```

Notes:

Refer section - SDK Error / Warning Codes.

RegisterCallbackMissedBlocks

Description:
Registers callback that returns Start-epoch and End-epoch of missed data blocks.

Format:
```
int  RegisterCallbackMissedBlocks ( _stdcall* callback(int, int));
```

Input Arguments:

1. Type: Pointer to callback function
   callback(int nStart, int nEnd):
   i) Type: int
   nStart: Start epoch of missed block
   nEnd : End epoch of missed block

Output Arguments:

1. Type: int
   1 = success, 0 = failed

PseudoCode:
```
static void _stdcall callback(int nStart, int nEnd) {}
int ret;
if(ret= RegisterCallbackMissedBlocks (callback))
   //success
```

Notes:

GetClassMu

Description:

Gets Power Spectral Densities (PSD) for alpha band averaged across EEG channels: C1*, C2*, C3, C4, Cz (* if channels exist in selected device configuration).
The PSDs are computed from PROCESSED EEG data.

Format:

```
float* GetClassMu(int& nEpochs)
```

Input Arguments:

1. Type: Address to int
   nEpoch: updated with the number of epochs returned in the output argument

Output Arguments:

2. Type: float *
   Pointer to array of float values containing PSD values alpha bands averaged across specific channels.
   The size of the return array = 3*nEpoch.

   3: AlphaSlow, AlphaFast, AlphaTotal

PseudoCode:

```
int nCount;
float *pData;
pData = GetClassMu(nCount);

pData[0]  - 1 epoch SlowAlpha
pData[1]  - 1 epoch FastAlpha
pData[2]  - 1 epoch TotalAlpha
pData[3]  - 2 epoch SlowAlpha
pData[4]  - 2 epoch FastAlpha
pData[5]  - 2 epoch TotalAlpha
…
```

Notes:

GetClassMidline

Description:

Gets Power Spectral Densities (PSD) for theta band averaged across EEG channels:
x10 = Fz,Cz,Poz, X24-VERP=Fz,Cz,Poz,Pz,Oz,Cpz, X24QEEG=Fz,Cz,Pz,POz

(* if channels exist in selected device configuration).

The PSDs are computed from PROCESSED EEG data.

Format:

```
float* GetClassMidline (int& nEpochs)
```

Input Arguments:

2. Type: Address to int
   nEpoch: updated with the number of epochs returned in the output argument

Output Arguments:

3. Type: float *
   Pointer to array of float values containing PSD values alpha bands averaged across specific channels.
   The size of the return array = 3*nEpoch.

   3: ThetaSlow, ThetaFast, ThetaTotal

PseudoCode:

```
int nCount;
float *pData;
pData = GetClassMidline (nCount);

pData[0]  - 1 epoch SlowTheta
pData[1]  - 1 epoch FastTheta
pData[2]  - 1 epoch TotalTheta
pData[3]  - 2 epoch SlowTheta
pData[4]  - 2 epoch FastTheta
pData[5]  - 2 epoch TotalTheta
…
```

Notes:

GetClassPrefrontal

Description:

Gets Power Spectral Densities (PSD) for gamma band averaged across EEG channels: FP1, FP2

(* if channels exist in selected device configuration).

The PSDs are computed from PROCESSED EEG data.

Format:

```
float* GetClassPrefrontal (int& nEpochs)
```

Input Arguments:

3. Type: Address to int
   nEpoch: updated with the number of epochs returned in the output argument

Output Arguments:

4. Type: float *
   Pointer to array of float values containing PSD values alpha bands averaged across specific channels.
   The size of the return array = 1*nEpoch.

   1: Gamma

PseudoCode:

```
int nCount;
float *pData;
pData = GetClassMidline (nCount);

pData[0]  - 1 epoch Gamma
pData[1]  - 2 epoch Gamma
…
```

Notes:

2.4. SDK Error / Warning Codes

2.4.1 Errors

ABM_ERROR_SDK_ACQUISITION_STOPPED	100
ABM_ERROR_SDK_NO_DATA_ARRIVING	101
ABM_ERROR_SDK_CREATE_MAIN_WINDOW_FAILED	110
ABM_ERROR_SDK_COULDNT_FIND_DEVICE	120
ABM_ERROR_SDK_COULDNT_CONNECT_DEVICE	121
ABM_ERROR_SDK_EDF_FILE_ERROR	130
ABM_ERROR_SDK_COULDNT_START_REALTIME	150
ABM_ERROR_SDK_COULDNT_START_SAVING	151
ABM_ERROR_SDK_COULDNT_STOP_REAL_TIME	170
ABM_ERROR_SDK_COULDNT_LOAD_CHANNEL_MAP	180
ABM_ERROR_SDK_WRONG_SESSION_TYPE	190
ABM_ERROR_SDK_WRONG_INPUT_SETTINGS	191
ABM_ERROR_SDK_WRONG_FILE_PATHS	192
ABM_ERROR_SDK_CLASSIFICATION_INIT_FAILED	193
ABM_ERROR_SDK_COULDNT_CLOSE_CONNECTION	194
ABM_ERROR_SDK_NOTSET_DEFFILE	195
ABM_ERROR_SDK_WRONG_DESTPATH	196
ABM_ERROR_SDK_TOO_LARGE_MISSED_BLOCK	197
ABM_ERROR_SDK_TOO_MANY_MISSED_BLOCKS	198
ABM_ERROR_SDK_COMMAND_IMP_START_FAILED	160
ABM_ERROR_SDK_COMMAND_IMP_STOP_FAILED	161
ABM_ERROR_SDK_COMMAND_IMP_HIGH_FAILED	162
ABM_ERROR_SDK_COMMAND_IMP_LOW_FAILED	163
ABM_ERROR_SDK_COMMAND_IMP_START_FAILED_IGNORED	164
ABM_ERROR_SDK_INVALID_ESU_CONFIG_MULTIPLE_PORTS	140
ABM_ERROR_SDK_INVALID_ESU_CONFIG_NO_PORTS	141
ABM_ERROR_SDK_INVALID_ESU_CONFIG_COM1	142
ABM_ERROR_SDK_INVALID_ESU_CONFIG_COM2	143
ABM_ERROR_SDK_INVALID_ESU_CONFIG_COM3	144
ABM_ERROR_SDK_INVALID_ESU_CONFIG_COM4	145

ABM_ERROR_SDK_INVALID_ESU_CONFIG_LPT	146

2.4.2 Warnings

ABM_WARN_SDK_ACQUISITION_STOPPED	200
ABM_WARN_SDK_COULDNT_CONNECT	230
ABM_WARN_SDK_DEVICE_NOT_FOUND	250
ABM_WARN_SDK_TIMESTAMP_WO_SAMPLE	260
ABM_WARN_SDK_TIMESTAMP_NOT_ARRIVED	270

3. B-Alert Matlab® Programming Interface

3. B-Alert Matlab® Programming Interface

3.1. Matlab Interface Architecture 69

3.2. Setup Instructions 69

3.3. Application Programming Interface 70

GetDeviceInfo 70

SetDestinationFile 71

SetDefinitionFile 72

InitSession 72

StartAcquisition 73

PauseAcquisition 74

ResumeAcquisition 75

StopAcquisition 75

GetRawData 76

GetFilteredData 77

GetDeconData 77

GetPSDData 78

GetPSDDataraw 79

GetBandOverallPSDData 79

GetBandOverallPSDRawData 80

GetPSDBandwidthData 81

GetPSDBandwidthRawData 81

GetBrainState 82

InitZScoreData 83

GetZScoreData 83
ResetZScoreData 84
GetQualityChannelData 85
GetMovementData 85
GetEKGData 86
CheckImpedances 87
TechnicalMonitoring 87
GetThirdPartyData 88
GetTimeStampsStreamData 89
GetCurrentSDKMode 89
ShowChannel 90
AlignChannels 90
SetArtifactsCallbackFuncs 91
RegisterCallbackImpedanceElectrodeFinished 91
RegisterCallbackDataArrived 92
RegisterCallbackOnStatusInfo 93
RegisterCallbackOnError 93
RegisterCallbackMissedBlocks 94

3.1. Matlab Interface Architecture

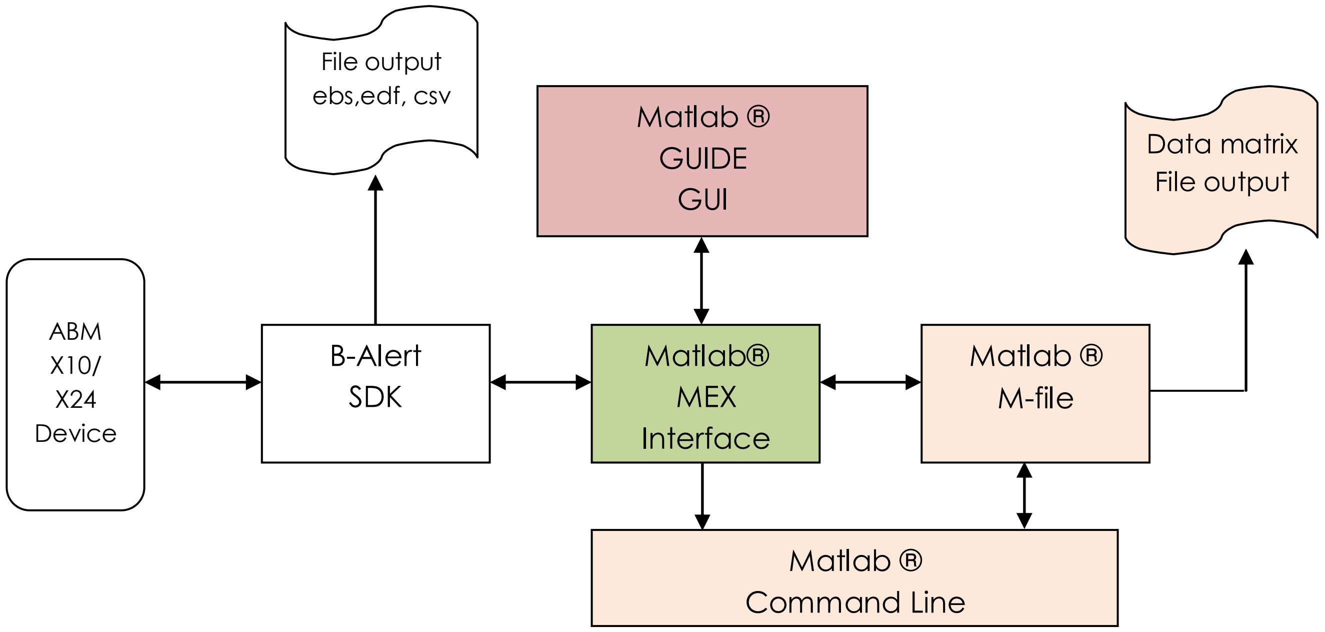


Figure 8: Matlab interface architecture

Third party applications can interact with ABM SDK in MATLAB environment via B-Alert Matlab Mex interface. The interface is developed in c++ and compiled to a mex32 file using standard Matlab compiler. The third party application can interact with the mex interface either through scripts (m-files) or through a GUI (developed using Matlab GUIDE). Sample scripts files as-well-as a basic GUI is also provided for reference.

3.2. Setup Instructions

1. Install ABM B-Alert software. The following files can be found in ABM\EEG\SDK\SampleClients\TestSDKMatlab folder:
    a. abmsdk.cpp – source code of mex interface files
    b. abmsdk.mexw32 – compile mex-file (R2008b)
    c. Test files

2. Compile abmsdk.cpp file (if recquired). In order to avoid incompatibility with different Matlab versions, it is highly recommended that the c++ file be recompiled and a new abmsdk.mexwXX created. The mex file can be compiled using standard Matlab C++ compiler by entering *mex abmsdk.cpp* at the command line.

3. Refer sample scripts and/or GUI provided in the installation in order to interact with the mex interface.

Note: B-Alert Pro dlls are compatible only for 32-bit architecture.

3.3. Application Programming Interface

GetDeviceInfo

Description:
Checks whether ABM device is properly connected to ABM receiver. Returns information about the connected device.

Format:

```
 [devChN,devComPort,devNumCh,devEKGPos, devESUtp,devTStp] =
abmsdk('GetDeviceInfo');
```

Input Arguments:

None

Output Arguments:

1. Type: char[256]
   devChN – Channel Names
2. Type: unsigned char
   devComPort – Com port number to which the device is connected
3. Type: unsigned char
   devNumCh – Num of channels (for example x10 =10)
4. Type: unsigned char
   devEKGPos – EKG channel number (X10 default=1, X24 default=21)
5. Type: unsigned char
   DevESUtp – ABM External Sync Unit (ESU) type (Single-channel=1/Multi-channel=0)
6. Type: unsigned char
   devTStp – Type of timestamp (Internal System time=1/ESU time=0)

Notes:

This function checks connectivity to the device and should be called as the first step in the initialization process.

Compatibility: X10,X24

SetDestinationFile

Description:
Informs SDK the path and name of the output file(s).

Format:
```
[status] = abmsdk('SetDestinationFile',destinationFilepath);
```

Input Arguments:

1. Type: string
   destinationFilepath – Absolute path to destination folder and filename.

Output Arguments:

1. SUCCESS (1) / FAILED (0)

Notes:

The ebs files are usually named based on session #(4-digit), group #(1-digit), iteration # (1-digit), task # (2-digit), task iteration #(1-digit).

Example: ‘C:₩₩ABM₩₩EEG₩₩SDK₩₩TestSDKCpp₩₩Output Files₩₩123456789.ebs’

The SDK also creates many comma separated variable (csv) files as-well-as binary (bin) files in the same path. These files are user selectable through parameters in AthenaSDK.xml file.

For SDK versions Beta v2.30 and later, SetDestinationFile API is optional. If the destination path is not set, none of the output files will be created.

Compatibility: X10,X24

SetDefinitionFile

Description:
Informs SDK the path and name of the definition file.

Format:
```
[status] = abmsdk('SetDefinitionFile',definitionFilepath);
```

Input Arguments:

1. Type: string
   destinationFilepath – Absolute path to the definition file

Output Arguments:

1. SUCCESS (1) / FAILED (0)

Notes:

The Definition file contains individualized information on each subject and is necessary for the computation of B-Alert classifications (Session Types: ABM_SESSION_BSTATE & ABM_SESSION_WORKLOAD). It is very important to use correct definition file when performing classifications – otherwise results will not be valid.

The Definition file is created using baseline sessions. Graphical interface to create the definition file is not available through SDK but can be invoked using B-Alert software (recommended) or through a utility called BAlert-Batch invoked through command line.

The definition file is mandatory for Session Types: ABM_SESSION_BSTATE & ABM_SESSION_WORKLOAD

Compatibility: X10

InitSession

Description:
Initializes SDK to begin a NEW session.

Format:

```
[status] = abmsdk('InitSession',deviceType,sessionType,deviceHandle,bplayEBS);
```

Input Arguments:

2. Type: int
   deviceType : informs SDK which device configuration to use
   Values:
   ```
   ABM_DEVICE_X10Standard=2
   ABM_DEVICE_X24Standard=1
   ABM_DEVICE_X24QEEG=0
   ```

3. Type: int
   sessionType: informs SDK which session configuration to use
   Values:
   ```
   ABM_SESSION_RAW =0          //For RAW & RAW-PSD data
   ABM_SESSION_DECON=1         //Additional DECON, DECON-PSD data
   ABM_SESSION_BSTATE=2        //Additional Brain State
   Classification data
   ABM_SESSION_WORKLOAD=3  //Additional B-Alert Workload data
   ```
4. Type: int
   deviceHandle = -1 //Reserved
5. Type: bool
   bplayEBS = 0 //Reserved

Output Arguments:

1. SUCCESS (1) / FAILED (0)

Notes:

When using session type = ABM_SESSION_BSTATE or ABM_SESSION_WORKLOAD, definition file should be set by calling SetDefinitionFile before calling InitSession.

Only session type = ABM_SESSION_RAW is supported for X24

Compatibility: X10, X24

StartAcquisition

Description:
Sends START command to ABM device and begins acquisition of data. The SDK will create and start filling the output data files after this function is successfully executed.

Format:
```
[status] = abmsdk('StartAcquisition');
```

Input Arguments:

None

Output Arguments:

1. SUCCESS (1) / FAILED (0)

Notes:

Compatibility: X10, X24

PauseAcquisition

Description:

Sends PAUSE command to ABM device and temporarily stops acquisition of data. The client program should call ResumeAcquisition in order to restart data acquisition.

Format:
```
[status] = abmsdk('PauseAcquisition');
```

Input Arguments:

None

Output Arguments:

1. SUCCESS (1) / FAILED (0)

Notes:

The elapsed time in the data files (Epoch & Offset, computed from the number of data samples obtained by the SDK from the device) will NOT reflect the pause state when data acquisition is re-started. The next consecutive sample number will be used after the pause. However, the absolute time (Hour, Minute, Second, MilliSecond) in the data files will indicate the pause in data acquisition.

Compatibility: X10,X24

ResumeAcquisition

Description:
Sends RESUME command to ABM device and restarts data acquisition.

Format:
```
[status] = abmsdk('ResumeAcquisition');
```

Input Arguments:
None

Output Arguments:
1. SUCCESS (1) / FAILED (0)

Notes:

Compatibility: X10, X24

StopAcquisition

Description:
Sends STOP command to ABM device and stops acquisition of data. This function resets the parameters in the SDK, thus the client program should initiate a NEW session prior to re-invoking StartAcquisition after this function is called.

Format:
```
[status] = abmsdk('StopAcquisition');
```

Input Arguments:
None

Output Arguments:
1. SUCCESS (1) / FAILED (0)

Notes:

StopAcquisition is usually called at the end of the session. Thus if additional data has to be collected after StopAcquisition function is executed, a new session has to be initiated. This involves calling SetDestinationFile, SetDefinitionFile(for applicable session types), and InitSession prior to calling StartAcquisition again.

For temporary halts in data acquisition use PauseAcquisition.

Compatibility: X10, X24

GetRawData

Description:
Gets raw data samples from the SDK.

Format:
```
fdata = abmsdk('GetRawData');
```

Input Arguments:
None

Output Arguments:
1. Type: double[r][c]
   fdata - r = samples, c = Epoch, Offset, Hour, Minute, Second, MilliSecond, Ch#1, Ch#2…..Ch#N (where, N=10 for X10 and N=24 for X24)

Notes:

ABM X10 & X24 devices have a sampling rate of 256, thus new samples are generated every ~4ms. In X10, two samples are combined into a single packet, thus the SDK receives new samples every ~8ms. In X24 a new packet is received every 4ms. Successive calls to GetRawData should have an appropriate intermediate delay, if new data samples are unavailable, NULL value will be returned.

X10 Channel mapping: EKG[1], POz[2], Fz[3], Cz[4], C3[5], C4[6], F3[7], F4[8], P3[9], P4[10]

X24 Channel mapping: F3[1], F1[2], Fz[3], F2[4], F4[5], C3[6], C1[7], Cz[8], C2[9], C4[10], CPz[11], P3[12], P1[13], Pz[14], P2[15], P4[16], POz[17], O1[18], Oz[19], O2[20], ECG[21], AUX1[22], AUX2[23], AUX3[24]

Compatibility: X10, X24

GetFilteredData

Description:
Gets filtered data samples from the SDK.

Format:
```
fdata = abmsdk('GetFilteredData');
```

Input Arguments:
None

Output Arguments:
1. Type: double[r][c]
   fdata - r = samples, c = Epoch, Offset, Hour, Minute, Second, MilliSecond, Ch#1, Ch#2…..Ch#N (where, N=10 for X10 and N=24 for X24)

Notes:

The following filters will be applied on the raw data:

1. 50hz Notch
2. 60hz Notch
3. 100hz Notch
4. 120hz Notch
5. 0.05hz High Pass
6. Median filter (order:256)

Compatibility: X10,X24

GetDeconData

Description:
Gets artifacts decontaminated data samples from the SDK.

Format:
```
fdata = abmsdk('GetDeconData');
```

Input Arguments:
None

Output Arguments:

1. Type: double[r][c]
   fdata - r = samples, c = Epoch, Offset, Hour, Minute, Second, MilliSecond, Ch#1, Ch#2…..Ch#N (where, N=10 for X10)

Notes:

ABM's proprietary artifact algorithm detects, extracts and compensates 5 types of artifacts: (1) Eye blink, (2) Saturation, (3) Excursion, (4) Spike, (5) EMG. The algorithm works on buffered data and is computationally intensive, thus would have a reasonable lag (1-2 seconds) compared to raw data samples. However, the time-stamping will be maintained accurately.

The SDK stores all information related to the detected artifacts in output files (Artifacts.csv & ArtifactInfo.csv files -- Please refer Appendix for the structure and details of the files).

Artifacts can be individually informed by registering callback functions in SDK. Refer SetArtifactsCallbackFuncs.

Compatibility: X10

GetPSDData

Description:

Gets Power Spectral Densities (PSD) from 1Hz to 128Hz of PROCESSED data for all EEG channels. The raw data is processed by artifact decontamination and smoothening using Kaiser window.

Format:

```
fdata = abmsdk('GetPSDData');
```

Input Arguments:

None

Output Arguments:

1. Type: double[r][c]
   fdata - r = epochs, c = Epoch, Offset, Hour, Minute, Second, MilliSec, PSD bin1 Ch#1…PSD bin128 Ch#1…..PSD bin128 Ch#N (where, N=10 for X10)

Notes:

Power Spectral Densities are calculated by averaging across 3 epoch overlays and reported every epoch (second). It is guaranteed that on average one result per second will be returned. Thus two successive calls to this function should have an intermediate delay of more than 1sec (minimum), thus when sampling faster NULL value will be returned.

PSD of EKG is NOT computed and is initialized to zero

Compatibility: X10

GetPSDDataraw

Description:
Gets Power Spectral Densities (PSD) from 1Hz to 128Hz of RAW data for all EEG channels.

Format:
```
fdata = abmsdk('GetPSDDataraw');
```

Input Arguments:

None

Output Arguments:

1. Type: double[r][c]
   fdata - r = epochs, c = Epoch, Offset, Hour, Minute, Second, Millisec, PSD bin1 Ch#1…PSD bin128 Ch#1…..PSD bin128 Ch#N (where, N=10 for X10)

Notes:

Compatibility: X10

GetBandOverallPSDData

Description:
Gets Power Spectral Densities (PSD) of specified bands averaged across all EEG channels. The PSDs are computed from RAW EEG data. The band PSDs are computed for two sets of channels: Referential & Differential. Differential PSDS are computed on digitally generated differential channels from referential inputs (for example, FzPOz, CzPOz, FzC3, C4C4 and F3Cz). The differential PSDs are used in ABM classification algorithms.

Format:
```
fdata = abmsdk(' GetBandOverallPSDData ');
```

Input Arguments:

None

Output Arguments:

1. Type: double[r][c]
   fdata - r = epochs, c = Epoch, Offset, Hour, Minute, Second, MilliSec, PSD band1 …PSD band M (where M is the number of bands)

Notes:

The default bands are: (1) Delata 1-2Hz, (2) ThetaSlow 2-5Hz, (3) ThetaFast 5-7Hz, (4) ThetaTotal 3-7Hz, (5) AlphaSlow 8-10Hz, (6) AlphaFast 10-12Hz, (7) AlphaTotal 8-12Hz, (8) Beta 13-29Hz, (9) Sigma 30-40Hz.

Compatibility: X10

GetBandOverallPSDRawData

Description:

Gets Power Spectral Densities (PSD) of specified bands averaged across all EEG channels. The PSDs are computed from PROCESSED (decontaminated and smoothened using Kaiser window) EEG data. The band PSDs are computed for two sets of channels: Referential & Differential. Differential PSDS are computed on digitally generated differential channels from referential inputs (for example, FzPOz, CzPOz, FzC3, C4C4 and F3Cz). The differential PSDs are used in ABM classification algorithms.

Format:

```
fdata = abmsdk(' GetBandOverallPSDRawData ');
```

Input Arguments:

None

Output Arguments:

1. Type: double[r][c]
   fdata - r = epochs, c = Epoch, Offset, Hour, Minute, Second, Millisec, PSD band1 …PSD band M (where M is the number of bands)

Notes:

The default bands are: (1) Delata 1-2Hz, (2) ThetaSlow 2-5Hz, (3) ThetaFast 5-7Hz, (4) ThetaTotal 3-7Hz, (5) AlphaSlow 8-10Hz, (6) AlphaFast 10-12Hz, (7) AlphaTotal 8-12Hz, (8) Beta 13-29Hz, (9) Sigma 30-40Hz.

Compatibility: X10

GetPSDBandwidthData

Description:
Gets Power Spectral Densities (PSD) of specified bands for all EEG channels individually. The PSDs are computed from RAW EEG data. The band PSDs are computed for two sets of channels: Referential & Differential. Differential PSDS are computed on digitally generated differential channels from referential inputs.

Format:
```
fdata = abmsdk('GetPSDBandwidthData');
```

Input Arguments:
None

Output Arguments:
1. Type: double[r][c]
   fdata - r = epochs, c = Epoch, Offset, Hour, Minute, Second, MilliSec, PSD bin1 Ch#1…PSD bin128 Ch#1…..PSD bin128 Ch#N (where, N=10 for X10)

Notes:
The default bands are: (1) Delata 1-2Hz, (2) ThetaSlow 2-5Hz, (3) ThetaFast 5-7Hz, (4) ThetaTotal 3-7Hz, (5) AlphaSlow 8-10Hz, (6) AlphaFast 10-12Hz, (7) AlphaTotal 8-12Hz, (8) Beta 13-29Hz, (9) Sigma 30-40Hz.

Compatibility: X10

GetPSDBandwidthRawData

Description:
Gets Power Spectral Densities (PSD) of specified bands for all EEG channels individually. The PSDs are computed from PROCESSED (decontaminated and smoothened using Kaiser window) EEG data. The band PSDs are computed for two sets of channels: Referential & Differential. Differential PSDS are computed on digitally generated differential channels from referential inputs.

Format:
```
fdata = abmsdk('GetPSDBandwidthData');
```

Input Arguments:

None

Output Arguments:

1. Type: double[r][c]

fdata - r = epochs, c = Epoch, Offset, Hour, Minute, Second, MilliSec, PSD bin1 Ch#1…PSD bin128 Ch#1…..PSD bin128 Ch#N (where, N=10 for X10)

Notes:

The default bands are: (1) Delata 1-2Hz, (2) ThetaSlow 2-5Hz, (3) ThetaFast 5-7Hz, (4) ThetaTotal 3-7Hz, (5) AlphaSlow 8-10Hz, (6) AlphaFast 10-12Hz, (7) AlphaTotal 8-12Hz, (8) Beta 13-29Hz, (9) Sigma 30-40Hz.

Compatibility: X10

GetBrainState

Description:

Gets B-Alert engagement and workload classifications.

Format:

```
fdata = abmsdk('GetBrainState');
```

Input Arguments:

None

Output Arguments:

1. Type: double[r][c]
   fdata - r = epochs, c = Epoch, Hour, Minute, Second, MilliSecond, Classification_Estimate, HighEngangement, LowEngagement, Distraction, Drowsy, WorkloadFBDS, WorkloadBDS, WorloadAverage.

Notes:

Classifications are calculated and reported every epoch (second). It is guaranteed that on average one result per second will be returned. Thus two successive calls to this function should have an intermediate delay of more than 1sec (minimum), otherwise if sampled faster, NULL values will be returned when there is no data available.

-99999 will be reported if classifications are not computed due to invalid PSD values. This usually happens if the datapoints are corrupted with excessive noise and sufficient EEG data cannot be extracted.

Compatibility: X10

InitZScoreData

Description:
Initializes parameters for computing z-scores

Format:
```
[status] = abmsdk('InitZScoreData',' List' );
```

Input Arguments:

1. Type: double[r][c]
   List - string with symbolic names of z-score data sources separated by commas:
   Classifications (class_higheng, class_loweng, class_distr, class_sleeponset, wl_ave),
   Heart rate (HeartRate),
   PSD values (OvRaw<Ref|Diff>_<band_index>, for example OvRawRef_0 = overall raw PSD from all referential channels in band 0. The default bands are: (1) Delata 1-2Hz, (2) ThetaSlow 2-5Hz, (3) ThetaFast 5-7Hz, (4) ThetaTotal 3-7Hz, (5) AlphaSlow 8-10Hz, (6) AlphaFast 10-12Hz, (7) AlphaTotal 8-12Hz, (8) Beta 13-29Hz, (9) Sigma 30-40Hz,
   Bandwidth PSD values (<channel_name>_<band_index>, for example, po_3 = bandwidth raw PSD from all referential channels in band 3 in channel PO.

   Example – 'class_higheng,class_loweng,class_distr, class_sleeponset,wl_ave,HeartRate,OvRawRef_0,OvRawDif_1,po_3,fzpo_6,c3_7'

Output Arguments:
SUCCESS (1) / FAILED (0)

Notes:
InitZScoreData must be called before StartAcquisition function.

Compatibility: X10

GetZScoreData

Description:
Gets computed Z-score values.

Format:

```
fdata = abmsdk('GetZScoreData');
```

Input Arguments:

None

Output Arguments:

1. Type: double[r][c]
   fdata - r = epochs, c = Epoch, Hour, Minute, Second, MilliSecond, ZScore values in the same order as specified in IntiZScoreData.

Notes:

InitZScore function must be called prior to calling GetZScoreData function.

ZScores are reported every epoch, hence intermediate delay between two successive calls must be 1 second (minimum), otherwise if sampled faster, NULL values will be returned when there is no data available.

Compatibility: X10

ResetZScoreData

Description:

Resets parameters for computing z-scores

Format:

```
[status] = abmsdk('ResetZScoreData',' List' );
```

Input Arguments:

1. Type: double[r][c]
   List – Refer InitZScoreData

Output Arguments:

SUCCESS (1) / FAILED (0)

Notes:

Compatibility: X10

GetQualityChannelData

Description:
Gets % of 1)overall data quality across all channels and 2) data quality (EMG & Other artifacts) for each individual channel.

Format:
```
fdata = abmsdk('GetQualityChannelData');
```

Input Arguments:

None

Output Arguments:

1. Type: double[r][c]
   fdata - r = epochs, c = Epoch, Hour, Minute, Second, MilliSecond, Overall Quality, EMG in Ch#2, Other in Ch#2…..EMG in Ch#10, Other in CH#10 (Note, that in X10 Ch#1 is EKG and no quality data is computed form EKG)

Notes:
Data quality is computed per epoch, so sufficient delay should be used between successive calls.

Compatibility: X10

GetMovementData

Description:
Gets movement-value and movement-level computed from the output of the 3-axis accelerometer integrated in the headset.

Format:
```
fdata = abmsdk('GetMovementData');
```

Input Arguments:

None

Output Arguments:

1. Type: double[r][c]
   fdata - r = epochs, c = Epoch, Hour, Minute, Second, MilliSecond, movement-value, movement-level

Notes:
Movement value is reported as the sum of the change in the two dominanant movement angles derived from raw 3-axis accelerometer tilt data. Movement scale (reported as a value between 0 and 5) is computed from the change in movement angles, based on a proprietory ABM algorithm.

Sampling interval of the output variable – per epoch.

Compatibility: X10, X24

GetEKGData

Description:
Gets heart rate values computed from EKG.

Format:
```
fdata = abmsdk('GetEKGData');
```

Input Arguments:
None

Output Arguments:
1. Type: double[r][c]
   fdata - r = epochs, c = Epoch, Offset,Hour, Minute, Second, MilliSecond, Heart Rate, Inter beat Interval, Beat Quality (good=1,else=0), Packet Type (0=sec by sec value, 1=beat by beat value, 2=Reserved, 3=GUI presentation value)

Notes:
Heart Rate is computed per epoch, so sufficient delay should be used between successive calls.

The value presented on the GUI is checked continuously for valid EKG. In case valid EKG was not detected, the output (display) is initialized to min value. Sec by sec & beat by beat value produce outputs continuously but the beat quality is marked.

Compatibility: X10, X24

CheckImpedances

Description:
Gets impedance values of all EEG channels.

Format:
```
[chName, chStatus, chValue] = abmsdk('CheckImpedances');
```

Input Arguments:
None

Output Arguments:
1. Type: char(N)(20)
   chName – Channel Names, where N = number of channels including reference channel (for example X10 has 9 EEG + 1 Reference channel)
2. Type: bool(N)
   chStatus – Channel Impedance Status (0=bad, 1=good)
3. Type: float(N)
   chValue – Channel Impedance Value in KOhms

Notes:
Impedance computation takes about 4-5 seconds / channel.

CheckImpedances should be called only after a new session is initiated using InitSession.

-9999 is reported if the impedances are out of range / invalid.

Compatibility: X10, X24

TechnicalMonitoring

Description:
Performs quality check of all EEG channels for a specified time.

Format:
```
[chName, chStatus] = abmsdk('TechnicalMonitoring',tsec);
```

Input Arguments:
1. Type: int
   tsec – # of seconds to monitor

Output Arguments:

1. Type: char(N)(20)
   chName – Channel Names, where N = number of channels
2. Type: bool(N)
   chStatus – Channel quality Status (0=bad, 1=good)

Notes:

TechnicalMonitoring should be called only after a session is successfully initiated using InitSession.

Compatibility: X10

GetThirdPartyData

Description:

Returns third party data acquired using ABM External Sync Unit (ESU).

Format:

```
cdata = abmsdk('GetThirdPartyData' );
```

Input Arguments:

None

Output Arguments:

1. Type: unsigned char [n]
   cdata – 1-D array of third party data

Notes:

The Matlab interface will filter out the third party data from proprietary ABM packet, thus the return value of this function does NOT follow any packet structure.

If no third party data is available, NULL will be returned.

Compatibility: X10, X24

GetTimeStampsStreamData

Description:
Returns timestamps for raw and processed data samples.

Format:
```
cdata = abmsdk('GetTimeStampsStreamData' ,' TimestampType' );
```

Input Arguments:

1. Type: int
   TimestampType – TIMESTAMP_RAW=0, TIMESTAMP_PSD=1,TIMESTAMP_DECON=2, TIMESTAMP_CLASS=3, TIMESTAMP_EKG=4

Output Arguments:

1. Type: unsigned char [n]
   cdata – 1-D array of timestamps (format = 4-bytes, high byte first)

Notes:
This function must be called immediately after the data acquisition function. For example, if GetRawData returns 2 epochs of data, then GetTimeStampsStreamData when called immediately after GetRawData (with Timestamp Type = TIMESTAMP_RAW), will return two 4-bytes timestamps corresponding to the 2 epochs.

ABM devices connect to two types of ABM receivers: 1) ABM Dongle, 2)Multi-channel ESU (MC-ESU). When using ABM dongle, windows time (8-bytes format) will be returned. When using MC-ESU, 4-bytes timestamp generated in the MC-ESU (using a dedicated hardware timer) will be returned.

Compatibility: X10, X24 (Note: For X24, PSD/DECON/CLASS should NOT be used)

GetCurrentSDKMode

Description:
Returns the current operating mode of SDK

Format:
```
idata = abmsdk('GetCurrentSDKMode' );
```

Input Arguments:

None

Output Arguments:

1. Type: int
   idata – SDK_WAITING_MODE = -1, SDK_NORMAL_MODE=0, SDK_IMPEDANCE_MODE=1, SDK_TECHNICALMON_MODE =2.

Notes:

Compatibility: X10, X24

ShowChannel

Description:
Displays graphical plot of input data.

Format:
```
[status] = abmsdk('ShowChannel' ,chNum,chGain,chTScale);
```

Input Arguments:

1. Type: int
   chNum: channel number
2. Type: int
   chGain: Gain (should be <= 20)
3. Type: int
   chTScale: timescale (should be <=60 seconds)

Output Arguments:

SUCCESS (1) / FAILED (0)

Notes:
It is recommended to pause few seconds before invoking this function multiple times.

Compatibility: X10, X24

AlignChannels

Description:
Aligns the graphical plots generated by ShowChannel.

Format:
```
[status] = abmsdk('AlignChannels' );
```

Input Arguments:

None

Output Arguments:

SUCCESS (1) / FAILED (0)

Notes:

Compatibility: X10, X24

SetArtifactsCallbackFuncs

Description:

Registers callback function to inform artifacts as and when they are detected. The five artifacts (Eyeblink, Excursion, Saturation, Spike, EMG) will be reported separately.

Format:

```
[status] = abmsdk('SetArtifactsCallbackFuncs' );
```

Input Arguments:

None

Output Arguments:

None

Notes:

Compatibility: X10

RegisterCallbackImpedanceElectrodeFinished

Description:

Registers callback function to return impedance values of each channel as and when it is completed. Unlike CheckImpedances, this function will report each channel separately.

Format:

```
[status] = abmsdk(' RegisterCallbackImpedanceElectrodeFinished' );
```

Input Arguments:

None

Output Arguments:

None

Notes:

Impedance computation takes approx 5sec/channel.

The channel index is reported as channel numbers. For example, reference channel = 0, channel 1 = 1, etc.

Compatibility: X10,X24

RegisterCallbackDataArrived

Description:

Registers callback function to return sample number when a new sample is acquired from the device.

Format:

```
[status] = abmsdk(' RegisterCallbackDataArrived' );
```

Input Arguments:

None

Output Arguments:

None

Notes:

The data arrived callback is called for every sample and is usually used for synchronization purposes. For example it can be used to send the elapsed time (epoch and offset) and clock time (hour, min, sec, miilisec) to a third party application whenever the callback returns.

Since the callback will be invoked for every sample, it is recommended not to put any computationally intensive application code inside this callback function.

Compatibility: X10,

RegisterCallbackOnStatusInfo

Description:
Registers callback to return information such as: battery voltage, timestamp, missed block count, current SDK mode, and last error code. The callback is invoked at the rate of 2Hz.

Format:
```
[status] = abmsdk(' RegisterCallbackOnStatusInfo' );
```

Input Arguments:
None

Output Arguments:
None

Notes:

Compatibility: X10

RegisterCallbackOnError

Description:
Registers callback that returns internally generated error codes in the SDK.

Format:
```
[status] = abmsdk(' RegisterCallbackOnError' );
```

Input Arguments:
None

Output Arguments:
None

Notes:
The list of error codes can be found in Appendix-A.

Compatibility: X10,

RegisterCallbackMissedBlocks

Description:
Registers callback that returns Start-epoch and End-epoch of missed data blocks.

Format:
```
[status] = abmsdk(' RegisterCallbackMissedBlocks' );
```

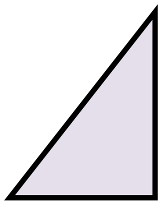
Input Arguments:
None

Output Arguments:
None

Notes:

Compatibility: X10,

4. B-Alert Client Programming Interface



4. B-Alert Client Programming Interface

4.1 Programming Notes 97

4.1.1 Interface 97

4.1.2 Configuration Steps 97

4.2. Validation 97

4.3. Troubleshooting Guide 98

4.4. Application Programming Interface 99

OpenConnection 99

CloseConnection 100

GetTimeStamp 101

GetEbsTimeStamp 101

AgetDeviceInfo 102

AgetChannelMapInfo 103

AgetRaw 104

AgetDecon 105

AgetPSD 106

AgetPSDraw 107

AgetPSDBandwidthData 107

AgetPSDBandwidthRawData 108

AgetBandOverallPSDData 109

AgetBandOverallPSDRawData 110

AgetCWPC 111

AgetEKG 111

AgetQualityChannel 112

AgetMovement 113

AgetAcc 114

AgetAngles 115

AgetThirdParty 116

ASetArtifactsCallbackFuncs 117

AgetNotification 118

4.1 Programming Notes

4.1.1 Interface

The B-Alert Client programming interface can be used to bypass the data presentation module (B-Alert Gauge GUI) and interface to the B-Alert SDK via the B-Alert Control GUI directly. The data presentation module is programmed as a network client that connects to the B-Alert Control GUI (server) over a TCP/IP network. The architecture supports more than one connection and thus multiple such clients can be run simultaneously. The Application Programming Interface (APIs) is provided through ABM_Datastreaming.dlll (found in SDK\bin). The corresponding lib file (found in SDK\lib) and header files (found in SDK\include\TCP) must also be included in the client program, depending on the type of build used in the project (static or dynamic).

The server buffers up-to 60 seconds of data from the SDK, thus the client APIs can be invoked at convenient intervals. Several client applets can perform acquisition at the same time and all of them will receive data packets with identical content from the server. Note that the artifact decontaminated data and processed outputs may be delayed up-to 5 seconds relative to raw data because of the computationally intensive nature of the algorithms. The delay will partly depend on the processing capability of the computer as well as the amount of noise in the data.

4.1.2 Configuration Steps

1. Install BAlert software in the server PC. Note that both server and client applications can run in the same PC, if necessary, through the native socket interface (IP address 127.0.01).

2. Configure data streaming in B-Alert Control GUI - The configuration dialog can be invoked from the menu – Settings/Configure Data Streaming. The following must be configured: i) Type of data streamed ii) Protocol used: *TCPandUDP* option transmits both data and timestamps, while the *UDPOnly* option transmits only timestamps. iii) IP Port – port to which the client connects.

3. Use Acquire & Retransmit button in the B-Alert Control GUI to start the server. Note that the ABM device must be synchronized to the ABM receiver prior to starting the server (Refer B-Alert User Manual for details).

4.2. Validation

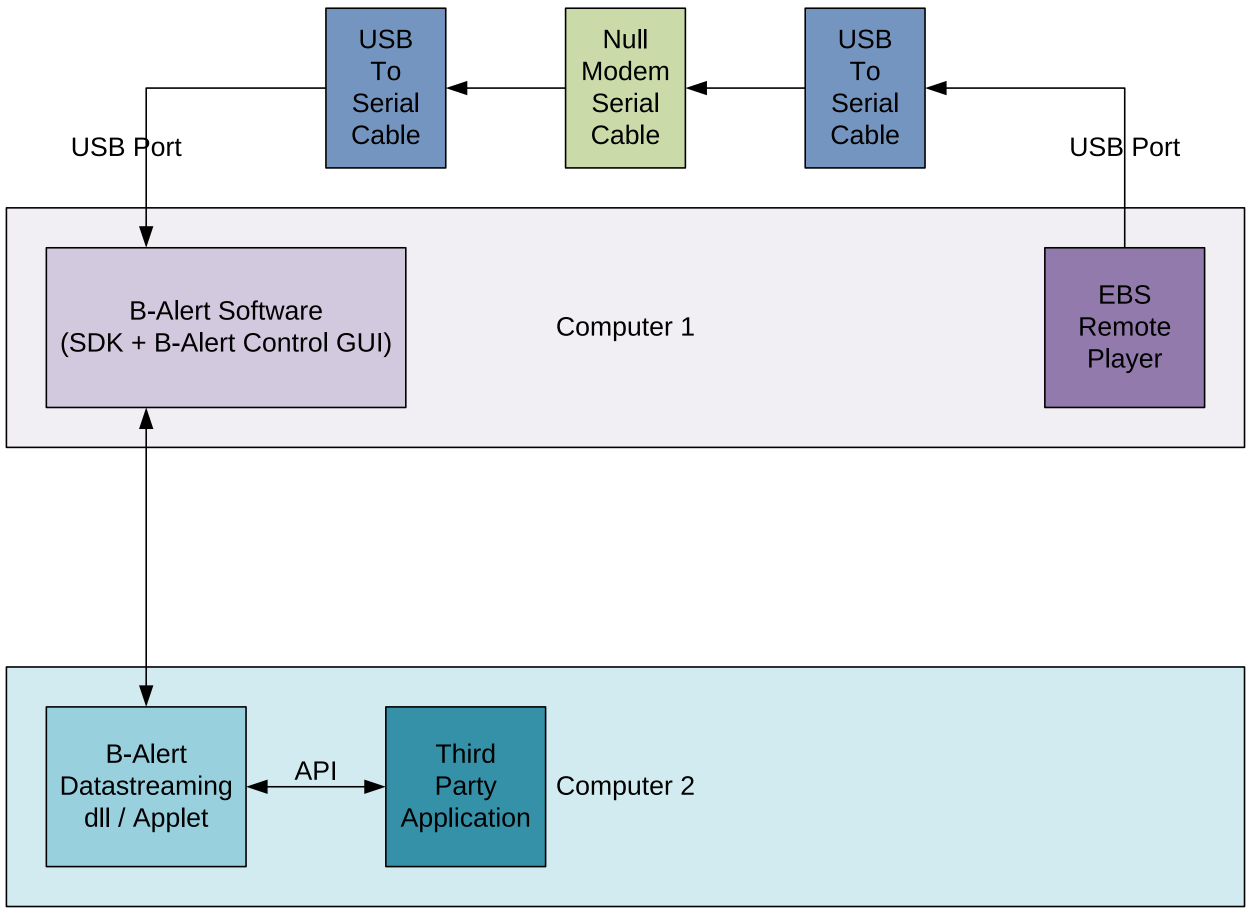


Figure 9: B-Alert Client Programming Interface validation setup

The validation procedure is identical to as detailed in the earlier chapter. The output files generated by the above setup must be compared with the Gold Standard files (in ABM\EEG\Data folder). Close similarity between the values must be achieved. Sample clients in both C++ and C# are provided in the installation for reference.

4.3. Troubleshooting Guide

In some cases the client will not receive any data from the server due to connection issues. This section will help diagnose such errors.

1. Check datastreaming configuration in B-Alert Control GUI. i) The Port number specified in the datastreaming interface of the server must be identical to the one used by the client. ii) The correct protocol (TCP / UDP) must be used, for example, if the server is set to UDP, only timestamps will be available at the client side. iii) The data selection boxes must be ticked.

2. Check IP address of the server: The ip address of the server can be viewed using ipconfig Windows command. Invoke Windows command interface by typing *cmd* in Windows Run menu. Verify that the correct IP address is specified in the client program.

3. Check network connectivity to the server: Ping the ip address of the server from the client computer and monitor for response. If the server is accessible then you should get multiple replies back from the server.

4. Check availability of port: Invoke *Netstat –a –o |more* in Windows command interface (using Run menu) to see a list of ports used in the server computer. If the port specified in the datastreaming interface of the B-Alert Control GUI is not listed, then possibly the firewall is blocking the port. If the port is specified, compare the PID attached to the port with the PID of B-Alert software in Windows Taskmanager. If the PIDs are different, then the port is not available for B-Alert - specify a different port number.

5. Check firewall settings: Use Windows Control Panel to either disable the firewall (not recommended) or provide access to B-Alert software at the server side.

4.4. Application Programming Interface

OpenConnection

Description:
Open connection toB-Alert Control GUI (server).

Format:
```
long  OpenConnection (char* sServerAdd, int &nHandle, BOOL bProtocol )
OpenConnection(char* sInput, int &handle, int bTCP = ABM_DS_CLIENT_TCP_VIEW);
```

Input Arguments:
1. sInput – IP address and Port Number of the server
2. nHandle – connection handle
3. bTCP – type of protocol (TCP = 1, UDP = 0, TCP CONTROL = 2)

Output Arguments:
1. Connection status (SUCCESS=1, FAILED=0)

PseudoCode:

```
char* sServerIP =  "127.0.0.1 4500" ; // Format is   "IP SPACE PORT"
int nHandle ;
bool bProtocol = 1; //TCP
if(OpenConnection(sServerIP, nHandle, bProtocol)
   //SUCCESS
else
   //FAILED
```

Notes:

Available in TCP & UDP protocols.

CloseConnection

Description:
Closes connection with B-Alert Control GUI (server).

Format:
```
long  CloseConnection (int nHandle )
```

Input Arguments:
1. nHandle – connection handle

Output Arguments:
1. Connection status (SUCCESS=1, FAILED=0)

PseudoCode:

Notes:

Available in TCP & UDP protocols.

GetTimeStamp

Description:
Gets elapsed time in clock format.

Format:
```
long  GetTimeStamp (float &fHour, float& fMinute, float& fSeconds, float& fMilliseconds, int nHandle)
```

Input Arguments:
1. fHour , fMinute, fSeconds, fMilliseconds - timestamps of datapoints in EBS file.
2. nHandle – connection handle

Output Arguments:
1. Timestamp status (VALID=1, INVALID=0)

PseudoCode:

Notes:

Available in TCP & UDP protocols.

GetEbsTimeStamp

Description:
Gets elapsed time in epoch/offset format (One epoch=256 offset values).

Format:
```
long  GetEbsTimeStamp (int& nEpoch, int& nOffset, int nHandle)
```

Input Arguments:
1. nEpoch – epoch number
2. nOffset – offset number
3. nHandle – connection handle

Output Arguments:

1. Timestamp status (VALID=1, INVALID=0)

PseudoCode:

Notes:

Available in TCP & UDP protocols.

AgetDeviceInfo

Description:
Gets information about the ABM device connected to the B-Alert SDK.

Format:
```
_ABM_DATA_DEVICE_INFO * AgetDeviceInfo (int nHandle)
```

Input Arguments:

1. nHandle – connection handle

Output Arguments:

1. Type: Pointer to structure _ABM_DATA_DEVICE_INFO
```
char  chDeviceName[256]; //device's name
int   nCommPort; //comm port
int   nECGPos; //ecg position
int   nNumberOfChannel; //number of channel
int   nNumberPSDBands; //number of PSD bandwidths
int   nNumberPSDBandsOverall; //number of PSD bandwidths (overall)
int  nDeconChannels;
int  nPSDRawChannels;
int  nPSDClassChannels;
int  nQualityCheckChannels;
       char  sessionID[ABM_DATA_SESSION_ID_LENGTH]; //SessionID
```

PseudoCode:

Notes:

This API can be used to check the device status and is usually called at the beginning of the third party client program.

AgetChannelMapInfo

Description:
Gets information about the channel map of the connected device. It includes, channel name, number of channel used and number of channels represented in AgetQualityChannel

Format:
`char* AgetChannelMapInfo (int &nLength, int nHandle)`

Input Arguments:
1. nLength – length of character array returned
2. nHandle – connection handle

Output Arguments:
1. Type: char array filled by the following strcture

```
typedef struct _EEGCHANNELS_INFO {
        char    cChName[24][20];              //names of available channels
        bool    bChUsed[24];                     //1=channel available, 
0=channel absent
        bool    bChUsedQualityData[24] ; //1=represented in AgetQualityChannel, 
0=absent              }
```

PseudoCode:

Notes:

AgetRaw

Description:

Gets raw samples of EEG (and EKG) from the server.

Format:

```
float* AgetRaw (int& nCount, int nHandle)
```

Input Arguments:

1. nCount – The number of samples that arrived per channel between successive calls
2. nHandle – connection h andle

Output Arguments:

1. Type: float*
   Pointer to array of float values containing raw data samples. The size of the return array = (nChannel+6)*nCount, where, nChannel is the number of channels in the ABM device (for example, X10 has 10 channels), nCount is the number of samples arrived. .

PseudoCode:

Notes:

If no data is available, NULL will be returned.

Example structure of the output argument (ABM X10 device):
pData[0]=Epoch, pData[1]=Offset //Epoch & Offset of First sample
pData[2]=Hour, pData[3]=Min, pData[4]=Sec, pData[5]=MiliSec // SDK timestamp
pData[6]=Sample1_Channel1…
pData[15] = Sample1_Channel10
pData[16]=Epoch, pData[17]=Offset //Epoch & Offset of Second sample
pData[18]=Hour, pData[19]=Min, pData[20]=Sec, pData[21]=MiliSec // SDK timestamp
pData[22]=Sample2_Channel1…

X10 Channel mapping: EKG[1], POz[2], Fz[3], Cz[4], C3[5], C4[6], F3[7], F4[8], P3[9], P4[10]

X24 Channel mapping: F3[1], F1[2], Fz[3], F2[4], F4[5], C3[6], C1[7], Cz[8], C2[9], C4[10], CPz[11], P3[12], P1[13], Pz[14], P2[15], P4[16], POz[17], O1[18], Oz[19], O2[20], EKG[21], AUX1[22], AUX2[23], AUX3[24]

The server always sends one packet at a time, thus nCount = 1.

AgetDecon

Description:
Gets artifact decontaminated EEG samples from the server.

Format:
```
float* AgetDecon (int& nCount, int nHandle)
```

Input Arguments:
1. nCount – The number of samples that arrived per channel between successive calls
2. nHandle – connection h andle

Output Arguments:
1. Type: float*
   Pointer to array of float values containing decontaminated data samples. The size of the return array = (nChannel+6)*nCount, where, nChannel is the number of channels in the ABM device (for example, X10 has 10 channels), nCount is the number of samples arrived.

PseudoCode:

Notes:

If not data is available, NULL will be returned.

Artifact detection and decontamination are computationally intensive processes, thus the outputs may lag a few seconds relative to raw data. The delay will depend on the processing capability of the server side PC as well as the amount of noise in the data.

The server always sends one packet at a time, thus nCount = 1.

Artifacts are not detected in EKG signal, and it is send without any artifact removal.

AgetPSD

Description:
Gets Power Spectral Densities (PSD) of 1Hz to 128Hz from PROCESSED data for all EEG channels. The processing of raw data include: artifact decontamination, smoothening using Kaiser window and averaging across 3 epoch overlays.

Format:
```
float* AgetPSD (int &nEpoch, int nHandle)
```

Input Arguments:
1. nEpoch – number of epochs returned
2. nHandle – connection handle

Output Arguments:
1. Type: float*
   Pointer to array of float values containing PSD values from each channel. The size of the return array = (128*nChannel+5)*nEpoch, where, nChannel is the number of EEG channels in the ABM device (for example, ABM X10 has 9 EEG channels), nEpoch is the number of epochs.

PseudoCode:

Notes:

If no PSD values are available between successive calls, NULL will be returned. PSDs are computed from artifact decontaminated data, thus may have delays similar to AgetDecon API.

Example structure of the output argument (ABM X10 device):
pData[0]=Epoch, //Epoch 1
pData[1]=Hour, pData[2]=Min, pData[3]=Sec, pData [4]=MilliSec // SDK timestamp
pData[5]=PSDHz1_Epoch1_Channel1
pData[6]=PSDHz2_ Epoch1_Channel1….

PSDs are not computed for EKG channel (or AUX channels in X24), thus they are not tranmistted.

The server always sends one packet at a time, thus nEpoch= 1.

AgetPSDraw

Description:

Gets Power Spectral Densities (PSD) of1Hz to 128Hz from RAW EEG data for all EEG channels. No artifact decontimation and Kaiser window smoothening is done on EEG samples prior to computation of PSDs for this function. Three-epoch overlays are however used for averaging.

Format:

```
float* AgetPSDraw (int &nEpoch, int nHandle)
```

Input Arguments:

1. nEpoch – number of epochs returned
2. nHandle – connection handle

Output Arguments:

1. Type: float*
   Pointer to array of float values containing PSD values from each channel. The size of the return array = (128*nChannel+5)*nEpoch, where, nChannel is the number of EEG channels in the ABM device (for example, ABM X10 has 9 EEG channels), nEpoch is the number of epochs.

PseudoCode:

Notes:

Refer AgetPSD for sampling interval, output variable format, etc.

AgetPSDBandwidthData

Description:

Gets Power Spectral Densities (PSD) from PROCESSED EEG data for specified bands in individual EEG channels. Bandwidths from both referential and differential (computed by digitally subtracting referential channels) channels are reported.

Format:

```
float* AgetPSDBandwidthData (int &nEpoch, int nHandle)
```

Input Arguments:

1. nEpoch – number of epochs returned
2. nHandle – connection handle

Output Arguments:

1. Type: float*
   Pointer to array of float values containing PSD values of specified bands for each channel. The size of the return array = nSize*nEpoch, where, nSize is the number of bytes returned per epoch and nEpoch is the number of epochs.

PseudoCode:

Notes:

If no data is available, NULL will be returned. PSDs are computed from artifact decontaminated data, thus may have delays similar to AgetDecon API.

The default bands are: (1) Delata 1-2Hz, (2) ThetaSlow 2-5Hz, (3) ThetaFast 5-7Hz, (4) ThetaTotal 3-7Hz, (5) AlphaSlow 8-10Hz, (6) AlphaFast 10-12Hz, (7) AlphaTotal 8-12Hz, (8) Beta 13-29Hz, (9) Sigma 30-40Hz.

Example structure of the output argument (ABM X10 device, 9 channels of EEG):
pData[0]=Epoch, //Epoch 1
pData[1]=Hour, pData[2]=Min, pData[3]=Sec, pData[4]=MiliSec // SDK timestamp
pData[5]=Band1_Refchannel1, pData[6]=Band2_Refchannel1… //referential channels

The server always sends one packet at a time, thus nEpoch= 1.

AgetPSDBandwidthRawData

Description:
Gets Power Spectral Densities (PSD) from RAW EEG data for specified bands in individual EEG channels. Bandwidths from both referential and differential (computed by digitally subtracting referential channels) channels are reported.

Format:
```
float* AgetPSDBandwidthRawData (int &nEpoch, int nHandle)
```

Input Arguments:

1. nEpoch – number of epochs returned
2. nHandle – connection handle

Output Arguments:

1. Type: float*
   Pointer to array of float values containing PSD values of specified bands for each channel. The size of the return array = nSize*nEpoch, where, nSize is the number of bytes returned per epoch and nEpoch is the number of epochs.

PseudoCode:

Notes:

Refer AgetPSDBandwidthData for output sampling interval, output variable structure, etc.

AgetBandOverallPSDData

Description:
Gets Power Spectral Densities (PSD) of specified bands averaged across all EEG channels. The PSDs are computed from PROCESSED EEG data. The band PSDs are computed for two sets of channels: Referential & Differential.

Format:
```
float* AgetBandOverallPSDData (int &nEpoch, int nHandle)
```

Input Arguments:

1. nEpoch – number of epochs returned
2. nHandle – connection handle

Output Arguments:

1. Type: float*
   Pointer to array of float values containing PSD values of specified bands for the two groups. The size of the return array = (2*nBands+5)*nEpoch, where, nBands is the number of bands, nEpoch is the number of epochs.

PseudoCode:

Notes:

If no data is available, NULL will be returned. PSDs are computed from artifact decontaminated data, thus may have delays similar to AgetDecon API.

Example structure of the output argument (ABM X10 device):
pDatа[0]=Epoch, //Epoch 1
pData[1]=Hour, pData[2]=Min, pData[3]=Sec, pData[4]=MillSec // SDK timestamp
pData[5]=Band1, pData[6]=Band2… // Raw referential signals

The server always sends one packet at a time, thus nEpoch= 1.

AgetBandOverallPSDRawData

Description:
Gets Power Spectral Densities (PSD) of specified bands averaged across all EEG channels. The PSDs are computed from RAW EEG data. The band PSDs are computed for two sets of channels: Referential & Differential.

Format:
`float* AgetBandOverallPSDRawData (int &nEpoch, int nHandle)`

Input Arguments:
1. nEpoch – number of epochs returned
2. nHandle – connection handle

Output Arguments:
1. Type: float*
   Pointer to array of float values containing PSD values of specified bands for the two groups. The size of the return array = (2*nBands+5)*nEpoch, where, nBands is the number of bands, nEpoch is the number of epochs.

PseudoCode:

Notes:

Refer AgetBandOverallPSDData for output sampling interval, output variable structure, etc

AgetCWPC

Description:
Gets B-Alert engagement and workload classification indices.

Format:
```
float*  AgetCWPC (int &nEpoch, int nHandle)
```

Input Arguments:
1. nEpoch – number of epochs returned
2. nHandle – connection handle

Output Arguments:
1. float array containing CWPC data

PseudoCode:

Notes:

If no classifications are available, NULL will be returned.

-99999 will be reported if classifications are not computed due to invalid PSD values. This usually happens when the datapoints are corrupted with excessive noise and sufficient EEG samples cannot be salvaged.

The server always sends one packet at a time, thus nEpoch= 1.

AgetEKG

Description:
Gets heart rate computed from EKG signal.

Format:

```
float* AgetEKG (int &nEpoch, int nHandle)
```

Input Arguments:

1. nEpoch – number of epochs returned
2. nHandle – connection handle

Output Arguments:

1. Type: float*
   Pointer to array of float values containing heart rate values. The size of the return array = (4+6)*nCount, where nCount is the number of epochs. The number of epochs may vary with the delay between two calls to the function.

PseudoCode:

Notes:

If no heart rate is available, NULL will be returned.

Example structure of the output argument (ABM X10 device):
pData[0]=Epoch, pData[1]=Offset //Epoch1 & Offset1
pData[2]=Hour, pData[3]=Min, pData[4]=Sec, pData[5]=MiliSec // SDK timestamp
pData[6]=Heart Rate,
pData[7]=Inter beat interval,
pData[8]=beat quality, //good=1, otherwise=0
pData[9]=packet type, // (0=sec by sec value, 1=beat by beat value, 2=Reserved, 3=GUI presentation value)

The value presented on the GUI is checked continuously for valid EKG. In case valid EKG was not detected, the output (display) is initialized to min value. Sec by sec & beat by beat value produce outputs continuously but the beat quality is marked.

AgetQualityChannel

Description:
Gets % of 1) overall data quality across all channels, 2) EMG & Other artifacts for each individual channel.

Format:

```
float* AgetQualityChannel (int &nEpoch, int nHandle)
```

Input Arguments:

1. nEpoch – number of epochs returned
2. nHandle – connection handle

Output Arguments:

1. Type: float*
   Pointer to array of float values containing overall data quality and the % of EMG and artifacts in each individual channel. The size of the return array = (2*nChannel+6+1)*nEpoch, where, nChannel is the number of EEG channels in the device (see AgetChannelMapInfo), nEpoch is the number of epochs.

PseudoCode:

Notes:

If no values are available between successive calls, NULL will be returned.

Example structure of the output argument (ABM X10 device):
pData[0]=Epoch, pData[1]=Offset //Epoch & Offset of First sample
pData[2]=Hour, pData[3]=Min, pData[4]=Sec, pData[5]=MiliSec // SDK timestamp
pData[6]=Overall quality across channels (% of good data),
pData[7]= % of data contaminated with EMG in EEG channel # 1
pData[8]= % of data contaminated with other artifacts in EEG channel # 1
pData[9]= % of data contaminated with EMG in EEG channel # 2
pData[10]= % of data contaminated with other artifacts in EEG channel # 2

The server always sends one packet at a time, thus nEpoch= 1.

AgetMovement

Description:

Gets movement-value and movement-level computed from the the 3-axis accelerometer data transmitted from ABM devices.

Format:

```
float* AgetMovement (int &nEpoch, int nHandle)
```

Input Arguments:

1. nEpoch – number of epochs returned
2. nHandle – connection handle

Output Arguments:

2. Type: float*
   Pointer to array of float values containing movement-value and movement-level in each epoch. The size of the return array = (2+5)*nEpoch, where nEpoch is the number of epochs.

PseudoCode:

Notes:

If no data is available, NULL will be returned.

Movement value is reported as the sum of the change in the two dominanant movement angles derived from raw 3-axis accelerometer tilt data. Movement scale (reported as a value between 0 and 5) is computed from the change in movement angles, based on a proprietory ABM algorithm.

Example structure of the output argument (ABM X10 device):
pDatal[0]=Epoch, //epoch 1
pData[1]=Hour, pData[2]=Min, pData[3]=Sec, pData[4]=MiliSec // SDK timestamp
pData[5]=Mvmt_value, //movement value
pData[6]=Mvmt_level,   // movement level

AgetAcc

Description:
Gets 3-axis accelerometer data transmited from ABM devices in raw format.

Format:
`float* AgetACC (int& nCount, int handle);`

Input Arguments:

1. nCount – number of data packets returned
2. nHandle – connection handle

Output Arguments:

1. Type: Pointer to float

Pointer to array of float values containing accelerometer data. The size of the return array = (3+6)*nCount, where nCount is the number samples

PseudoCode:

Notes:

If no data is available, NULL will be returned.

Example structure of the output argument:
pData[0]=Epoch, //epoch 1
pData[1]=Offset, //offset 1
pData[2]=Hour, pData[3]=Min, pData[4]=Sec, pData[5]=MiliSec // SDK timestamp
pData[6]=Tilt_X, // tilt in raw format- X axis
pData[7]=Tilt_Y, // tilt in raw format- Y axis
pData[8]=Tilt_Z, // tilt in raw format- Z axis

AgetAngles

Description:
Gets 3-axis accelerometer data transmited from ABM devices in angles format.

Format:
```
float* AgetAngles (int& Count, int handle);
```

Input Arguments:
1. nCount – number of datapackets returned
2. nHandle – connection handle

Output Arguments:
1. Type: Pointer to float
   Pointer to array of float values containing angles derived from accelerometer data. The size of the return array = (3+6)*nCount, where nCount is the number of samples returned

PseudoCode:

Notes:

If no data is available, NULL will be returned.

Example structure of the output argument:
pData[0]=Epoch, //epoch 1
pData[1]=Offset, //offset 1
pData[2]=Hour, pData[3]=Min, pData[4]=Sec, pData[5]=MiliSec // SDK timestamp
pData[6]=Tilt_X, // tilt in angles format- X axis
pData[7]=Tilt_Y, // tilt in angles format- Y axis
pData[8]=Tilt_Z, // tilt in angles format- Z axis

AgetThirdParty

Description:
Gets third party data acquired using ABM External Sync Unit (ESU).

Format:
`char* AgetThirdParty (int &nCount, int nHandle)`

Input Arguments:
1. nCount – number of third party packets with fixed length = 256
2. nHandle – connection handle

Output Arguments:
1. Type: char*
   Pointer to array of third party data.

PseudoCode:

Notes:

Refer ABM MC-ESU programming interface section for details regarding configuring, sending and decoding data.

Third party packets available at the client side have a fixed length of 256 bytes. Unused bytes are padded with zeros. The structure of the packet is as follows:
Flag (0x56, 0x56) – 2 bytes

Message Counter	– 1 byte
ESU Timestamp	- 4 bytes
Packet Length	- 2 bytes
Packet Type	- 1 byte
Third Party Data	- bytes (= Packet Length)
Checksum	- 1 byte

The server always sends one packet at a time, thus nCount= 1. The server packets are of fixed length = 256 bytes. Unfilled bytes are padded with zero value.

ASetArtifactsCallbackFuncs

Description:
Registers callback function to inform detected artifacts. Five types of artifacts are detected and reported - Eyeblink, Excursion, Saturation, Spike, and EMG.

Format:
```
int  ASetArtifactsCallbackFuncs ( _stdcall* callbackEB, _stdcall* callbackExc,
_stdcall* callbackSat, _stdcall* callbackSpk, _stdcall* callbackEMG, int
nHandle);
```

Input Arguments:

1. Type: Pointer to Eye blink artifact callback function
   callbackEB(int epSt, int offSt, float shour, float smin, float ssec, float smillisec, int eep, int eoff, float ehour, float emin, float esec, float emillisec)
2. Type: Pointer to Excursion artifact callback function
3. callbackExc(int epSt, int offSt, float shour, float smin, float ssec, float smillisec, int eep, int eoff, float ehour, float emin, float esec, float emillisec)
4. Type: Pointer to Saturation artifact callback function
   callbackSat(int epSt, int offSt, float shour, float smin, float ssec, float smillisec, int eep, int eoff, float ehour, float emin, float esec, float emillisec)
5. Type: Pointer to Spike artifact callback function
   callbackSpk(int epSt, int offSt, float shour, float smin, float ssec, float smillisec, int eep, int eoff, float ehour, float emin, float esec, float emillisec)
6. Type: Pointer to EMG artifact callback function
   callbackEMG(int epSt, int offSt, float shour, float smin, float ssec, float smillisec, int eep, int eoff, float ehour, float emin, float esec, float emillisec)
7. nHandle – connection handle

Output Arguments:

1. Type: int
   1 = success, 0 = failed

PseudoCode:

Notes:

AgetNotification

Description:
Gets notifications such as battery status, missed blocks etc from the server.

Format:
```
float* AgetNotification ( int& Count, int handle);
```

Input Arguments:

3. nCount = 1 (always)
4. nHandle – connection handle

Output Arguments:

2. Type: Pointer to float
   Struncture: float FLAG1, float FLAG2, float INFO

PseudoCode:

Notes:

Many internal status notifications are reported using this function and useful values have to be filtered out based on the FLAG bytes.

Battery status : FLAG1 = 200, FLAG2 = 100, INFO = Battery %
Missed block status: FLAG1 = 200, FLAG2 = 101, INFO = Total number of missed blocks

AgetBattery

Description:
Gets battery information for last epoch sent from server. Information is sent by server per epoch, but pulling battery status information once per minute should be enough for all practical purposes.

Format:
```
int AgetBattery (int& nEpoch, int handle);
```

Input Arguments:
1. nEpoch – Epoch for which given battery information applies
2. nHandle – connection handle

Output Arguments:
3. Type: Battery voltage level in percentages as int value

PseudoCode:

Notes:

AgetMissedBlocks

Description:
Gets a total number of missed block in acquisition from server

Format:
```
int AgetMissedBlocks ( int& nEpoch, int handle);
```

Input Arguments:
1. nEpoch – Epoch for which given total missed blocks information applies
2. nHandle – connection handle

Output Arguments:
3. Return value is a total number of missed blocks from start of acquisition to the given epoch.

PseudoCode:
Notes:

5. B-Alert Server Programming Interface

5. B-Alert Server Programming Interface

5.1 Programming Notes 123
5.1.1 Interface 123
5.1.2 Configuration Steps 123
5.2. Validation 125
5.3 Application Programming Interface 125
StartServer 126
StopServer 127
StopServerTCP 127
StopServerUDP 128
SetEEGChannelMapInfo 129
GetStatusServerTCP 129
GetStatusServerUDP 130
SendTCPTimeStamp 131
SendUDPDatastreamingTS 131

SendRawData 132
SendDeconData 133
SendPSDData 133
SendPSDRawData 134
SendPSDBandwidthData 135
SendPSDBandwidthRawData 136
SendBandOverallPSDData 136
SendBandOverallPSDRawData 137
SendBrainState 138
SendQualityChannel 139
SendAccelerometer 139
SendMovement 140
SendAngles 141
SendZScore 141
SendEKGData 142
SendEBData 143
SendEXCData 143
SendSATData 144
SendSPKData 145
SendEMGData 145

5.1 Programming Notes

5.1.1 Interface

Third party programs may bypass B-Alert Control GUI module and serve as the link between B-Alert SDK and B-Alert Gauge GUI. This configuration will allow the third party application to control the data acquisition session transparently while using the features provided by the data presentation module and the SDK. Only one way communication (from Server to Client) is supported currently and the data presentation client module must function as a passive receiver in this configuration. Thus, functionalities such as impedance monitoring, technical monitoring etc cannot be invoked from the client side.

5.1.2 Configuration Steps

1. Interface B-Alert SDK – Develop interface to B-Alert SDK (refer B-Alert SDK Programming Interface section)

2. Start B-Alert TCP/IP server – The server can be started using APIs provided by ABM_ThirdPartyCommunication.dll. The APIs transmit data to B-Alert Gauge GUI (client) using ABM protocol over the TCP/IP network. The lib file can be found in SDK\lib folder and the header files in SDK\include\TCP folder.

3. Start B-Alert Gauge GUI – The Gauge GUI module can be started by invoking AthenaGUI.exe from ABM\EEG\ATheNA folder. The executable uses many of the files in the folder and thus it is HIGHLY RECOMMENDED that it is invoked from the same path. RomeSettings.xml file (ini file found in the same path) must be modified as follows:

```
<ipaddress> 127.0.0.1 </ipaddress>  -- change to the the ip address of the B-Alert server
<ipport> 4500 </ipport> -- change to the ip port specified during initialization of B-Alert server
<dataproducer> DataStreaming </dataproducer>  -- change from SDK to DataStreaming
<action> ViewRetransmit </action> -- change from Acquisition to ViewRetransmit
```

The executable can be invoked in two ways. 1) Using Windows command line (or Windows Batch file), 2) Using code. Examples are provided below

Sample script:

```
Athenagui.exe C:\ABM\EEG\ATheNA\RomeSettings.xml
```

Sample C++ code:

```
CWaitCursor wait;
_DEVICE_INFO info;
```

```
CString strPath = GetAppPath();
CString strRomeSettings = strPath + "\\RomeSettings.xml";
CFileFind find;
if (!find.FindFile(strRomeSettings))  {
        AfxMessageBox("Session settings file not found!\nPlease, try again.");
        return;
}
find.Close();
AfxGetApp()->DoWaitCursor(1);

strPath = GetAthenaAppPath();
CString strAthenaFullPath = strPath + "\\AthenaGUI.exe";

if (!find.FindFile(strAthenaFullPath))
{
        AfxGetApp()->DoWaitCursor(-1);
        AfxMessageBox("Athena executable not found.");
        return;
}
find.Close();
SetCurrentDirectory(strPath);
STARTUPINFO si;
PROCESS_INFORMATION pi;
ZeroMemory( &si, sizeof(si) );
si.cb = sizeof(si);
ZeroMemory( &pi, sizeof(pi) );

BOOL bRet = FALSE;
CString strCommandLine = "\"" + strAthenaFullPath + "\" \"" + strRomeSettings + "\"";

bRet = CreateProcess(NULL, (LPTSTR)(LPCTSTR)strCommandLine, NULL, NULL,
        FALSE, CREATE_NEW_CONSOLE|NORMAL_PRIORITY_CLASS, NULL, NULL, &si, &pi);

if (!bRet)
{
        AfxGetApp()->DoWaitCursor(-1);
        AfxMessageBox("Failed command line '" + strCommandLine + "'");
        return;
}

CloseHandle(pi.hProcess);
CloseHandle(pi.hThread);
ResumeThread(pi.hThread);
```

Important: Even though the SDK may send multiple packets for each API (depending on the interval between the calls), the B-Alert Server APIs can send only one packet at a time. In order to avoid missing of data, the third party program must either invoke the SDK APIs at a sampling rate higher than the data production rate (10 times higher is recommended) ensuring single packet delivery OR index the data pointer returned by the SDK appropriately and extract /send each packet individually. Note that the packet size depends on and varies with the data format returned by the SDK API; the elapsed time (EPOCH) is a good indicator to partition the packets. The sun

5.2. Validation

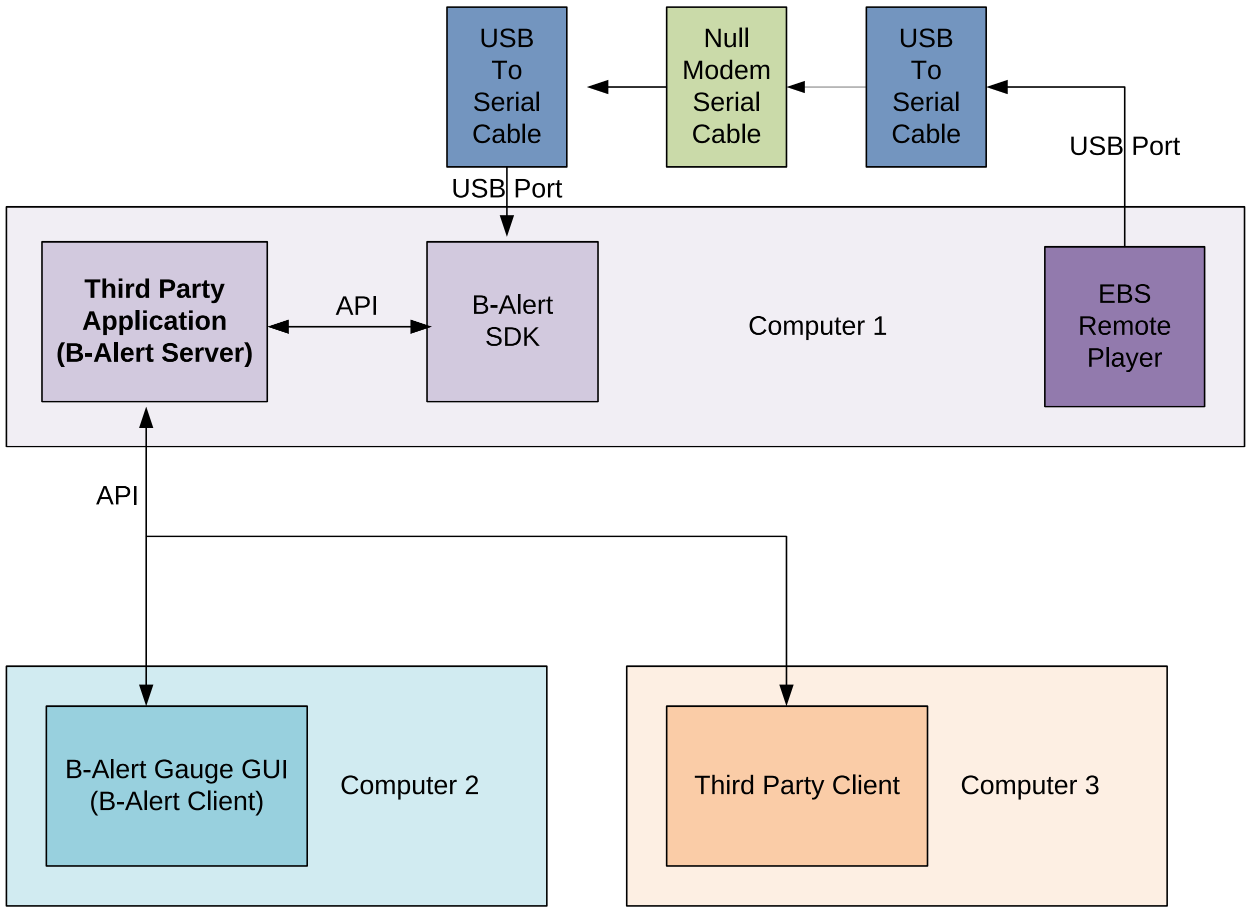


Figure 10: B-Alert Server Programming Interface validation setup

The validation procedure is identical to B-Alert SDK validation detailed in the earlier chapter. The output files generated by the above setup must the compared with the Gold Standard files (in ABM\EEG\Data folder) for similarity.

5.3 Application Programming Interface

StartServer

Description:
Starts B-Alert Control GUI server (TCP or UDP).

Format:
```
int  StartServer (_SERVER_INFO sInfo);
```

Input Arguments:
1. Type: Pointer to _SERVER_INFO

```
typedef struct _SERVER_INFO {
        unsigned int     port;          //server port to which the client
connects
        unsigned char     chProtocolTCPIP; // protocol (TCP=1, UDP=0)
        unsigned char     chDatagramProtocol; // Deafult = 1
        int    nDatastreaming_RawChannels;   //Number of channels (e.g., X10 =
10)
        int    nDatastreaming_EKGindex;        //Position of EKG Channel
(optional)
        int    nPSDBandsCount;          //number of psd bands (optional)
        int    nPSDBandsOverallCount; //number of psd bands (optional)
       char sessionID[ABM_3RD_SESSION_ID_LENGTH];  //session name
       int nDeconChannels;
       int nRawPSDChannels;
       int nClassPSDChannels;
       int nQualityCheckChannels;
}

(ABM_3RD_SESSION_ID_LENGTH = 9)
```

Output Arguments:
1. Type: int
   1 = success, 0 = failed

PseudoCode:

```
_SERVER_INFO server;
server.port = m_nPort; //TCP port number
strcpy(server.sessionID, strSessionID); //session name
server.chProtocolTCPIP = 1; // 1 for TCP
server.chDatagramProtocol = 1; // 1 for ABM data transfer protocol v1
server.nDatastreaming_RawChannels = 10 // number of EEG channels, i.e. 10 for x10 device
GetPSDCountBands( server.nPSDBands, server.nPSDBandsOverall ); // number of psd bands (optional)
server.nDatastreaming_EKGindex = devInfo.nECGPos; //index of ECG channel (optional)
if (StartServer(server) != OPERATION_PERFORMED_SUCCESSFULLY)
{
       AfxMessageBox("Server not started.", MB_OK|MB_ICONERROR);
       return;
}
```

Notes:

StopServer

Description:
Stops both TCP and UDP B-Alert Control GUI server.

Format:
```
int  StopServer ();
```

Input Arguments:

Output Arguments:
1. Type: int
   1 = success, 0 = failed

PseudoCode:

Notes:

If only TCP server was started, then it is recommend using StopServerTCP instead of StopServer.

StopServerTCP

Description:
Stops TCP B-Alert Control GUI server.

Format:
```
int  StopServerTCP ();
```

Input Arguments:

Output Arguments:

1. Type: int
   1 = success, 0 = failed

PseudoCode:

Notes:

The TCP server sends both data as-well-as timestamps to the client. B-Alert Gauge GUI works only with TCP server.

StopServerUDP

Description:
Stops UDP B-Alert Control GUI server.

Format:

```
int  StopServerUDP ();
```

Input Arguments:

Output Arguments:

1. Type: int
   1 = success, 0 = failed

PseudoCode:

Notes:

The UDP server is used to transmit timestamps to the client. The timestamps are usually used to synchronize between ABM and third party data. Note that B-Alert Gauge GUI will not work with UDP server.

SetEEGChannelMapInfo

Description:
Sets EEG channel Map info to make it available for connecting clients.

Format:

```
int  SetEEGChannelMapInfo (char* chMapInfoBytes, int nLength);
```

Input Arguments:

1. chMapInfoBytes – char array filled with the following structure

```
typedef struct _EEGCHANNELS_INFO {
        char    cChName[24][20];              //names of available channels
        bool    bChUsed[24];                     //1=channel available,
0=channel absent
        bool    bChUsedQualityData[24] ; //1=represented in AgetQualityChannel,
0=absent              }
```

2. nLength = sizeof(chMapInfoBytes)

Output Arguments:

1. Type: int
   1 = success, 0 = failed

PseudoCode:

Notes:

GetStatusServerTCP

Description:
Gets number of clients connected to the TCP server and TCP server status.

Format:

```
int  GetStatusServerTCP (unsigned int &nClients);
```

Input Arguments:

3. nClients – number of clients connected to the TCP server

Output Arguments:

4. Type: int
   1 = server running, 0 = server stopped

PseudoCode:

Notes:

B-Alert Client-Server architecture supports simultaneous connections between multiple clients and servers.

GetStatusServerUDP

Description:
Gets number of clients connected to the UDP server and UDP server status.

Format:

```
int  GetStatusServerUDP (unsigned int &nClients);
```

Input Arguments:

1. nClients – number of clients connected to the UDP server

Output Arguments:

2. Type: int
   1 = server running, 0 = server stopped

PseudoCode:

Notes:

SendTCPTimeStamp

Description:
Sends elapsed time for each sample received from the SDK (TCP Protocol)

Format:

```
int  SendTCPTimeStamp (unsigned int nEpoch, unsigned int nOffset, unsigned int nHour, unsigned int nMinute, unsigned int nSecond, unsigned int nMilliSecond );
```

Input Arguments:

1. nEpoch 2. nOffset 3. nHour, 4. nMinute, 5. nSecond, 6. nMilliSecond

Output Arguments:

1. Type: int
   1 = success, 0 = failed

PseudoCode:

Notes:

SendUDPDatastreamingTS

Description:
Sends elapsed time for each sample received from the SDK (UDP Protocol)

Format:

```
int  SendUDPDatastreamingTS (unsigned int nEpoch, unsigned int nOffset, unsigned int nHour, unsigned int nMinute, unsigned int nSecond, unsigned int nMilliSecond );
```

Input Arguments:

1. nEpoch 2. nOffset 3. nHour, 4. nMinute, 5. nSecond, 6. nMilliSecond

Output Arguments:

1. Type: int
   1 = success, 0 = failed

PseudoCode:

Notes:

SendRawData

Description:
Sends raw data to the client.

Format:
```
int  SendRawData(float* pData, int nLength);
```

Input Arguments:

1. pData – pointer to data
2. nLength – packet size = NumberOfChannels + TIMESTAMP_SIZE_RAW where, NumberOfChannels = 10 for X10, and TIMESTAMP_SIZE_RAW = 6 for all devices

Output Arguments:

1. Type: int
   1 = success, 0 = failed

PseudoCode:

```
int nCount = -1;
float* pRawData = GetFilteredData(nCount);
int iRet;
for (int i=0; i<nCount; i++)
{
    int rawDataPackageSize = m_nNumberofChannels + TIMESTAMP_SIZE_RAW; /*timestamp size = 6*/
    iRet = SendRawData(pRawData + i*rawDataPackageSize, rawDataPackageSize);
}
```

Notes:

Even though the SDK may send multiple packets (nCount >1) depending on the interval between the calls, the B-Alert server can send only one packet at a time.

SendDeconData

Description:
Sends artifact decontaminated data to the client.

Format:
```
int  SendDeconData(float* pData, int nLength);
```

Input Arguments:
1. pData – pointer to data
2. nLength – packet size = NumberOfChannels + TIMESTAMP_SIZE_RAW where, NumberOfChannels = 10 for X10, and TIMESTAMP_SIZE_RAW = 6 for all devices

Output Arguments:
2. Type: int
   1 = success, 0 = failed

PseudoCode:

Notes:

Even though the SDK may send multiple packets (nCount >1) depending on the interval between the calls, the B-Alert server can send only one packet at a time.

SendPSDData

Description:
Sends Power Spectral Densities of PROCESSED data to the client.

Format:
```
int  SendPSDData(float* pData, int nLength);
```

Input Arguments:
1. pData – pointer to data
2. nLength – packet size = NumberOfChannels + TIMESTAMP_WO_OFFSET where, NumberOfChannels = 10 for X10, and TIMESTAMP_WO_OFFSET = 5 for all devices

Output Arguments:

1. Type: int
   1 = success, 0 = failed

PseudoCode:

```
int nEpoch = -1, nSize = -1;
int TIMESTAMP_WO_OFFSET=5;
float* pPSDData = GetPSDData(nEpoch);
int iRet;
for (int i=0; i<nEpoch; i++)
{
  int packetSize = MAX_NUM_CHANNELS4SDK * 128 + TIMESTAMP_WO_OFFSET;
  iRet = SendPSDData(pPSDData + i* packetSize, packetSize);
}
```

Notes:

Even though the SDK may send multiple packets (nEpochs >1) depending on the interval between the calls, the B-Alert server can send only one packet at a time.

MAX_NUM_CHANNELS4SDK is the number of EEG channels (For, example X10=9, X24=20, X4=3)

SendPSDRawData

Description:
Sends Power Spectral Densities of RAW data to the client.

Format:
`int  SendPSDRawData(float* pData, int nLength);`

Input Arguments:

1. pData – pointer to data
2. nLength – packet size = NumberOfChannels + TIMESTAMP_WO_OFFSET where, NumberOfChannels = 10 for X10, and TIMESTAMP_WO_OFFSET = 5 for all devices

Output Arguments:

1. Type: int
   1 = success, 0 = failed

PseudoCode:

Notes:

Even though the SDK may send multiple packets (nEpochs >1) depending on the interval between the calls, the B-Alert server can send only one packet at a time.

MAX_NUM_CHANNELS4SDK will be automatically filled.

SendPSDBandwidthData

Description:
Sends Power Spectral Densities (PSD) of specified EEG bands calculated from PROCESSED data.

Format:
`int  SendPSDBandwidthData(float* pData, int nLength);`

Input Arguments:

1. pData – pointer to data
2. nLength – updated by the SDK with the number of data bytes in each epoch

Output Arguments:

1. Type: int
   1 = success, 0 = failed

PseudoCode:

```
nEpoch = -1;
nSize = -1;
float* pPSDBandwidthData = GetPSDBandwidthData(nEpoch, nSize);
if (nEpoch > 0)
        iRet = SendPSDBandwidthData(pPSDBandwidthData, nSize);
```

Notes:

Even though the SDK may send multiple packets (nEpochs >1) depending on the interval between the calls, the B-Alert server can send only one packet at a time.

SendPSDBandwidthRawData

Description:
Sends Power Spectral Densities (PSD) of specified EEG bands calculated from RAW data.

Format:
```
int  SendPSDBandwidthRawData(float* pData, int nLength);
```

Input Arguments:
1. pData – pointer to data
2. nLength – updated by the SDK with the number of data bytes in each epoch

Output Arguments:
1. Type: int
   1 = success, 0 = failed

PseudoCode:

Notes:

Even though the SDK may send multiple packets (nEpochs >1) depending on the interval between the calls, the B-Alert server can send only one packet at a time.

SendBandOverallPSDData

Description:
Sends Power Spectral Densities (PSD) computed on PROCESSED EEG data for specified bands averaged across all EEG channels.

Format:
```
int  SendBandOverallPSDData(float* pData, int nLength);
```

Input Arguments:
1. pData – pointer to data
2. nLength – updated by the SDK with the number of data bytes in each epoch

Output Arguments:
1. Type: int

1 = success, 0 = failed

PseudoCode:

```
int nEpoch = -1, nSize = -1;
float* pBandOverallPSDData = GetBandOverallPSDData(nEpoch, nSize);
int iRet;
if (nEpoch > 0)
        iRet = SendBandOverallPSDData(pBandOverallPSDData, nSize);
```

Notes:

Even though the SDK may send multiple packets (nEpochs >1) depending on the interval between the calls, the B-Alert server can send only one packet at a time.

SendBandOverallPSDRawData

Description:
Sends Power Spectral Densities (PSD) computed on RAW EEG data for specified bands averaged across all EEG channels.

Format:
`int  SendBandOverallPSDRawData(float* pData, int nLength);`

Input Arguments:

1. pData – pointer to data
3. nLength – updated from SDK with the number of data bytes in each epoch

Output Arguments:

1. Type: int
   1 = success, 0 = failed

PseudoCode:

Notes:

Even though the SDK may send multiple packets (nEpochs >1) depending on the interval between the calls, the B-Alert server can send only one packet at a time.

SendBrainState

Description:
Sends B-Alert engagement and workload classifications

Format:
```
int  SendBrainState(float* pData, int nLength);
```

Input Arguments:
2. pData – pointer to data
4. nLength – packetsize = 13

Output Arguments:
2. Type: int
   1 = success, 0 = failed

PseudoCode:

```
int nEpoch = -1;
_BRAIN_STATE* pBrainState = GetBrainState(nEpoch);
int iRet;
for (int i=0; i<nEpoch; i++)
{
        float pfBrainState [13];
        pfBrainState[0] =  (pBrainState+i)->fEpoch;
        pfBrainState[1] =  (pBrainState+i)->fABMSDKTimeStampHour;
        pfBrainState[2] =  (pBrainState+i)->fABMSDKTimeStampMinute;
        pfBrainState[3] =  (pBrainState+i)->fABMSDKTimeStampSecond;
        pfBrainState[4] =  (pBrainState+i)->fABMSDKTimeStampMilsecond;
        pfBrainState[5] =  (pBrainState+i)->fClassificationEstimate;
        pfBrainState[6] =  (pBrainState+i)->fHighEngagementEstimate;
        pfBrainState[7] =  (pBrainState+i)->fLowEngagementEstimate;
        pfBrainState[8] =  (pBrainState+i)->fDistractionEstimate;
        pfBrainState[9] =  (pBrainState+i)->fDrowsyEstimate;
        pfBrainState[10] =  (pBrainState+i)->fWorkloadFBDSEstimate;
        pfBrainState[11] =  (pBrainState+i)->fWorkloadBDSEstimate;
        pfBrainState[12] =  (pBrainState+i)->fWorkloadAverageEstimate;
        iRet = SendBrainState(pfBrainState, 13);
}
```

Notes:

Even though the SDK may send multiple packets (nEpochs >1) depending on the interval between the calls, the B-Alert server can send only one packet at a time.

SendQualityChannel

Description:
Sends % of 1)overall data quality across all channels and 2) data quality (EMG & Other artifacts) for each individual channel.

Format:
```
int  SendQualityChannel(float* pData, int nLength);
```

Input Arguments:
1. pData – pointer to data
2. nLength – packetsize = 6+NumRawChannels*2, where NumRawChannels= Num of EEG channels, for example X10 has 9 EEG channels

Output Arguments:
1. Type: int
   1 = success, 0 = failed

PseudoCode:

Notes:

Even though the SDK may send multiple packets (nEpochs >1) depending on the interval between the calls, the B-Alert server can send only one packet at a time.

SendAccelerometer

Description:
Sends 3-axis accelerometer data received from ABM devices in raw format.

Format:
```
int  SendAccelerometer(float* pData, int nLength);
```

Input Arguments:
1. pData – pointer to data
2. nLength – packetsize = 9 (epoch, offset, 4xtimestamp, 3 x axes)

Output Arguments:

1. Type: int
   1 = success, 0 = failed

PseudoCode:

Notes:

Even though the SDK may send multiple packets (nEpochs >1) depending on the interval between the calls, the B-Alert server can send only one packet at a time.

SendMovement

Description:
Sends movement-value and movement-level computed from the output of the 3-axis accelerometer integrated in the headset.

Format:
```
int  SendMovement(float* pData, int nLength);
```

Input Arguments:

1. pData – pointer to data
2. nLength – packetsize = 7

Output Arguments:

1. Type: int
   1 = success, 0 = failed

PseudoCode:

Notes:

Even though the SDK may send multiple packets (nEpochs >1) depending on the interval between the calls, the B-Alert server can send only one packet at a time.

SendAngles

Description:
Sends 3-axis accelerometer data received from ABM devices in angles format.

Format:
```
int  SendAngles(float* pData, int nLength);
```

Input Arguments:
1. pData – pointer to data
2. nLength – packet size = 9

Output Arguments:
1. Type: int
   1 = success, 0 = failed

PseudoCode:

Notes:

Even though the SDK may send multiple packets (nEpochs >1) depending on the interval between the calls, the B-Alert server can send only one packet at a time.

SendZScore

Description:
Sends z-score values (initialized in SDK).

Format:
```
int  SendZScore(float* pData, int nLength);
```

Input Arguments:
1. pData – pointer to data
5. nLength – packet size = updated from SDK with the number of data bytes in each epoch

Output Arguments:
1. Type: int

1 = success, 0 = failed

PseudoCode:

Notes:

Even though the SDK may send multiple packets (nEpochs >1) depending on the interval between the calls, the B-Alert server can send only one packet at a time.

SendEKGData

Description:
Sends heart rate values

Format:
```
int  SendEKGData(float* pData, int nLength);
```

Input Arguments:
1. pData – pointer to data
2. nLength - 10

Output Arguments:
1. Type: int
   1 = success, 0 = failed

PseudoCode:

Notes:

Even though the SDK may send multiple packets (nEpochs >1) depending on the interval between the calls, the B-Alert server can send only one packet at a time.

SendEBData

Description:
Sends eye blink artifact detection markers

Format:
```
int  SendEBData(char* pData, int nLength);
```

Input Arguments:
1. pData – pointer to data
2. nLength – packet size = 12

Output Arguments:

2. Type: int
   1 = success, 0 = failed

PseudoCode:

Notes:

Even though the SDK may send multiple packets (nEpochs >1) depending on the interval between the calls, the B-Alert server can send only one packet at a time.

SendEXCData

Description:
Sends excursion artifact detection markers

Format:
```
int  SendEXCData(char* pData, int nLength);
```

Input Arguments:
1. pData – pointer to data
2. nLength – packet size = 13

Output Arguments:
1. Type: int
   1 = success, 0 = failed

PseudoCode:

Notes:

Even though the SDK may send multiple packets (nEpochs >1) depending on the interval between the calls, the B-Alert server can send only one packet at a time.

SendSATData

Description:
Sends saturation artifact detection markers

Format:
```
int  SendSATData(char* pData, int nLength);
```

Input Arguments:
1. pData – pointer to data
2. nLength – packet size = 13

Output Arguments:
1. Type: int
   1 = success, 0 = failed

PseudoCode:

Notes:

Even though the SDK may send multiple packets (nEpochs >1) depending on the interval between the calls, the B-Alert server can send only one packet at a time.

SendSPKData

Description:
Sends spike artifact detection markers

Format:
```
int  SendSPKData(char* pData, int nLength);
```

Input Arguments:
1. pData – pointer to data
2. nLength – packet size = 13

Output Arguments:
1. Type: int
   1 = success, 0 = failed

PseudoCode:

Notes:

Even though the SDK may send multiple packets (nEpochs >1) depending on the interval between the calls, the B-Alert server can send only one packet at a time.

SendEMGData

Description:
Sends EMG artifact detection markers

Format:
```
int  SendEMGData(char* pData, int nLength);
```

Input Arguments:
2. pData – pointer to data
3. nLength – packet size = 13

Output Arguments:
2. Type: int
   1 = success, 0 = failed

PseudoCode:

Notes:

Even though the SDK may send multiple packets (nEpochs >1) depending on the interval between the calls, the B-Alert server can send only one packet at a time.

SendBatteryPercentage

Description:
Sends information about battery level

Format:
```
int  SendBatteryPercentage(int nBatteryLevel, int nEpoch);
```

Input Arguments:
1. nBatteryLevel – percentage of battery voltage level
2. nEpoch – Epoch to which battery information is related

Output Arguments:
1. Type: int
   1 = success, 0 = failed

PseudoCode:

Notes:

SendMissedBlocks

Description:
Sends a total number of missed blocks during current acquisition session

Format:
```
int  SendMissedBlocks(int nTotalMissedBlocks, int nEpoch);
```

Input Arguments:
1. nTotalMissedBlocks – total number of missed blocks
2. nEpoch – Epoch to which battery information is related

Output Arguments:
1. Type: int
   1 = success, 0 = failed

PseudoCode:

Notes:

6. B-Alert Maintenance Programming Interface

6. B-Alert Maintenance Programming Interface

ConfigUploadFirmware 149
ConfigSearchBTDevices 150
ConfigSynchWithDevice 151

6.1 Config Upload Firmware

ConfigUploadFirmware

Description:
Upload firmware to ABM device.

Format:
```
int  ConfigUploadFirmware (unsigned char* ucFpath);
```

Input Arguments:

1. Type: Pointer to unsigned char
   ucFpath: path to firmware file

Output Arguments:

1. Type: int
   1 = success, 0 = failed

PseudoCode:
```
CString m_strFirmwareFile;
m_strFirmwareFile = GetFirmwarePathName();
if (m_strFirmwareFile.IsEmpty())
{
      AfxMessageBox("Firmware file path is empty!");
      return;
}
nSuccess = ConfigUploadFirmware((unsigned char*)m_strFirmwareFile.GetBuffer(0));
m_strFirmwareFile.ReleaseBuffer();
AfxMessageBox(GetDeviceConfigErrorMessage(nSuccess));
```

Notes:

Uploading invalid firmware to ABM devices could cause permanent damage to ABM devices and is considered a breach of warranty.

This function should be used only by authorized technician.

6.2 Config Search BT Devices

ConfigSearchBTDevices

Description:
Returns information about all devices within the range of the Bluetooth dongle.

Format:
```
int  ConfigSearchBTDevices (unsigned char devN[], unsigned char btN[ ], int devNLen[], int& nNumOfDev);
```

Input Arguments:
1. Type: Pointer to unsigned char array
   devN: names of identified devices
2. Type: Pointer to unsigned char array
   btN: Bluetooth identification of devices
3. Type: int array
   devNLen: device name lengths
4. Type: int
   nNumOfDev: number of devices

Output Arguments:
1. Type: int
   1 = success, 0 = failed

PseudoCode:
```
unsigned char devN[1024], btN[1024];
int devNLen[1024];
int nNumOfDev = 0;
memset(devN,0, sizeof(unsigned char)*1024);
memset(btN,0, sizeof(unsigned char)*1024);
memset(devNLen,0, sizeof(int)*1024);
int nSuccess = ConfigSearchBTDevices(devN, btN, devNLen, NumOfDev);
if (nSuccess == DEVICE_SEARCH_BTDEVICES_SUCCESS) {
     for(int i = 0; i < nNumOfDevices; i++)
          AddDeviceToArray(deviceNames, btNum, nDeviceNameLength, nNumOfDevices);
}
else
     AfxMessageBox(GetDeviceConfigErrorMessage(nSuccess));
```

Notes:
This function should be used only by authorized technician.

6.3 Config Synch With Device

ConfigSynchWithDevice

Description:
Syncs ABM device with the Bluetooth dongle.

Format:
```
int  ConfigSynchWithDevice (unsigned char * btN, BOOL useVer2ForX4);
```

Input Arguments:
1. Type: Unsigned char array
   btN: Bluetooth identification of the device (obtained via ConfigSearchBTDevices)
2. Type: Boolean
   Whether to use protocol v2 for X4

Output Arguments:
4. Type: int
   1 = success, 0 = failed

PseudoCode:
```
int nSuccess = ConfigSynchWithDevice(btNum);
if (nSuccess != DEVICE_CONNECT_SUCCESS)
      AfxMessageBox(GetDeviceConfigErrorMessage(nSuccess));
else
      AfxMessageBox(GetDeviceConfigErrorMessage(nSuccess),MB_ICONINFORMATION|MB_OK);
```

Notes:

This function should be used only by authorized technician.

7. ABM Multi-Channel External Sync Unit (MC-ESU) Programming Interface

7. ABM Multi-Channel External Sync Unit (MC-ESU) Programming Interface

7.1 Interface 154
7.2 Using the MC-ESU 157
7.2.1. Configuring the MC-ESU: 157
7.2.2. Sending data to the MC-ESU: 158
7.2.3. Decoding data from the MC-ESU: 159

7.1 Interface

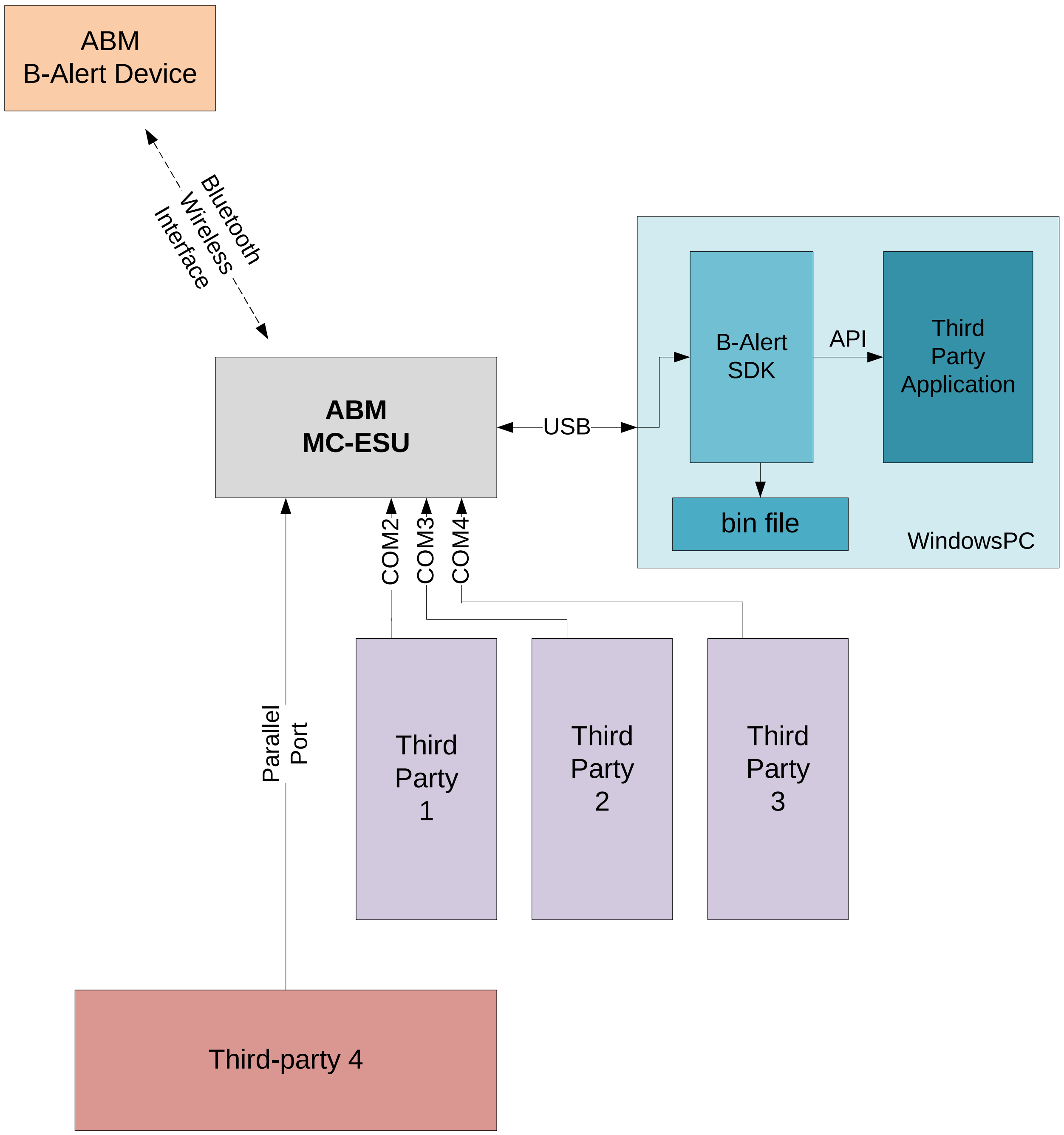


Figure 11: Block diagram of data acquisition using MC-ESU

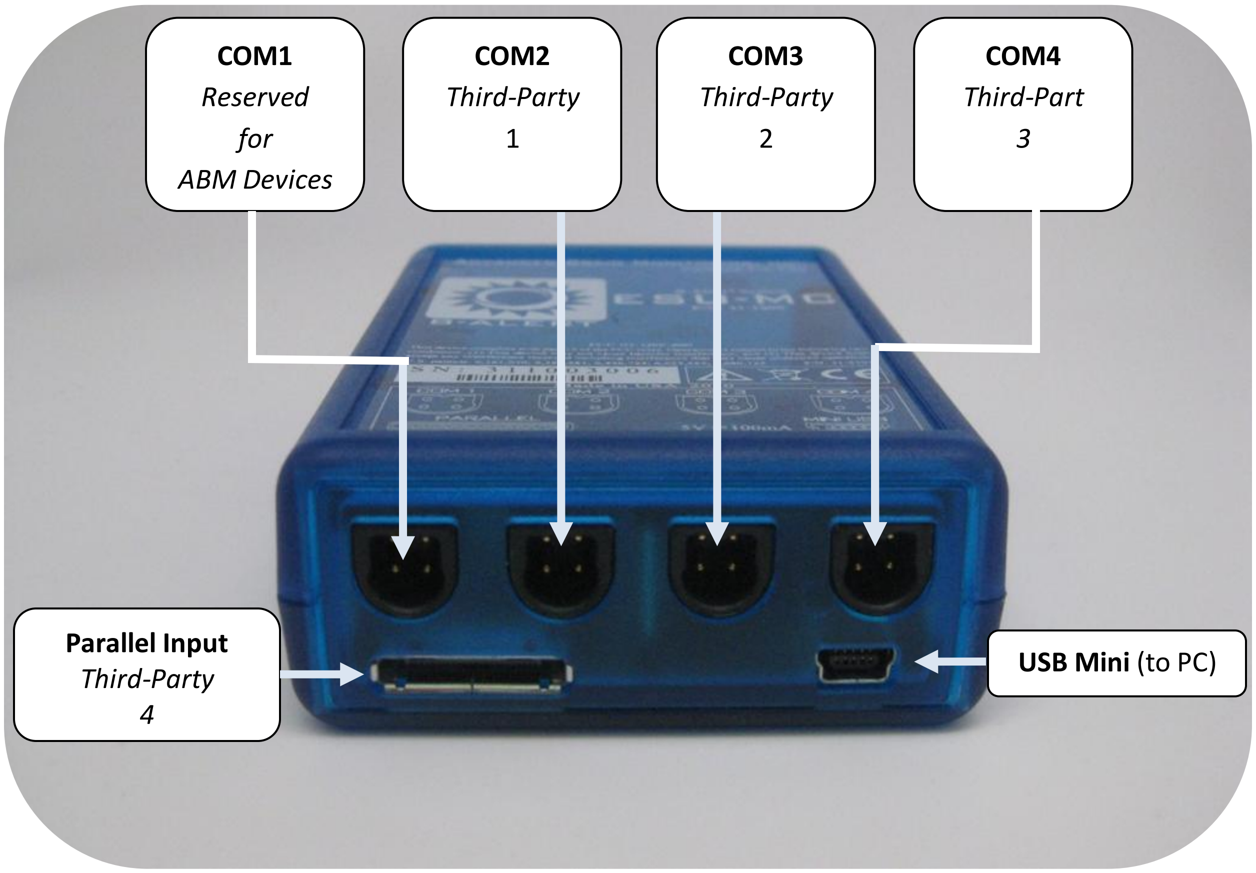


Figure 12: Ports in MC-ESU

The ABM Multi-Channel External Sync Unit (MC-ESU) can be used to timestamp data from ABM devices as-well-as events from third party applications at millisecond level precision through a dedicated robust hardware timer. Timestamping using Windows timers is at the mercy of the Windows scheduler and an average variable latency of ~30-50 millisec is unavoidable in most circumstances. The variable latency can cause havoc in EEG signal analysis, especially in studies that rely on synchronization with external stimuli in order to extract event-related-synchronization/desynchronization. It also reduces the Signal-To-Noise-Ratio (SNR) while averaging the EEG samples to extract features such as evoked potentials. The MC-ESU thus helps in reducing the variable delay by timestamping the data packets external to the windows environment.

The MC-ESU functions as a Bluetooth receiver for ABM devices as-well and it has the intelligence to decode ABM protocol and stream the data to the SDK via the USB port (the USB port registers a Virtual COM port in the PC). ABM X10 and X4 devices bundles two samples from all the channels into a single packet that gets timestamped every 8 millisecond, while the X24 sends only one sample and is timestamped every 4 millisecond. COM1 is reserved for ABM devices (wired configuration), while COM 2,3&4 can be used to acquire third-party events via RS232 serial-port protocol. Applications can also send events via the parallel port either with or without the STROBE signal. All third party events will be timestamped without delay in the MC-ESU however in case of conflict, EEG packets will have priority over other data. Proprietary cables supplied with the MC-ESU (see Figure 13) must be used to send data. All third-party applications must follow ABM protocol (specified in this chapter) while sending data to the MC-ESU. The SDK will unpack the third-party events and store them in a bin file along with ebs/edf data files for offline analysis. The third party events can also be acquired in real-time using SDK APIs.

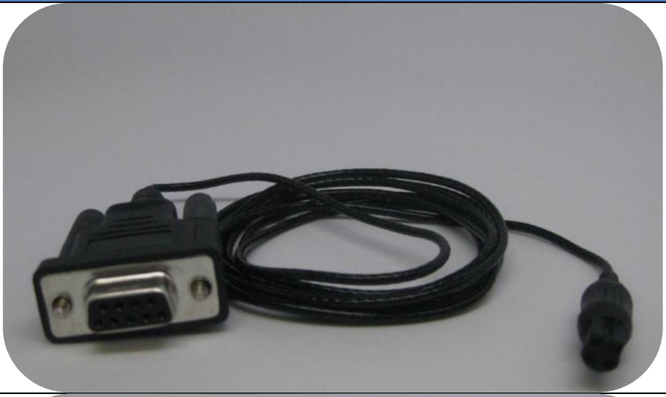
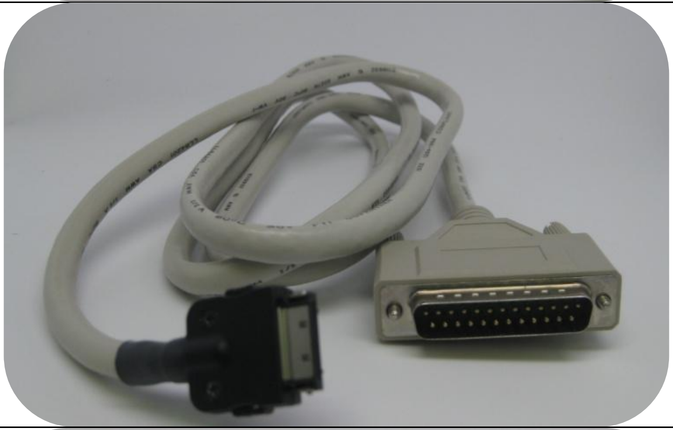
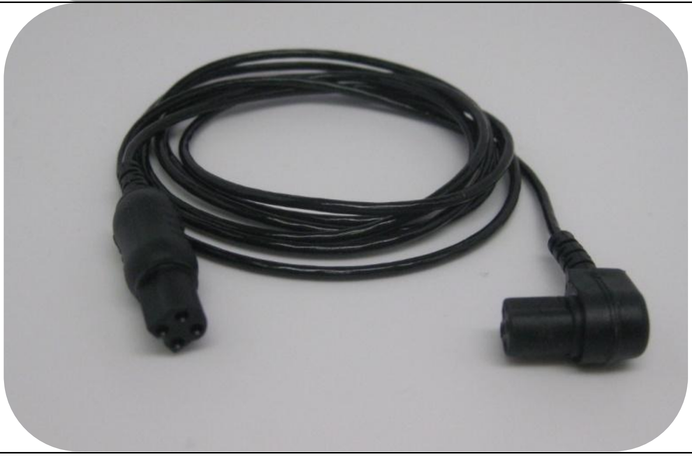
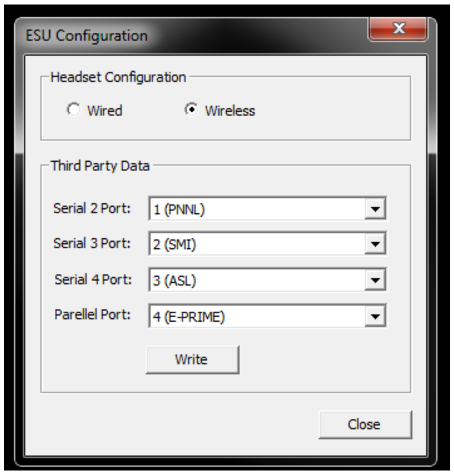
Cable Name	Photo
Serial Port to ESU cable	
Parallel port to ESU Cable	
4-pin to 4-pin serial (for **wired** headset data collection only)	

Figure 13: Proprietary cables for MC-ESU ports

7.2 Using the MC-ESU

7.2.1. Configuring the MC-ESU:

The MC-ESU can be configured using B-Alert Control GUI by invoking the dialog interface from Operations\Configure ESU menu. Headset configuration must be selected depending on wired or wireless configuration. The wireless configuration suffers from a variable delay of ~ 8-12 msec due to the Bluetooth protocol; this can be eliminated using the wired protocol. Please note that ABM devices work quite well in the wireless mode and the variable delay is not an issue in most cases and it provides the added convenience of complete mobility.



Headset Configuration: Wired [✓ Wireless]

Third Party Data

Serial 2 Port: 1 (PNNL)
Serial 3 Port: 2 (SMI)
Serial 4 Port: 3 (ASL)
Parellel Port: 4 (E-PRIME)

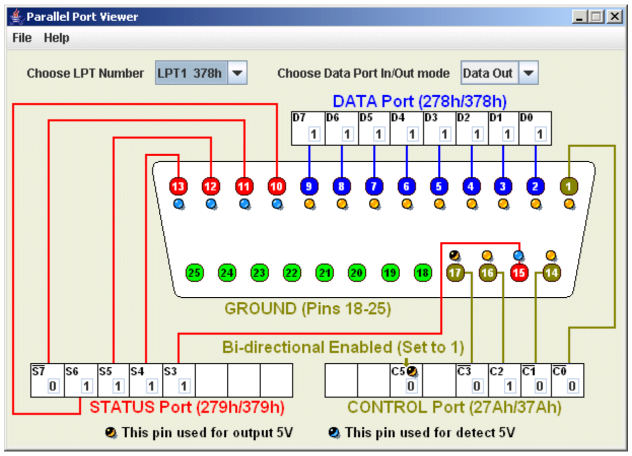
Figure 14: Configure ESU interface in B-Alert Software

The MC-ESU supports multiple protocols for acquiring third party data. The specifications of the protocols are listed below:

Serial communication protocols				
Type	**Baud Rate (kb/s)**	**Data bits**	**Stop bit**	**Parity**
PNNL	57600	8	1	None
SMI	9600	8	1	None
ASL	19200	8	1	None
AMP	Reserved	8	1	None

Parallel communication protocols		
Type	**STROBE (YES/NO)**	**Packet Structure (YES/NO)**
PNNL	Yes	Yes
ANITA	Yes	Yes
DAIMLER	No	No*
E-PRIME	No	No*

*For protocols without a packet structure, the changes in the 8-bit parallel lines are time-stamped. Idle values (0) must be used in between valid data and the values must be held for at-least 100 usec for registration.



Choose LPT Number: LPT1 378h
Choose Data Port In/Out mode: Data Out

Figure 15: MC-ESU parallel port pin configuration (STROBE – pin[1], DATA – pins[2:9]

7.2.2. Sending data to the MC-ESU:

The third party applications must send packets according to the ABM Universal Protocol to the serial and parallel (with STROBE protocols) ports of the MC-ESU. The packet follows BIG ENDIAN format and has the following fields:

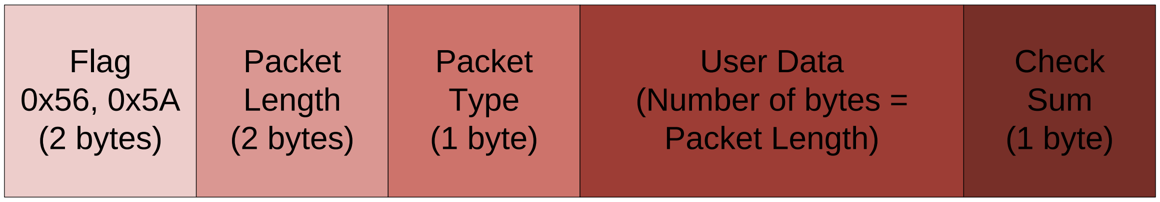

Flag 0x56, 0x5A (2 bytes)
Packet Length (2 bytes)
Packet Type (1 byte)
User Data (Number of bytes = Packet Length)
Check Sum (1 byte)

Figure 16: ABM Universal Protocol (packet structure for third party data).

- Flag: fixed (0x56,0x5A)
- Packet Length: packet length is equal to the number of bytes in the User Data field

- Packet Type: depends on the type of protocol
  - PNNL (Serial = 1, Parallel = 2)
  - SMI (Serial = 3)
  - ASL (Parallel = 4)
  - AMP (Parallel = 8)
  - ANITA (Parallel = 6)
  - DAIMLER (Parallel = 7)
  - EPRIME (Parallel = 10)
- User Data:
- Checksum = 255 - Mod(SumOfAllBytes(Packet Length, PTY, DATA), 256)

Sample programs: Programs for reference that mimic sending data through the serial port and the parallel port can be found in ABM/EEG/Utilities/ESU_TParty folder.

7.2.3. Decoding data from the MC-ESU:

Offline acquisition: B-Alert SDK unpacks the data received from the MC-ESU and stores it in a bin-file. The bin-file has the following packet structure for each event:

Flag 0x56, 0x56 (2 bytes)	Message Counter (1 byte)	ESU Timestamp (4 bytes)	Packet Length (2 bytes)	Packet Type (1 byte)	User Data (Number of bytes = Packet Length)	Check Sum (1 byte)

Figure 17: Structure of third party packets stored in bin file by B-Alert SDK

- Flag: fixed (0x56,0x56)
- Message Counter: (reserved)
- ESU Timestamp: 4-bytes ESU timestamp in BIG ENDIAN format (the timestamp can be reconstructed using multiples of 2 (for example, timestamp in millisec = BYTE1*2^24 + BYTE2*2^16 + BYTE3*2^8 + BYTE4*2^0)
- Packet Length: equal to the number of bytes in User Data field
- Packet Type: protocol used
- User Data
- Checksum: Mod(SumOfAllBytes(Packet Length, PTY, DATA), 256)

Real-time acquisition: Applications can also acquire third-party data in real-time using SDK API GetThirdPartyData (refer B-Alert SDK Programming Interface).

Revision History:

Date	Description
05/9/11	Updated X24, GetFilteredData,
05/10/11	Programming note-4, GetTimeStampsStreamData.
05/20/11	Added RegisterCallbackMissedBlocks
07/18/11	Changed GetQualityChannelData (V0.06)
08/19/11	Added Matlab client APIs
09/21/11	Added B-Alert Client Programming Interface
09/22/11	Added B-Alert Server Programming Interface
09/22/11	Added ABM MC-ESU Programming Interface
10/06/11	Add new APIs GetPacketChannelNmbInfo, GetChannelMapInfo, SetEEGChannelMapInfo, AgetChannelMapInfo
08/15/12	Change device code for X24_Standard to 3, edited matlab setup instructions.
09/18/12	Updated signatures of C++ functions
07/03/13	Changed cover page
07/08/13	Changed name to "B-Alert"
09/18/13	Changed DeviceType codes for InitSession
10/3/13	Added GetClassMu, GetClassMidline,GetClassPrefrontal