Amp Server™ Pro SDK 2.1

Network Protocols, Data Formats, and Technical Details

Programmer's Reference Manual

8409502-56 (20151214)

WARNING: All EGI system components must be installed and configured by an EGI support or authorized engineer. Deviating from the supported configuration or running the system with non-EGI-approved components attached can cause hazards or unexpected performance.

Amp Server Pro SDK is a software development kit (SDK) library that allows users to capture data in real time from EGI amplifiers into non-EGI applications. Amp Server Pro SDK provides two methods for communicating with Amp Server:

1. Using a high-level API, which is a client-side protocol that is usable with languages that are linkable to C++ via its ready-to-use client libraries.

   Note: The high-level (client-side protocol) API includes library support for Linux and Mac OS. These libraries have been tested under Mac OS 10.10, Ubuntu 15.04, and Ubuntu 15.10.

2. Using a network protocol that is language independent.

CAUTION: Anyone who is not completely proficient with all aspects of customizing their EEG system should not attempt it. Poorly programmed applications could cause processes to fail or corrupt EEG data.

This document itemizes the commands that are the basis of the network protocols.

Amplifier and Amp Server Commands

Most commands apply to all EGI Net Amps amplifiers. Exceptions are highlighted and noted.

Command	Arguments	Description
Amplifier		
cmd_None	int64_t amp id, int16_t channel (N/A), int16_t value (N/A)	This is a null command. It has no function and is not used to command the amplifier.
cmd_Start	int64_t amp id, int16_t channel (N/A), int16_t value (N/A)	This command causes the amplifier to begin to send data. The cmd_SetPower command should be used to turn on the amplifier before trying to issue the start command.
cmd_Stop	int64_t amp id, int16_t channel (N/A), int16_t value (N/A)	This command tells the amplifier to stop sending data.
cmd_TurnAll10KOhms	int64_t amp id, int16_t channel (N/A), int16_t value (0 = Off, 1 = On)	This command instructs the amplifier to place or remove 10 KOhm resistors between the inputs of all channels and ground. This state is used to measure impedance and noise.
cmd_TurnChannel10KOhms	int64_t amp id, int16_t channel (channels 0-287), int16_t value (0 = Off, 1 = On)	This command instructs the amplifier to place or remove 10 KOhm resistors between the inputs of a selected channel and ground. This state is used to measure impedance and noise.
cmd_setCOM10KOhms ***NA 400s only***	int64_t amp id, int16_t channel (N/A), int16_t value (N/A)	Like the cmd_TurnChannel10KOhms command, this command places a 10 KOhm resistor on the input to the COM channel. This can be used to measure impedance and noise of the COM channel.
cmd_TurnAllDriveSignals	int64_t amp id, int16_t channel (N/A), int16_t value (0 = Off, 1 = On)	This command instructs the amplifier to drive all channels with a test or calibration signal. The signal drives the inputs to the amplifier and provides a test signal that can be used in the absence of an EEG Net to test inputs to the amplifier. This command is used to measure impedance and gains.
cmd_SetCOMDriveSignal ***NA 400s only***	int64_t amp id, int16_t channel (N/A), int16_t value (N/A)	Like the cmd_TurnAllDriveSignals command, this command turns on the calibrations signal on the input to the COM channel. This can be used for debugging and diagnostics for measuring impedance and gains.
cmd_TurnChannelDriveSignals	int64_t amp id, int16_t channel (channels 0-287), int16_t value (0 = Off, 1 = On)	This command instructs the amplifier to drive a selected channel with a test or calibration signal. The signal drives the inputs to the amplifier and provides a test signal that can be used in the absence of an EEG Net to test inputs to the amplifier. This command is used to measure impedance and gains.
cmd_SetSubjectGround	int64_t amp id, int16_t channel (N/A), int16_t state (0 = Off, 1 = On)	This command turns the common channel On and Off and is used only for impedance measurements.
cmd_SetCurrentSource	int64_t amp id, int16_t channel (N/A), int16_t value	This command switches the calibration signal between constant voltage and constant current. The default is constant voltage.
cmd_SetCalibrationSignalFreq	int64_t amp id, int16_t channel (N/A), int16_t value (frequency)	This command sets the frequency of the calibration signal that is enabled when the *cmd_TurnChannelDriveSignals* and *cmd_SetWaveShape* commands are used.

Command	Arguments	Description
cmd_SetBufferedReference	int64_t amp id, int16_t channel (N/A), int16_t value (0 = On, 1 = Off)	This command enables the reference signal. It causes the reference electrode to become active and is used when measuring differential inputs on the channels.
cmd_SetOscillatorGate	int64_t amp id, int16_t channel (N/A), int16_t value	This command disconnects the calibration signal from the switches used to introduce the signal on a channel or channels. When the calibration signal is not in use, having that signal on the open switch can still induce signals on the channels. This disconnects the signal well before the amplifier inputs to eliminate any induced signals in the data.
cmd_SetReference10KOhms	int64_t amp id, int16_t channel (N/A), int16_t value	Like the *cmd_TurnChannel10KOhms* command, this command places a 10 KOhm resistor on the input to the reference channel. This can be used to measure impedance and noise of the reference channel.
cmd_SetReferenceDriveSignal	int64_t amp id, int16_t channel (N/A), int16_t value	Like the *cmd_TurnAllDriveSignals* command, this command turns on the calibrations signal on the input to the reference channel. This can be used for debugging and diagnostics for measuring impedance and gains.
cmd_SetPower	int64_t amp id, int16_t channel (N/A), int16_t (0 = Off, 1 = On)	This command turns On and Off the amplifier.
cmd_Reset ***NA 400s only***	int64_t amp id, int16_t channel (N/A), int16_t value (N/A)	This command resets the selected amplifier.
cmd_SetWaveShape	int64_t amp id, int16_t channel (N/A), int16_t (0 = Off (default sine wave), 1 = square wave, 2 = triangle wave, 3 = saw tooth [***NA 400 only***])	This command determines the waveform of the calibration signal. By default, a sine wave is used.
cmd_SetDrivenCommon	int64_t amp id, int16_t channel (N/A), int16_t state (0 = Off, 1 = On)	This command is intended for use with CANCL technology. This provides for active cancellation of noise using the signal from the reference channel. When activated, the common channel is driven with the signal from the reference to actively cancel the noise signal on the normal EEG channels. This capability is only used in conjunction with EEG Nets that support CANCL technology.
cmd_ SetCalibrationSignalAmplitude	int64_t amp id, int16_t channel (set by default to channel 8), int16_t value (12 bit range)	This command sets the amplitude of the calibration signal.
cmd_SetAnalogOutput ***NA 300s only***	int64_t amp id, int16_t channel (), int16_t value (12 bit range)	This command sets the selected channel on the selected amplifier to provide D/A output with a signal strength determined by 'value'. Value is a 12-bit integer, 0 -> 4095.
cmd_SetDigitalOutputData	int64_t amp id, int16_t channel (N/A), int16_t value (bit mask of DIO line to be set)	This command sets the bits of the 16-bit Digital I/O port on the amplifier. By default, all DIO bits are intended for input, and this command will have no effect. To set specific bits to outputs bits, use the *cmd_SetDigitalInOutDirection* command.

Command	Arguments	Description
cmd_SetDigitalInOutDirection	int64_t amp id, int16_t channel (N/A), int16_t value (bit mask indicating the direction status of the I/O lines)	This command is to tell the amplifier to use specific bits of its 16-bit DIO board for input. By default, all DIO bits are intended for input. Their default behavior can be changed using this command. For example, to set the first 8 bits for input and the last 8 bits for output, send the binary value (1111111100000000) = 0xFF00 = 65280 in this command. The amplifier masks input bits when writing to output bits, so for this example, if a value is sent using the *cmd_SetDigitalOutputData* command that has bits set in the first byte (e.g., 00000010 01010110), these will not affect bits that are set for input. In this example, if all bits were zero before this call, the bits when read will be 00000000 01010110. Note that the second bit in the first byte is not set to one. A bit set to input is not affected by a *cmd_SetDigitalOutputData* command. Likewise, the value of a bit set to output is not affected by a signal on the port.
cmd_IQAmpData ***NA 300s only***	int64_t amp id, int16_t channel (N/A), int16_t value (N/A)	This command queries the digital information on the DIO port. It provides two bytes, which is the size of the DIO port indicating the corresponding bits on that port. This is TTL, so an input voltage on a given bit over 3.5 volts will be considered a 1 and voltage below this a zero. Note that any bits that are assigned for output will return their most recent value set by the *cmd_SetDigitalOutputData* command when this command is called. This command will not affect the state of any bit, whether for input or output.
cmd_GetStartTime	int64_t amp id, int16_t channel (N/A), int16_t value (N/A)	This command gets the start time of the selected amplifier. This value is set every time the amplifier is started, and represents the system time when the amplifier started, measured in microseconds that have passed since the Epoch: 00:00:00 January 1, ????.
cmd_GetCurrentTime	int64_t amp id, int16_t channel (N/A), int64_t value (N/A)	This command is not supported.
cmd_GetCurrentDrift	int64_t amp id, int16_t channel (N/A), int64_t value (N/A)	This command is not supported.
cmd_SetFilterAndDecimate ***NA 300s only***	int64_t amp id, int16_t channel (N/A), int16_t (0 = Off, 1 = On)	This command turns On and Off the Filter and Decimation routines. When Off, filtering and decimation is not performed and data is propagated to clients at the amplifier's native sampling rate. When On, filtering and decimation is performed according to the requested decimation (sampling) rate. Also see cmd_SetDecimatedRate.
cmd_SetNativeRate ***NA 400s and NA 410s only***	int64_t amp id, int16_t channel (N/A), int16_t value	This command is similar to cmd_SetFilterandDecimate, except that for the NA 400 and NA 410 you are getting the actual native sampling rate of the amplifier. **For NA 400s**: The values are 500, 1000, 2000, 4000, and 8000. **For NA 410s**: Only the decimated rate of 20000 is supported. Note that there is no software filtering at these rates, just a hardware filter on the sampling circuits. Once this command is sent, a subsequent call to cmd_SetDecimateRate will negate it.

Command	Arguments	Description
cmd_SetDecimatedRate	int64_t amp id, int16_t channel (N/A), int64_t value	This command sets the decimation rate or downsampling rate of the amplifier's native rate to specific values from that amplifier's set: • ***NA 400s only***: 250, 500, 1000 • ***NA 300s only***: 50, 100, 200, 250, 500, or 1,000
cmd_SetMRIPulseInfo	int16_t channel, int64_t value (high byte = DIN line (0 = detection off), low byte = DIN value)	This command is only used for MRI compatible amplifiers and is not supported in the Amp Server Pro SDK.
cmd_setPIBChannelGain	int64_t amp id, int16_t channel (1-32), int16_t value (NA 400/Physio16 gain = 1, 2, 3, 4, 6, 8, and 12; NA 300/PIB gain = 2, 20, 200, and 2,000)	Sets the gain of the named Physio16 (used with NA 400s) or PIB (used with NA 300s) channel to the specified value. Note: A gain of 20 matches the gain of EEG channels.
cmd_TurnChannelZeroOhms ***NA 400s only***	int64_t amp id, int16_t channel (0-287), int16_t value(0 = normal operation, 1 = ground)	If the value is 1, this command shorts the specified channel to ground. If the value is 0, this command sets the specified channel to operate normally. All other values are ignored.
cmd_TurnAllZeroOhms ***NA 400s only***	int64_t amp id, int16_t channel (N/A), int16_t value(0 = normal operation, 1 = ground)	Like the cmd_TurnChannelZeroOhms command, this command shorts all channels to ground if the value is 1, or normal if the value is 0. All other values are ignored.
cmd_SetPhoticStimSequence ***NA 400s only***	int64_t amp id, int16_t channel (N/A), char * value (mff PhoticStimRun xml string)	The MFF layer defines an XML format for PhoticStim sequences. Sending this command with a string composed of an xml PhoticStim sequence as the value will cause the NA 400 to parse the string and begin issuing PhoticStim pulses as specified.
cmd_ GetPhysioConnectionStatus ***NA 400s only***	int64_t amp id, int16_t channel (N/A), int16_t value(N/A)	Issuing this command will cause the NA 400 to query the connection status of Physio16 hardware and issue a notification (over the notification port). Returned values can be interpreted as follows: 0: No Physio16 hardware is attached to the NA 400. 1: One Physio16 unit is attached on port 1. 2: One Physio16 unit is attached on port 2. 3: Two Physio16 units are attached.
Amplifier Status		
cmd_GetAmpDetails	int64_t amp id, int16_t channel (N/A), int64_t value (N/A)	This command returns details about the amplifier.
cmd_GetAmpStatus	int64_t amp id, int16_t channel (N/A), int64_t value (N/A)	This command returns general status information about the amplifier. It is currently not implemented.
Extended Amplifier Configuration Settings		
cmd_DefaultAcquisitionState	int64_t amp id (N/A), int16_t channel (N/A), int64_t value (N/A)	This is an 'extended' command that is received by Amp Server and is executed by sending the commands listed below to the amplifier. The default acquisition state is the normal state in which to place the amplifier for data acquisition. It turns off all components that are used for measuring gains, impedance, or noise and makes sure any calibration signal is off.

Command	Arguments	Description
		cmd_TurnAll10KOhms, channel = 0, value = 0 cmd_TurnAllDriveSignals, 0, value = 0 cmd_SetSubjectGround, channel = 0, value = 1 cmd_SetCurrentSource, channel = 0, value = 0 cmd_SetCalibrationSignalFreq, channel = 0, value = 0 cmd_SetBufferedReference, channel = 0, value = 1 cmd_SetOscillatorGate, channel = 0, value = 0 cmd_SetReference10KOhms, channel = 0, value = 0 cmd_SetReferenceDriveSignal, channel = 0, value = 0 cmd_SetWaveShape, channel = 0, value = 0 cmd_SetCalibrationSignalAmplitude, channel = 0, value = 0 cmd_SetAnalogOutput, channel = 7, value = 0 cmd_SetDrivenCommon, channel = 0, value = 0
cmd_DefaultSignalGeneration	int64_t amp id (N/A), int16_t channel (N/A), int64_t value (N/A)	This is an 'extended' command that is received by Amp Server and is executed by sending the commands listed below to the amplifier. The default signal generation state is a typical state in which to place the amplifier for testing or debugging purposes. It enables the calibration signal and switches on the calibration signal for all channels. cmd_TurnAllDriveSignals, channel = 0, value = 0 cmd_SetReferenceDriveSignal, channel = 0, value = 0 cmd_TurnAllDriveSignals, channel = 0, value = 1 cmd_SetSubjectGround, channel = 0, value = 0 cmd_SetCalibrationSignalFreq, channel = 0, value = 5 cmd_SetBufferedReference, channel = 0, value = 0 cmd_SetOscillatorGate, channel = 0, value = 1 cmd_SetCalibrationSignalAmplitude, channel = 0, value = 50 cmd_SetAnalogOutput, channel = 7, value = 50
Amp Server	**Note: If a command is listed in AS_Network_Types_h, but not here, it is not supported in the SDK.**	
cmd_NumberOfAmps	int64_t amp id (N/A), int16_t channel (N/A), int64_t value (N/A)	This command returns the number of amplifiers (active or not) seen by the system. *Active* usually means an amplifier is connected, but amplifiers can be *not active* for various reasons. To determine a connected amplifier's state, request its information.
cmd_NumberOfActiveAmps	int64_t amp id (N/A), int16_t channel (N/A), int64_t value (N/A)	Returns the number of amplifiers that are active in the system. *Active* usually means connected and available to stream data.
cmd_ListenToAmp	int64_t amp id, int16_t channel (N/A), int64_t value (N/A)	When connecting to the stream port, this command is sent to indicate the amplifier for which streaming is desired on this connection port.
cmd_StopListeningToAmp	int64_t amp id, int16_t channel (N/A), int64_t value (N/A)	This command is sent to stop listening to data from the amplifier on this port.
cmd_ReceiveNotifications	int64_t amp id (N/A), int16_t channel (N/A), int64_t value (N/A)	When connecting to the notification port, this command is sent to indicate the amplifier for which notifications are desired on this connection port.
cmd_ StopReceivingNotifications	int64_t amp id (N/A), int16_t channel (N/A), int64_t value (N/A)	This command is sent to stop receiving notifications from the amplifier on this port.
cmd_InstallEGINA300TestAmp ***NA 300s only***	int64_t amp id (N/A), int16_t channel (N/A), int64_t value (N/A)	This command installs the test amplifier, a software based amplifier that emulates most of the major functions of the amplifier.
cmd_Exit	int64_t amp id (N/A), int16_t channel (N/A), int64_t value (N/A)	This command tells Amp Server to terminate.

Command Responses

Command responses will always be of the form:

(sendCommand_return …)

In addition, commands will always respond with status information relating to the command itself. This is indicated by the status element. Below is a list of valid status elements:

(status complete)

(status error)

Therefore, for a command that does not return any additional information, the two valid options are:

(sendCommand_return (status complete))

(sendCommand_return (status error))

Other commands return information. For example, the cmd_GetAmpDetails command returns:

(sendCommand_return (status complete) (amp_details (serial_number A14150128) (amp_type NA400) (legacy_board false) (packet_format 2) (system_version 1.6.15) (number_of_channels 256)))

Here are a few examples of additional information elements. Please refer to the AS_Network_Types.h header for an up-to-date list of supported commands and their return types.

(amp_server_status_info …)

(number_of_amps …)

(amp_status_info …)

(start_time …)

(current_time …)

(dio_values …)

Please note, the form (sendCommand_return …) is wrapped up in an AS_ReturnValue class if using the high-level API. For usage details, see the example code.

Experimental Control Interface (ECI) Commands and Return Values

as defined in AS_Network_Types.h

Specific Command	Controller Command (cmd) or Response	Follows controller command if more data expected	Description
ECI Commands			
eci_Query	**Q** controller cmd	cccc	This command is the legacy method for conveying machine type. It uses a four character code. Note that these are no longer descriptive of the actual machine type or byte order. NTEL, MAC+, etc., are not generally known to be big- or little-endian, but you should choose the appropriate one.
eci_NewQuery	**Y** controller cmd		This command is the new query for machine type and map specifically to the chip used by the OS. The return value should be one byte, and be the ECI version number.
eci_Exit	**X** controller cmd		This command ends the communications session. The server can assume that communications will no longer be sent after sending back the acknowledgement.
eci_BeginRecording	**B** controller cmd		This command tells the server application to start recording. The return value should be one byte.
eci_EndRecording	**E** controller cmd		This tells the server application to stop recording. The return value should be one byte.
eci_Attention	**A** controller cmd		This command is sent in advance of the chosen sync command. It provides the server indication that the sync command is about to be sent so it knows not to change anything on the server side that might affect synchronization.
eci_ClockSynch	**T** controller cmd	1111	This is the non-NTP synchronization command. Sending this command will result in the server recalculating the offset between clocks. This enables the client and server to account for the drift between the systems.
eci_NTPClockSynch	**N** controller cmd	1111	This command indicates that the client is also an NTP client and has a clock that is synchronized with the server clock (and in the case of Net Station, it is synchronized with the EEG being recorded. The time sent represents the absolute start time of the clients clock in NTP time format. The server will compare this value with the absolute time of start of its clock to determine the offset between clocks. Sending this command more than once will have no effect as the start times of each clock does not change. The return should be one byte.
eci_NTPReturnClock Synch	**S** controller cmd		This command is identical to the eci_NTPClockSynch command except that the server returns the start time of its clock in NTP form as well. This allows the client to adjust its clock or timestamps to already be relative to the server's clock sparing the server the responsibility to adjust incoming timestamps in events. The return value should be 8 byte NTP time from server.

Specific Command	Controller Command (cmd) or Response	Follows controller command if more data expected	Description
eci_EventData	**D** controller cmd	<data>	This command sends data as a timestamp event. In all cases, timestamps will be 32 bit integers representing relative time from some absolute start time. For normal (eci_ClockSynch) and NTP (eci_NTPClockSynch) sync methods, the timestamps for these events will be relative the start time of the client's clock. For NTP with return time (eci_NTPReturnClockSynch), these relative times will already be converted so they are relative to the servers clock. The return value should be one byte.
ECI Return Values			
eci_OK	**Z** response		One byte return value equal to 'Z'.
eci_Failure	**F** response		One byte return value equal to 'F'.
eci_NoRecordingDeviceFailure	**R** response		One byte return value equal to 'R'.
eci_Identify	**I** response		One byte return value equal to 'I'.

Experiment Control Protocol

The following aids you in your design of a custom external controller.

Requirements

Hardware Requirements

A TCP/IP connection is required to link the Net Station EEG data acquisition computer and the experimental control computer (ECC). Net Station must run on a Mac computer, but the machine providing experiment control need not be a Mac.

Whatever the experiment control machine, it must communicate to Net Station via TCP/IP.

For TCP/IP port communication, you can connect to ports 55513, 55514, 55515, or 55516—port 55513 is preferred.

During the initial query, you should send the machine "type" of the ECC, if using the legacy query command. Type in this case means byte order. For example, Intel processors are typically little-endian; therefore, you would send the following during a query exchange:

'QNTEL'

- Q for the query command
- NTEL for the machine type

You may use either the new or legacy query command, as follows.

```
// ECI machine types.
static const ECIMachineType  eci_unknown =      'u';
static const ECIMachineType  eci_i386 =         'i';
static const ECIMachineType  eci_x86_64 =       'x';
static const ECIMachineType  eci_ppc =          'p';


/**
      *  Legacy ECI machine types.
     *
     *  Notes:
     *
     *  1) Given that all clients are distributing these fourcc types as
     *  simple char strings,
     *  is is probably better to move these to char strings in the future.
      */
    typedef enum {

        eciMachineType_Mac          = 'MAC-',   // Big-endian
        eciMachineType_Unix         = 'UNIX',   // Big-endian
        eciMachineType_Intel  = 'NTEL'    // Little-endian


    } LegacyECIMachineType;
```

Note: *Little-endian* refers to a method of storing two- and four-byte integers in which the lower address has the lower significance, or "little end first." Most processors, other than Intel machines and VAXen, use *big-endian* notation, which means that the most significant byte, or "big end," is stored in the lower address.

Software Requirements

Amp Server Pro SDK 2.1 works within the following software environments:

- Net Station 5.x
- Net Station 4.5.x (legacy)
- A server-side environment that supports the ECI protocol

Protocol Description

The following serves as more detailed descriptions of the usage of the commands and types listed in the AS_Network_Types.h header file. There is some repetition for the sake of clarity.

These commands are designed to reduce communication latency by minimizing the amount of data transmitted.

Commands

All commands consist of a single ASCII character followed by a zero or additional bytes of information. Additional information, except in the case of the **D** command, is expected to be one of four data types.

Data type descriptions

Data type	Description
cccc	A four-character descriptor type (DescType); usually printable ASCII characters
b	A single byte interpreted as a number between -128 and 127
ss*	A short integer (2 bytes) representing a number between -32,768 and 32,767
llll*	A long integer (4 bytes) representing a number between -2,147,483,648 and 2,147,483,647

*Numbers represented by ss and llll are machine dependent.

Command Summary

Commands, responses, and descriptions

Controller command	Brief description	Net Station responses	Brief description
Qcccc	Hello, I am … (machine type)	**IB** or **Fss**	Identify/I'm version … or Failure
X	Exit	**Z** or **Fss**	OK or Failure
B	Begin recording	**Z** or **Fss**	OK or Failure
E	End recording	**Z** or **Fss**	OK or Failure
A	Attention	**Z** or **Fss**	OK or Failure
Tllll	Clock sync	**Z** or **Fss**	OK or Failure
D<data>	Event data stream	**Z** or **Fss**	OK or Failure after complete transmission

Command Descriptions

Command Qcccc. Initiates a connection between the ECC computer and the EEG computer. The four-character descriptor type indicates which computer type is connected so that Net Station can account for any integer notation problems.

The descriptor must be one of the following:

'MAC-'	Macintosh computer
'UNIX'	UNIX computer
'NTEL'	Intel computer (i.e., a PC)

Command X. Indicates that the experiment has successfully ended. Net Station closes the file to which it has been sending data (the Recording file) and returns to an idle state.

Command B. Instructs Net Station to begin recording. If a Recording file has not been specified, then a new file is created using the default naming scheme. If the Acquisition Setup does not contain a device that belongs to the Recorder class, then Net Station returns an error.

Command E. Ends the recording but does not close the file. Net Station is capable of storing multiple recording epochs in a single file and does not impose a limit on the number of B and E sequences that it might receive per file.

Command A. Requests Net Station to do nothing else until another command is sent. Use the command to prepare Net Station for time synchronization. Once Net Station has successfully responded to the A command, a T command is usually sent.

Command T1111. Synchronizes the ECC computer's time base with Net Station's time base. The long value is the ECC's current time in milliseconds.

Command D. Transmits event information. An event may be loosely defined as any occurrence that has both time and duration and needs to be recorded. Net Station provides a flexible and extensible structure for storing events for later retrieval and display. The transmission protocol for the data following this command is described in detail in the following section.

Note: To synchronize transmitted stimulus/response data, Net Station stores the ECC's time value, along with its own time value, at the moment of synchronization. When an event is received, Net Station subtracts the event time from the ECC's sync time and adds this to Net Station's time sync value. In other words, you should only send event times that are relative to the start of the recording in milliseconds.

Event Data Stream

The ECI device creates stimulus events by converting the information accompanying the D command into an event object. An event object is composed of several required fields, several optional fields, and an optional list of keyword-described data fields.

Event data stream fields and descriptions

Field	Required/Optional	Description
Start Time	Required	Event start time (in milliseconds)
Duration	Required	Event duration (in milliseconds). All events must have a duration of at least one millisecond.
Source Device	Required	Net Station stimulus device from which the event originated. This field is assigned by Net Station and cannot be programmed by an external controller.
Event Code	Required	A unique, user-defined, four-character descriptor type that identifies this event type. For instance, you might use the code 'AUDI' to indicate an audio stimulus.
Event Description	Optional	A string that even further describes the event.
Key List	Optional	A list of fields containing additional properties and data to further relate information about the event.

⚠ **CAUTION:** Do not send simultaneous ECI events (that is, ECI events with the same time stamps) to Net Station. Although Net Station can record simultaneous ECI events, users cannot access them properly because the events are on the same track. Simultaneous events on different tracks (for example, DIN events) can be accessed properly.

Event Key Field Descriptions

Event key data types are described and used by Net Station as follows. Other user-defined data types are not excluded or rejected by Net Station event-handling functions.

Data length. Length (in bytes) of the *variable length data* field.

Data type. A four-character descriptor type that describes the data type of the data stored in the Event Key. Net Station currently supports seven predefined descriptor types.

Key. A unique four-character identifier for the data.

Variable length data. The raw data.

Predefined descriptor types (codes have built-in meaning)

Type	**Length** (bytes)	**Description**
'bool'	1	Boolean (0 or 1)
'shor'	2	Short signed integer
'long'	4	Long signed integer
'sing'	4	Single-precision real
'doub'	8	Double-precision real
'type'	4	Four-character descriptor type
'TEXT'	variable	Sequence of characters

Note: Mac OS developers will recognize these descriptor types as Apple event descriptor types.

The data following the Data Event command are flattened and packed representations of the preceding event object structure. The entire Data Event command sequence is as follows.

Data Event command sequence

Type	**Length** (bytes)	**Description**
'D'	1	Data Event command
unsigned short	2	Length of following data block (bytes)
long	4	Event start time
long	4	Event duration
Desc	4	TypeEvent code
byte	1	Event label length, or 0 if no label
characters	variable	Event label (256 characters max.)
byte	1	Event description length, or 0 if no description
characters	variable	Event description (256 characters max.)
unsigned character	1	Number of keys, or 0 if no keys
DescType	4	Key

Type	**Length** (bytes)	**Description**
DescType	4	Data type
unsigned short	2	Length
??	variable	Variable length data

Response Descriptions

Response Ib. Follows a successful query command by the controlling device. The single byte of data following the command indicates the version number of the protocol that is currently being supported.

Response Z. Follows any other successful command other than the initial query.

Response Fss. Is sent by Net Station when it receives an incorrectly constructed or undefined command, or is unable to comply with the command's instructions. No error numbers are currently defined.

Reading the Data Stream

CAUTION:
Packet format is firmware version dependent. Before upgrading your amplifier's firmware, confirm compatibility with EGI Technical Support. Otherwise, an upgrade could break your code.

There are two packet formats, depending upon the Net Amps model being used and, in the case of the NA 400, the version of the firmware.

Amplifier Firmware	Packet Format 1	Packet Format 2
NA 300	✔	▨
NA 400 *with 1.4.3 and earlier*	✔	▨
NA 400 *with 1.6.3 and later*	▨	✔

Packet Format 1

The network protocols deliver data with Packet Format 1 in the following manner:

Layout	| 8 Bytes (Network Byte Order)	| 8 Bytes (Network Byte Order)	| Sample Packets… |
Description	| Amp ID	| Size of Data (in bytes)	| Sample Packets in Packet Format 1 |

*When using the high-level API, the data received in the AmpDataObject's update function has already removed the Amp ID and the size of the data fields. These fields are only present if you are working with the network layers directly. See the Simple Client in the examples for the high-level usage.

The data packets for Packet Format 1 are as follows:

For clarity (from AS_NetworkTypes.h):

```
#pragma pack(1)

    /**
     *  Packet Format 1 Used by the NA300 and NA400 prior to and including
     *  version 1.4.3 of the NA400 firmware.).
     *
     *  <b>Notes:</b><br>
     *  &nbsp; 1) Used by the NA300 and NA400 prior to and including
     *  version 1.4.3 of the NA400 firmware.<br>
     *
     *  &nbsp; 2) If this is raw data from the amplifier, it will be in analog to digital units
     *  and this will need to be converted to microvolts. Please see the Amp Server Pro SDK
     *  documentation for details.<br>
     *
     *  &nbsp; 3) Data in this packet format is always in network byte order.<br>
     */
    typedef struct
    {
        uint32_t  header[8];    // DINS (Digital Inputs) 1-8/9-16 at bytes 24/25; net type at byte 26.
        float     eeg[256];     // EEG Data.
        float     pib[7];       // PIB data.
        float     unused1;      // N/A
        float     ref;          // The reference channel.
        float     com;          // The common channel.
        float     unused2;      // N/A
        float     padding[13];  // N/A
    } PacketFormat1;

#pragma pack()
```

Thus, the size of samples in Packet Format 1 is 1152 bytes for each sample (32 bytes of header + 1120 bytes of float data).

Typically, the order of operations on a coding level is to first read 16 bytes of "Amp ID" and "Size of Data", then read the "Size of Data" number of bytes from the network port. For example, suppose the following was sent.

| 0 | 5760 | Samples |

Your software would read the first 16 bytes. This will give an Amp ID of 0, and a size for the samples section of 5760. This means that you have received 5760 / 1152 = 5 samples laid out in Packet Format 1.

Header and Channel Data For Packet Format 1

Header. The header data is largely unused except for 3 bytes of interest:

DINs and net are stored in the sample header (1 byte indices)

```
            0-based
dins 1-8 =  24
dins 9-16 = 25
net type =  26
```

Channel Data. This data is in a raw format and needs to be converted to microvolts before use. For instructions, refer to the "Scaling Factors - AD unit to microvolt" document included in the documentation.

Packet Format 2

As with Packet Format 1, you will receive 16 initial bytes, as the network protocols deliver data with Packet Format 2 in the following manner. Note the change to Packet Format 2.

Layout	| 8 Bytes (Network Byte Order)	| 8 Bytes (Network Byte Order)	| Sample Packets… |
Description	| Amp ID	| Size of Data (in bytes)	| Sample Packets in Packet Format 2 |

*When using the high-level API, the data received in the AmpDataObject's update function has already removed the Amp ID and the size of the data fields. These fields are only present if you are working with the network layers directly. See the Simple Client in the examples for the high-level usage.

The data packets for Packet Format 2 are as follows:

```
#pragma pack(1)

     /**
     *  Packet Format 2 (Used NA400 in firmware versions AFTER 1.4.3.).
     *
     *  <b>Notes:</b><br>
     *  &nbsp; 1) If this is raw data from the amplifier, it will be in analog to digital units
     *  and this will need to be converted to microvolts. Please see the Amp Server Pro SDK
     *  documentation for details.<br>
     *
     *  &nbsp; 2) If this is raw data from the amplifier, it will be in analog to digital units
     *  and this will need to be converted to microvolts. Please see the Amp Server Pro SDK
     *  documentation for details.<br>
     *
     *  &nbsp; 3) Data in this packet format is always in little endian byte order.<br>
     */
    typedef struct
    {
        uint8_t digitalInputs;
        uint8_t status;
        uint8_t batteryLevel[ 3 ];
        uint8_t temperature[ 3 ];
        uint8_t sp02;
        uint8_t heartRate[ 2 ];
    } PacketFormat2_PIB_AUX;

    typedef struct
    {
        uint16_t                digitalInputs;
        uint8_t                 tr;
        PacketFormat2_PIB_AUX   pib1_aux;
        PacketFormat2_PIB_AUX   pib2_aux;
        uint64_t                packetCounter;
        uint64_t                timeStamp;
        uint8_t                 netCode;
        uint8_t                 reserved[ 38 ];
        uint32_t                eegData[ 256 ];
        uint32_t                auxData[ 3 ];
        uint32_t                refMonitor;
        uint32_t                comMonitor;
        uint32_t                driveMonitor;
        uint32_t                diagnosticsChannel;
        uint32_t                currentSense;
        uint32_t                pib1_Data[ 16 ];
        uint32_t                pib2_Data[ 16 ];
    } PacketFormat2_SamplePacket;

#pragma pack()
```

The size for each packet is 1264 bytes.

As with Packet Format 1, the order of operations on a coding level is to first read 16 bytes of "Amp ID" and "Size of Data", then read the "Size of Data" number of bytes from the network port. For example, suppose the following was sent.

| 0 | 6320 | Samples |

Your software would read the first 16 bytes. This will give an Amp ID of 0, and a size for the samples section of 6320. This means that you have received 6320 / 1264 = 5 samples laid out in Packet Format 2.

Sampling Rates and Net Types For Both Formats

Sample Rate. With the NA 300 and NA 400 amplifiers, the sampling rate can drop below 1000 samples per second.

For the NA 300, this is accomplished using the following commands:

"`cmd_SetFilterAndDecimate`" // This sets the NA 300 to (or not to) filter and decimate the data.

"`cmd_SetDecimatedRate`" // This sets the rate at which you want this to occur.

For the NA 400, this is accomplished using the following command:

" `cmd_SetDecimatedRate`"

For command details, see section "Amplifier and Amp Server Commands."

If the sampling rate drops below 1000 s/s, you will STILL receive data at 1000 s/s. This is for legacy reasons. Thus, if you drop the sampling rate to 250 s/s, the data will be "up-sampled" back to 1000 s/s by replicating. You will, in this case, receive four identical samples (almost), and thus you can discard three out of the four samples. The almost in the preceding sentence is due to the digital inputs. In the case of the NA 300, the digital inputs are always at a rate of 1000s/s, and this information is preserved during the up-sample process. In the case of the NA 400, the samples are truly identical.

It is possible to change the decimation rate of the NA 400 amplifier. This mirrors the 1000 s/s TMS setting in Net Station Acquisition (v 5.2). By doing this, you can avoid the decimation filter delays at 1000 s/s. Be aware, however, that your effective analysis frequency range is one quarter of the sampling rate in this mode. Therefore, at 1 k samples per second, it is 1000/4 = 250 Hz. The command for this is: cmd_SetNativeRate.

If you select a sampling rate of less than 1000 s/s (i.e., 500), you will see repeated samples up to 1 k as with the cmd_SetDecimatedRate command.

Note that Net Station 5.2 supports only the 1000 s/s TMS setting. Amp Server Pro SDK, however, provides more options.

For each amplifier's hardware anti-alias filter delay at different sampling rates, and for details regarding the 1000 s/s TMS sampling rate, refer to the Net Station 5 manual (8100050).

Physiological Unit Data – Packet Format 2 Only

A good example of how to decode physiological aux information (PacketFormat2_PIB_AUX) is provided in the Simple Client example in the class DataStreamObserver, and should be referred to for more information. Up to two physiological units are supported on a single NA 400 amplifier, as can be seen from the PacketFormat2_SamplePacket data structure.

There is an accepted mechanism (perhaps not ideal) for detecting the presence of a physiological unit, and this is exampled in the PhysioDetection class, again in the Simple Client example.

Net types (or codes as they are sometimes called):

```
// Net codes
typedef enum{
  GSN64_2_0,       //  GSN 64
  GSN128_2_0,      //  GSN 128
  GSN256_2_0,      //  GSN 256

  HCGSN32_1_0,     //  HGSN 32
  HCGSN64_1_0,     //  HGSN 64
  HCGSN128_1_0,    //  HGSN 128
  HCGSN256_1_0,    //  HGSN 256

  MCGSN32_1_0,     //  MGSN 32
  MCGSN64_1_0,     //  MGSN 64
  MCGSN128_1_0,    //  MGSN 128
  MCGSN256_1_0,    //  MGSN 256

  AMP_SAMPLE,      //  EAmpSample displayable channels (internal use only)
  TestConnector = 14,
  NoNet = 15,      //  net not connected
  Unknown = 0xFF   //  Unknown or net not connected

}NetCode;
```

Scaling Factors for NA 400s and NA 300s

In order to convert raw data to microvolts, scaling factors need to be applied to the raw streaming data. Which scaling factors are used is dependent upon the type of amplifier (NA 400 or NA 300) that is streaming the raw data and the specific data being converted. In general, multiply the raw value by the specific scaling factor (as indicated by the scaling factor tables below) to get the microvolts.

```
microvolts = raw_value x scale_factor
```

Note: All scaling factors assume the gains shown in the calculations on pages 22-23.

NA 400 Scaling Factors

Variable Name	NA 400 Scaling Factors	scale_factor
eegData [0 - 255]	NA400 EEG Scale Factor	0.00009313225
auxData [0 - 2]	NA400 EEG Scale Factor	0.00009313225
refMonitor	NA400 EEG Scale Factor	0.00009313225
comMonitor	NA400 EEG Scale Factor	0.00009313225
driveMonitor	NA400 EEG Scale Factor	0.00009313225
diagnosticsChannel	NA400 EEG Scale Factor	0.00009313225
pib1_Data [0 - 7]	Physio16 Scale 1-8	-0.00111758708
pib1_Data [8 - 15]	Physio16 Scale 9-16	0.00111758708
pib2_Data [0 - 7]	Physio16 Scale 1-8	-0.00111758708
pib2_Data [8 - 15]	Physio16 Scale 9-16	0.00111758708

NA 300 Scaling Factors

Variable Name	NA 300 Scaling Factors	scale_factor
eegData [0 - 255]	SwapInt32 and NA300 EEG Scale Factor	0.0244140625
pib [0-2 & 4-7]	SwapInt32 and NA300 EEG Scale Factor	0.0244140625
pib [3] (Body position)	SwapInt32 and NA300 EEG Scale Factor * 10	0.0244140625

Scaling Factors

SwapInt32. Data from the NA 300 arrives in big-endian format and must be swapped to little-endian on i386 machines.

NA300 EEG Scale Factor. For all NA 300 amplifiers, use the following scaling factor:

```
ref_voltage    = 4.096
amp_gain       = 20
range_in_volts = ref_voltage * 2 / amp_gain
               = 4.096 * 2 / 20
               = 0.4096

volts_per_bit = range_in_volts >> 24
              = (ref_voltage * 2 / amp_gain) >> 24
              = 2.44140625e-8

mv_per_bit    = volts_per_bit * 1,000,000
              = ((ref_voltage * 2 / amp_gain) >> 24 ) * 1,000,000
              = 0.0244140625

eeg_scale     = mv_per_bit
              = 0.0244140625
```

Note: Please ensure that the gain values in as.conf are set to the correct gains for the PIB channels.

NA400 EEG Scale Factor. If the NA 400 amplifier has a legacy board (currently, only the NA 410), use the following scaling factor:

```
ref_voltage    = 4.096
amp_gain       = 19.7936

range_in_volts = ref_voltage * 2 / amp_gain
               = 4.096 * 2 / 19.7936
               = 0.41387115027

volts_per_bit  = range_in_volts >> 24
               = (ref_voltage * 2 / amp_gain) >> 24
               = 2.46686429e-8

mv_per_bit     = volts_per_bit * 1,000,000
               = ((ref_voltage * 2 / amp_gain) >> 24 ) * 1,000,000
               = 0.02466864289

eeg_scale      = mv_per_bit >> 8
               = (((ref_voltage * 2 / amp_gain) >> 24 ) * 1,000,000) >> 8
               = 0.00009636188
```

For all other NA 400 amplifiers, use the following scaling factor:

```
ref_voltage    = 4.0
amp_gain       = 12

range_in_volts = ref_voltage * 2 / amp_gain
               = 4.0 * 2 / 12
               = 0.4
               = 0.666 . . . 7
```

```
volts_per_bit  = range_in_volts >> 24
               = (ref_voltage * 2 / amp_gain) >> 24
               = 2.38418579e-8
               = 3.973643e-8

mv_per_bit     = volts_per_bit * 1,000,000
               = ((ref_voltage * 2 / amp_gain) >> 24 ) * 1,000,000
               = 0.02384185791
               = 0.03973642985026

eeg_scale      = mv_per_bit >> 8
               = (((ref_voltage * 2 / amp_gain) >> 24 ) * 1,000,000) >> 8
               = 0.00009313225
               = 0.0001552204291
```

Physio16 Scale 1-8. For channels 1–8 of the Physio16-1 or Physio16-2, use the following scaling factor:

```
ref_voltage    = 2.4
amp_gain       = 1

range_in_volts = ref_voltage * 2 / amp_gain
               = 2.4 * 2 / 1
               = 4.8

volts_per_bit  = range_in_volts >> 24
               = (ref_voltage * 2 / amp_gain) >> 24
               = 2.86102295e-7

mv_per_bit     = volts_per_bit * 1,000,000
               = ((ref_voltage * 2 / amp_gain) >> 24 ) * 1,000,000
               = 0.28610229492

pib_scale_1_8  = -1 * mv_per_bit >> 8
               = -1 * (((ref_voltage * 2 / amp_gain) >> 24 ) * 1,000,000) >> 8
               = - 0.00111758708
```

Physio16 Scale 9-16. For channels 9–16 of the Physio16-1 or Physio16-2, use the following scaling factor:

```
ref_voltage     = 2.4
amp_gain        = 1

range_in_volts  = ref_voltage * 2 / amp_gain
                = 2.4 * 2 / 1
                = 4.8

volts_per_bit   = range_in_volts >> 24
                = (ref_voltage * 2 / amp_gain) >> 24
                = 2.86102295e-7

mv_per_bit      = volts_per_bit * 1,000,000
                = ((ref_voltage * 2 / amp_gain) >> 24 ) * 1,000,000
                = 0.28610229492

pib_scale_9_16  = mv_per_bit >> 8
                = (((ref_voltage * 2 / amp_gain) >> 24 ) * 1,000,000) >> 8
                = 0.00111758708
```

End-User Software License

The Amp Server Pro SDK is licensed to the end user for their own internal research use under the terms of the EGI Amp Server Pro SDK license. No use of the Amp Server Pro SDK to produce a product or service that will be resold, licensed, transferred, or shared is permitted by the end-user license. No clinical use of this product is permitted under this license. Users wishing to make commercial use of this product are invited to contact EGI regarding participation in its fee-based developers program. Contact abunnenberg@egi.com regarding participation in the developers program.

Amp Server Pro SDK
End-User Software License

Read the full license before downloading the software.

For questions or additional assistance, please contact us at:

Electrical Geodesics, Inc. (EGI)
500 E 4th Avenue, Suite 200
Eugene, OR 97401 USA

+1.541.687.7962 ● supportteam@egi.com ● www.egi.com