PSNADBoard.DLL and Linux Library Documentation

Version: Windows and Linux 5.2

Updated: April 29, 2014

Download Windows DLL and Documentation

Download Linux Library and Documentation

This document in PDF and MS Word formats

Introduction:

The PSNADBoardx.DLL (PSNADBoard32.dll and PSNADBoard64.dll) is a Windows Dynamic Link Library used to interface between the users application and the PSN-ADC-SERIAL / PSN-ADC-USB 16-Bit 8 Channel Analog to Digital Converter board, VolksMeter Interface board, or the new, PSN-ACCEL strong motion sensor and PSN-ADC24 digitizer boards. A Linux/Unix port of the DLL is also available for people who would like to develop applications on other platforms. The DLL/Library can be used with either 32 or 64 bit programs.

This Windows DLL or Linux Library was written to allow user to write their own software to configure and receive data from the ADC board. There are some major differences between the boards and the purpose of this DLL/Library is to isolate the differences with a common interface between the users program and the ADC board.

One advantage of using a DLL/Library is the user can write software in any programming language that can import DLL functions or use a library. This includes C, C++, Visual Basic and Java. The PSNADBoardx.DLL was developed using Visual C++ Version 8.0 and the Linux Library under OpenSUSE 11.4. It has been tested with Ubuntu 12.04 and MAC OS X 10.8.

Version 5.2 Changes:

Added support for the Ublox NEO-6M GPS Module. This GPS receiver can be purchased on eBay for around $15.00USD including free shipping from China. The user will need to add a RS-232 driver / receiver like a MAX232 between the receiver and A/D board. The user will also need to solder a wire on the GPS module to pin 1 on the DB9 GPS input connector on the A/D board. This is the 1 pulse per second signal needed for accurate timekeeping. Unfortunately the module we purchased did not have a solder pad for the 1PPS signal like the transmit and receive lines to/from the module. So the user must carefully solder a small diameter wire directly to pin 3 of the module. You will see a trace from pin 3 to the resistor and LED used to indicate the receiver is sending out good time and location information.

Added support for the Android operating system. See the android.zip file for more information and an example on how to get data into a Android device using a FTDI RS-232 adapter or one of our A/D boards with a USB interface.

This release also fixes several timing issues when a GPS receiver is used as a timing reference.

Version 4.8 Changes:

Improved error handling when the A/D board is reset. Under some circumstances Version 3 A/D boards could hang do to receiving data during CPU startup.

Changes from DLL/Library Version 3.x:

One of the difference is the ***AdcBoardConfig*** structure. This structure has been changed to add more parameters and increase the length of the *CommPortStr* string. It has also been renamed to *commPortTcpHost*. The new ***AdcBoardConfig2*** structure must be used when sending configuration information from your application to the DLL/Library.

One of the new features of the 4.x DLL/Library is the ability to open a TCP/IP connection to the ADC board through a Ethernet or Wireless to RS-232 adapter. The *commPortTcpHost* member of the structure is used to specify the IP address or host name of the device to connect to and the *tcpPort* member is used to specify the connection port.

Also included in the new DLL/Library is support for our new PSN-ACCEL Strong Motion and PSN-ADC24 24-Bit Digitizer boards.

How to Use the DLL or Linux Library in your Application:

Under Windows there are two ways for a program to use a DLL. The link library (PSNADBoardxx.LIB) can be added to the projects link list or the DLL can be loaded into the applications work space after the program starts up. If the reader has not used a DLL before, you should do a Google search to learn how to use them with the program language you plan to use to create your program. In the zip file containing the DLL and LIB files there is a Visual C++ program called AdcDemo.exe demonstrating how to use the DLL.

Applications written under Linux should simply link to the 32-Bit or 64-Bit library.

ADC Board Differences:

As noted above, there are some differences between the various PSN boards . Here are the differences that the programmer needs to be aware of. The following abbreviations will be used to differentiate the boards:

- V1 = Rabbit Core based ADC board
- V2 = PICC based ADC board
- V3 = dsPIC based ADC board
- VM = VolksMeter Interface board
- ADC24 = PSN-ACCEL Strong Motion board and the PSN-ADC24 24-Bit Digitizer board

Baud Rates:

V1 board support the following baud rates: 4800, 9600, 19200 and 38400

V2, V3, VM and ADC24 boards support the following baud rates: 9600, 19200, 38400 and 57600

The V1 board's baud rate is changed using a command sent to the board. V2, V3, VM and ADC24 boards have a DIP switch on the board that is used to set the baud rate. The boards are sent out with the baud rate set to 38400.

Sample Rates and Number of Channels:

V1 board support the following sample rates: 5, 10, 20, 25, 50, 100 and 200 SPS. At 200 SPS this board can record up to 4 channels. Below this sample rate the board can record all 8 channels.

V2 board support the following sample rates: 10, 20, 50, 100 and 200 SPS. This board can record data from all 8 channels independent of the sample rate.

V3 board support the following sample rates: 10, 20, 50, 100, 200, 250 and 500 SPS. This board can record up to 4 channels at 500 SPS and up to 8 channels for all other sample rates.

VM board supports the following sample rates: 5, 10, 20, 25, 40, 50 and 80 SPS. This board can record one or two channels.

ADC24 boards supports the following sample rates: 15, 25, 30, 50, 60, 100, 120 and 200SPS. This board can record one to four channels.

How to Collect Data using the DLL/Library:

There are two ways to receive data from the ADC board. The AdcDemo.exe (or AdcDemo under Linux) program shows how to use the two modes of operation.

Callback Mode:

To use this mode the user supplies a Callback function to the DLL/Library that will be used to send new ADC data and other ADC or DLL/Library messages to the application program. The user will need to use the Poll mode if the language you are using does not support Callback functions.

Polled Mode:

This is the preferred method to receive data from the ADC board. In this mode, messages containing ADC sample data and statues messages generated by the DLL/Library and ADC board are queued to prevent data loss. There is no queuing of the data in the Callback mode. In the Callback mode of operation the thread used to receive data from the ADC board directly calls the applications callback function. If the callback function takes too long to execute incoming data will be dropped do to the Comm port's input buffer overflowing.

The queue used to store data in the DLL/Library can keep up to 60 seconds worth of data. If the users does not poll the DLL for new data often enough, data will be dropped when the queue overflows.

In both modes of operation the user must relinquish CPU time so that the DLL/Library and other processed on the system can function properly. In C or C++ the _sleep(), Sleep() or usleep() (Linux) should be called periodically to allow other processed on the computer to run. In the poll mode, sleeping for 10 or 20 milliseconds seems to work fine. In the Callback mode the main thread or process should call the sleep function, do not place a sleep or wait function call in the Callback function!

Collecting Data from the ADC Board:

To start and stop collecting data from the ADC board the following sequence of events should be done in your application:

1. Open the DLL using the PSNOpenBoard() function.

2. Use the PSNSendBoardCommand() function with the ADC_CMD_SET_GAIN_REF parameter and *AdcConfig* structure. This is only needed for the PSN-ACCEL and PSN-ADC24 boards and if the gain for any channel needs to be changed from the default of 1.

3. Use the PSNConfigBoard() function to pass configuration and optional callback function pointer to the DLL/Library.

4. Use the PSNStartStopCollect() function with the start collection parameter to start the data collection.

5. Use the PSNGetBoardData() or Callback functions to receive the data.

6. When done the user should again call the PSNStartStopCollect() with the stop collection parameter.

7. Close the DLL/Library using the PSNCloseBoard() function.

Windows DLL Import Function Call Information:

Below is the information needed to use the PSNADBoardx.DLL functions. Starting with version 1.2, the DLL entry functions use the standard C calling convention. Previous version used the C++ calling convention. This should make it easier to interface the DLL with different compliers. Also, the *__stdcall* keyword has been added so the DLL can be used with Visual Basic. To use the DLL in C++ projects you will need to include the following so you can link to the DLL functions. See the AdcDemo example included in the zip file for more information.

```
#ifdef __cplusplus
```

```
extern "C" {
#endif

/* Imported DLL functions */
#define PSNADBOARD_API __declspec(dllimport)

/* Opens the ADC board and returns a HANDLE that is then
used by the other functions */
PSNADBOARD_API HANDLE __stdcall PSNOpenBoard();
....

#ifdef __cplusplus
}
#endif
```

Linux Library Function Call Information:

The Linux library was compiled using g++. To allow CPP applications to link to the library the following calling convention should be used.

```
#ifdef __cplusplus
extern "C" {
#endif

/* Opens the ADC board and returns a HANDLE that is then
used by the other functions */
HANDLE PSNOpenBoard();
....
#ifdef __cplusplus
}
#endif
```

The zip file containing the library has the following files AdcDemo.c, AdcDemo.cpp and Makefile to show how to build C and CPP applications under Linux.

Structure and Data Size Information:

Please note the structures documented below where compiled in the DLL and Linux library using BYTE aliment structure member packing. In Visual C++, and g++ under Linux, the compiler directive #pragma pack(1) was used to compile the program. This means that all members of a structure are BYTE packed and no byte padding is used to align structure members on even byte, word or long boundaries. The user must make sure that the structure size in your application matches the size the DLL/Library is using. By default C compilers, and maybe other language compilers, align structure members in either WORD or LONG boundaries. Here is some other information you will need to use the function calls:

TRUE is defined as a non-zero number, usually a 1
FALSE is defined as a zero number

HANDLE is defined as a 32 bit void pointer
PSNADBOARD_API is defined as __declspec(dllexport)
(Needed to properly import the DLL functions)
BYTE = unsigned 8 bit number (0-255)
char = signed 8 bit number (+-127)
short = signed 16 bit number (+-32767)
WORD = unsigned 16 bit number (0-65536)
long, SLONG or LONG = signed 32 bit number
DWORD or ULONG = unsigned 32 bit number
BOOL = unsigned 32 bit number used to indicate TRUE or FALSE

DLL and Linux Library Function Descriptions:

PSNADBOARD_API HANDLE __stdcall PSNOpenBoard(void)

Purpose: Used to open the DLL/Library and get a HANDLE to the ADC board

Input Parameters: None

Returned Value: Returns a HANDLE used by the DLL/Library or zero if an error has occurred

Notes: The returned HANDLE is used as the first parameter to all of the other DLL/Library function calls.

PSNADBOARD_API BOOL __stdcall PSNCloseBoard(HANDLE)

Purpose: Used to close the DLL/Library

Input Parameters: HANDLE to the board returned by PSNOpenBoard()

Returned Value: Returns FALSE if an error has occurred or TRUE if no error has occurred

Notes: The user should call this function before the application shuts down.

PSNADBOARD_API BOOL __stdcall PSNConfigBoard (HANDLE, AdcBoardConfig *,
void (*callback)(DWORD, void *, void *, DWORD))

Purpose: Used to send configuration information to the DLL/Library. Also supplies the optional Callback function pointer when the Callback mode of data collection is used.

Input Parameters:

HANDLE to the board returned by PSNOpenBoard()

Pointer to an AdcBoardConfig structure. See below.

Pointer to the Callback function or zero (NULL) if the poll mode is used

Returned Value: Returns FALSE if an error has occurred or TRUE if no error has occurred

Notes: This function must be called before the PSNStartStopCollect() function is called.

AdcBoardConfig2   Structure: Size = 316 Bytes

```
struct  {
    ULONG CommPort;
    ULONG CommSpeed;
    ULONG NumberOfChannels;
    ULONG SampleRate;
    ULONG TimeRefType;
    ULONG AddDropTimer;
    ULONG WwvPulseWidth;
    ULONG Mode12BitFlags;
    ULONG HighToLowPPS;
    ULONG NoPPSLedStatus;
    ULONG CheckPCTime;
    ULONG SetPCTime;
    ULONG TcpPort;
    SLONG AddDropTimerMode;
    SLONG TimeOffset;
    char CommPortTcpHostStr[256];
} AdcBoardConfig2;
```

Structure Member Information:

CommPort:

Comm port number to use to communicate with the ADC board. Can be 1 to 255. Under Windows this parameter should be filled in with the port number. If 0 (zero), and if there is a string in the *CommPortTcpHost* parameter, the DLL will attempt to open a TCP/IP connection to the ADC board through a user supplied RS-232 to TCP bridge connected between the ADC board and your network.

Under Linux this parameter is used as a flag or switch to tell the Library to use a RS-232 Comm Port or TCP/IP connection. If 0 the Library will open a Comm Port or USB Virtual RS-232 Port and if set to 1 the Library will attempt to open a TCP/IP connection to the TCP/IP bridge device connected to the ADC board. See *CommPortTcpHost* below for more information.

CommSpeed:

Baud rate to use when communication with the ADC board. Can be 4800 (V1 only), 9600, 19200, 38400 or 57600 (V2, V3 and VM boards only)

NumberChannels:

The number of channels to record. Can be 1 to 8 except for V1 boards running at 200 SPS. V1 boards at 200 SPS can record up to 4 channels. VM board can record 1 or 2 channels. V3 boards can record up 4 channels at 500 SPS.

SampleRate:

Sample rate in samples per seconds. V1 board can record data at 5, 10, 20, 25, 50, 100 and 200 SPS. V2 boards can record data at 10, 20, 50, 100 and 200 SPS and the VM board 5, 10, 20, 25, 40, 50 and 80 SPS. V3 board can record data at 5, 10, 20, 25, 50, 100, 200, 250 and 500 SPS

TimeRefType:

Time Reference Type.

- 0 = Use PC Time
- 1 = Garmin GPS 16, 18, or 18x - Supported by all boards
- 2 = ONCORE NMEA output mode - Supported by V1, V2 and V3 boards
- 3 = ONCORE binary output mode - Supported by V1, V2 and V3 boards
- 4 = WWV Mode - Supported by V1 and V2 boards
- 5 = WWVB mode - Supported by V1 and V2 boards
- 6 = SKG GPS Receiver - Supported by V3 boards with V2.x firmware, PSN-ACCEL and PSN-ADC24 boards

AddDropTimer:

This number is used to adjust the time on the ADC board to compensate for the time reference oscillator being a little off frequency. This number is calculated by the DLL/Library (V2/V3/VM board) or the ADC board (V1 board). The user should set this value to 0 if unknown and let the DLL//Library or the ADC board (V1 only) calculate the average pulse width. An ADC_SAVE_TIME_INFO message will be sent to the application when this parameter is calculated. The user should save this number in a file or system registry and supply the number back to the DLL at program startup. The AdcDemo.exe and C source file show one method on how to do this.

WwvPulseWidth:

This is the average pulse width in milliseconds of the WWV top of the minute signal. This number is calculated by the DLL (V2/V3/VM board) or the ADC board (V1 board). The user should set this value to 0 if unknown and let the DLL/Library or ADC board (V1 only) calculate the average pulse width. An ADC_SAVE_TIME_INFO message will be sent to the application when this parameter is calculated. The user should save this number in a file or system registry and supply the number back to the DLL/Library at program startup. The AdcDemo.exe and C source file show one method on how to do this.

Mode12BitFlags:

This is a BYTE bitmap (one bit per channel) to set the ADC channel's output mode to 12-Bits. In this mode, the data for the channel is divided by 16. This can be used when recording data from a noisy source where a lot of the lower ADC bits are filled with noise.

HighToLowPPS:

V2 and V3 boards only. If set to TRUE the ADC board will use the high to low PPS signal transitions to indicate the top of the second. If FALSE, a low to high PPS signal will be used to indicate the top of the second.

NoPPSLedStatus:

If set to TRUE, the LED on the ADC board will stay on rather then blink at a 1PPS rate. With a high gain amplifier near the ADC board the 1PPS may show up in the data. If this is a problem, the user can turn off the status blinking to prevent the problem.

CheckPCTime:

If set to TRUE, the DLL/Library will periodically check the time difference between the ADC board and the PC's time. An ADC_MSG message will be sent to the application with the time difference. This function is disabled when using the Use PC Time (TimeRefType = 0) mode.

SetPCTime:

If set to TRUE, the DLL/Library will periodically check the time difference between the ADC board and the PC's time. The PC's time will be set to the ADC time if the difference between the ADC board and PC is greater then 50 milliseconds. This function is disabled when using the Use PC Time (TimeRefType = 0) mode.

TcpPort:

Port number for TCP/IP Connections. This number should be in the range of 1 to 65535. Only used for TCP/IP connections to the ADC board.

AddDropTimerMode:

This parameter is used to adjust the time on the ADC board to compensate for the time reference oscillator being a little off frequency. This parameter is calculated by the DLL/Library (V2/V3/VM board) or the ADC board (V1 board). The user should set this value to 0 if unknown and let the DLL/Library or ADC board calculate the average pulse width. An ADC_SAVE_TIME_INFO message will be sent to the application when this parameter is calculated. The user should save this number in a file or system registry and supply the number back to the DLL/Library at program startup. The AdcDemo.exe and C source file show one method on how to do this.

TimeOffset:

Offset in milliseconds to add or subtract to the time reference. V1 board will use this value for all time reference modes. V2/V3/VM boards currently only use this for WWV timing. In this time reference mode, the users should use a value of 25ms plus the time for the WWV signal to arrive at the receiver.

CommPortTcpHost:

Comm port or TCP/IP host string used to connect to the ADC board. For RS-232 Comm Ports this field should be NULL under Windows or the Comm Port string under Linux. Linux uses a string like /dev/ttys0 or /dev/ttys1 etc to identify a Comm Port. To identify a USB Virtual Comm Port port use /dev/ttyUSB0 or /dev/ttyUSB1 etc.

If the ADC board is connected to a TCP/IP to RS-232 bridge adapter the user can connect to the bridge using a TCP/IP host name or IP address. The user will also need to specify the connection port number using the TcpPort parameter above.

Under Windows the user must set the *CommPort* number parameter above to 0 (zero) and fill in this field with the IP address or host name string to open a TCP/IP connection to the RS-232 bridge.

Under Linux the *CommPort* parameter above is used to tell the Library to open a Comm Port or make a TCP/IP connection to the RS-232 bridge. To open a Comm Port set *CommPort* = 0 or set it to 1 for a TCP/IP connection.

PSNADBOARD_API BOOL __stdcall PSNStartStopBoard(HANDLE, DWORD)

Purpose: Used to start and stop data collection.

Input Parameters:

HANDLE to the board returned by PSNOpenBoard()

DWORD 1 = Start data collection, 0 = Stop collection.

Returned Value: Returns FALSE if an error has occurred or TRUE if no error has occurred

Notes: This function must be called after the PSNConfigBoard() function is called.

PSNADBOARD_API BOOL __stdcall PSNGetBoardInfo(HANDLE, DWORD, void *)

Purpose: Used to retrieve information stored in the DLL/Library.

Input Parameters:

HANDLE to the board returned by PSNOpenBoard()

DWORD Retrieve Data Type. Can be 0, 1 or 2

void pointer to memory used to receive the data

Returned Value: Returns FALSE if an error has occurred or TRUE if no error has occurred

Retrieve Types:

0 = Get ADC Board Type. Void pointer should point to a DWORD to receive the board type. Possible returned values:

- 0 = Board type is unknown
- 1 = Version 1 ADC board
- 2 = Version 2 ADC board
- 3 = VolksMeter Interface board
- 4 = Version 3 ADC board
- 5 = PSN-ACCEL or PSN-ADC24 Board

1 = Get DLL/Library Version. Void pointer should point to a DWORD to receive the DLL/Library version number.

2 = Get DLL/Library Information. Void pointer should point to a DLLInfo structure.

DLLInfo Structure: Size = 28 Bytes

```
struct  {
    DWORD MaxInQueue;
    DWORD MaxUserQueue;
    DWORD MaxOutQueue;
    DWORD CrcErrors;
    DWORD UserQueueFullCount;
    DWORD XmitQueueFullCount;
    DWORD CpuLoopErrors;
} DLLInfo;
```

Structure Member Information:

MaxInQueue:

This is the maximum Comm Port input buffer queue size.

MaxUserQueue:

This is the maximum user data queue size. This number is only valid in the Poll Data mode.

MaxOutQueue:

This is the maximum output transmit queue to the ADC board.

CrcErrors:

Number of incoming packet CRC errors.

UserQueueFullCount:

This counter is incremented if the user data queue overflows.

XmitQueueFullCount:

This counter is incremented if the outout data queue overflows.

CpuLoopErrors:

Indicates a problem with the V2/V3 board's timing loop.

3 = Get Number of Channels. Void pointer should point to a DWORD to receive the number of channels on the ADC board.

Note: V1, V2, and V3 boards will always return 8. The VolksMeter board can have one or two channels. The PSN-ACCEL and PSN-ADC24 boards one to four channels.

4 = Get Last Error Number. Void pointer should point to a DWORD to receive the last error number.

Possible return Values:

- 0 = No Error
- 1 = Error opening Comm Port
- 2 = No configuration information
- 3 = Thread start error
- 4 = Too many board handles
- 5 = Internal Error
- 6 = No ADC Control - Indicates a problem allocating memory

- 7 = Configuration Error

PSNADBOARD_API BOOL __stdcall PSNSendBoardCommand(HANDLE, DWORD , void *)

Purpose: Used to send various commands to the DLL/Library and/or the ADC board.

Input Parameters:

HANDLE to the board returned by PSNOpenBoard()

DWORD Command Type. Can be one of the types below

void pointer to memory to send option data to the DLL/Library

Returned Value: Returns FALSE if an error has occurred or TRUE if no error has occurred

Notes: All of the commands below, except the *Reset ADC Board*, should be sent after the start data collection command (using *PSNStartStopBoard())* has been issued. The *Reset ADC Board* command can be sent after using the *PSNConfigBoard* function.

Command Types:

0 = Exit Command - Sends an exit command to the ADC board. Void pointer is not user.

1 = Send Status - Tells the DLL/Library to send a ADC_STATUS message. Void pointer is not used.

2 = Reset GPS - Sends a command to the ADC board to reset the GPS receiver. Void pointer is not used.

3 = Force GPS Time Test - Sends a command to the ADC board to force a GPS time test. Void pointer is not used.

4 = Clear Counter - Used to clear various counters in the DLL/Library and ADC board. All of the members in the DLLInfo structure above are cleared with this command. Void pointer not used.

5 = GPS Data On/Off - Turns sending GPS data on or off. When on, raw GPS data will be sent to the application using the ADC_GPS_DATA message. The void pointer parameter is used to turn the data on or off. Data pass to the DLL/Library should be a 0 (zero) to turn off GPS data and 1 to turn on GPS data. The parameter should be passes as a DWORD and not as a pointer to a DWORD. By default this mode is turned off on the ADC board.

6 = GPS Echo Mode - Sends a command to the ADC to enter the Echo GPS mode. In this mode any data sent to the ADC board is sent out the COMM port and any data sent to the COMM port by the host will be sent to the GPS receiver. In the mode ADC sample data is not collected. To exit this mode the users must send three ESC (0x1b) characters to the GPS receiver. Void pointer is not used.

7 = Reset ADC Board - Toggles the Comm port's DTR line to reset the CPU on the ADC board. Void pointer is not used.

8 = Go to Bootloader - Sends a command to the V2/V3/VM board to go to the Bootloader stored in Flash memory. Once in this mode the ADC board's firmware can be uploaded to the PICC CPU. Void pointer is not used.

9 = Send Time Information - Used to send an AdjTimeInfo structure to the V2, V3,

VM or ADC24 board. This data is sent back to the DLL/Library during the startup dialog between the ADC board and DLL/Library.

AdjTimeInfo Structure: Size = 5 Bytes

```
struct {
    ULONG AddDropTimer;
    BYTE AddTimeFlag;
} AdjTimeInfo;
```

Structure Member Information:

AddDropTimer:

AddDrop time number.

AddTimeFlag:

Add time flag.
1 = Add time
0 = Drop Time

70 = Set ADC Gain and Reference - Used to set the gains on the PSN-ACCEL and PSN-ADC24 boards. The void pointer should point to a AdcConfig structure documented in the ADC24 information below.

PSNADBOARD_API DWORD __stdcall PSNGetBoardData(HANDLE, DWORD *, void *, void *, DWORD *)

Purpose: Used in the Poll Data mode to retrieve data from the Windows DLL or Linux library.

Input Parameters:

HANDLE to the board returned by PSNOpenBoard()

DWORD pointer to M*essage Type* variable

void pointer to local memory to receive data void pointer to local memory to receive data (optional)

DWORD pointer to *Data Length* variable

Returned Values: 0 = Error, 1 = No New Data, 2 = New Data

Notes: The user should call this function in the Poll Data mode. If the function returns a 2, new data is available to the application. The data return, and what void points are used to return data, depends on the Message Type. The void pointers need to point to some local memory that is large enough to hold the maximum data length of the various message types. The first pointer should point to an BYTE array of 1024 bytes and the second void pointer should point to an array of shorts that can hold one second worth of ADC sample data. This works out to a maximum array size of 3200 bytes or 1600 shorts.

Message Types:

0 = General Text Message from the DLL/Library. The text string, a standard C string that is terminated with a NULL at the end, is passed in the first void pointer. The Data Length variable will contain the length of the string including the null character. The second void pointer is not used.

1 = Error Text Message from the DLL/Library. The text string is passed in the first void pointer. The Data Length variable will contain the length of the string including the null character. The second void pointer is not used.

2 = ADC Text Message. The ADC board generates these messages. The text string is passed in the first void pointer. The Data Length variable will contain the length of the string including the null character. The second void pointer is not used.

3 = ADC Sample Data Message. This message type indicates new sample data and header information has been placed in the users memory data buffers. The first void pointer receives the *DataHeader* information structure and second void pointer receives the ADC sample data. The Data Length variable will contain the length in bytes of the

sample data. The *DataHeader* size is fixed, so there is no need to know the length of this data item.

The sample data should be treated as an array of shorts (signed 16-Bit numbers) for the V1, V2 and V3 boards and an array of longs (signed 32-Bit numbers) for the VolksMeter board. The number of samples is one-second worth of data from the ADC board and will vary depending on the sample rate and number of channels being recorded. The format of the sample data is as follows if recording two channels:

```
Sample 1 Channel 1
Sample 1 Channel 2
Sample 2 Channel 1
Sample 2 Channel 2
...
```

Three channels would look this this:

```
Sample 1 Channel 1
Sample 1 Channel 2
Sample 1 Channel 3
Sample 2 Channel 1
Sample 2 Channel 2
Sample 2 Channel 3
...
```

4 = GPS Data Message. This message type will received if the ADC is instructed to send GPS data to the host using the *PSNSendBoardCommand()*. Raw GPS data will be placed in the first void pointer memory array and the Data Length variable will reflect the size of the data block. The second void pointer is not used. Data received is either in the NMEA or ONCORE binary formats depending on the receiver type.

5 = Statues Message. This message is sent in response to the Send Status (Type 1) command using the *PSNSendBoardCommand()* function call. The first void pointer will receive a *StatusInfo* structure. The Data Length variable will reflect the size of the structure. The second void pointer is not used.

6 = Save Time Information Message. This message will be periodically called by the DLL/Library to tell the application to save AddDrop Timer information to a file or

in the system's registry. This information should be supplied to the DLL/Library at program startup. See the *PSNConfigBoard()* function call for more information. The first void pointer memory array will contain a *TimeInfo* structure and the Data Length variable will contain the size of this structure. The second void pointer is not used.

The following data structures are used by this function.

DataHeader Structure: Size = 22 Bytes

```
struct  {
    SYSTEMTIME PacketTime;
    ULONG PacketID;
    BYTE TimeRefStatus;
    BYTE Flags;
} DataHeader;
```

Structure Members:

packetTime:

Time of the first sample in the sample data array.

PacketID:

Data message ID. This ID number will increment with each new data block sent by the DLL/Library. The user can check this number to see is any data messages are missing.

TimeRefStatus:

Indicates the status of the time reference. This member will be 0 (zero) if the time reference is not locked to the time reference. A 1 indicates that the time is not locked to a time reference, but was previously locked a few hours ago. A 2 indicates the time on the ADC board is locked to the time reference.

Flags:

Currently three of the 8 bits are used to indicate the ADC board type.

V1 Board: Bit 1 = 0, Bit 7 = 0 and Bit 8 = 0

V2 Board: Bit 1 = 0, Bit 7 = 0 and Bit 8 = 1

V3 Board: Bit 1 = 1, Bit 7 = 0 and Bit 8 = 1

VM Board: Bit 1 = 0, Bit 7 = 1 and Bit 8 = 0

ADC24: Bit 0 = 1, Bit 7 = 1 and Bit 8 = 1

SYSTEMTIME Structure: Size = 16 Bytes

```
struct _SYSTEMTIME {
    WORD wYear;
    WORD wMonth;
    WORD wDayOfWeek;
    WORD wDay;
    WORD wHour;
    WORD wMinute;
    WORD wSecond;
    WORD wMilliseconds;
} SYSTEMTIME;
```

This is a standard Windows system time structure. The DLL/Library does not use the wDayOfWeek member.

TimeInfo Structure: Size = 20

```
struct  {
    char AddDropMode;
    BYTE Flags;
    WORD WwvPulseWidth;
    long AddDropTimer;
    long TimeLocked;
    long AdjustNumber;
    short TimeDifference;
    short TimeOffset;
} TimeInfo;
```

Structure Member Information:

AddDropMode:

Indicates if time is being added to, or subtracted from, the ADC time accumulator.

Flags:

Not used.

WwvPulseWidth:

Current WWV top of the minute

AddDropTimer:

Current Add / Drop timer interval.

TimeLocked:

The number of seconds the ADC board has been locked to the time reference.

AdjustNumber:

Number of milliseconds that have been added to, or subtracted from, the ADC time accumulator.

TimeDifference:

Current time difference in milliseconds between the

ADC time accumulator and
the time reference.

TimeOffset:

Current offset time in
milliseconds being added to,
or subtracted from, the ADC
board time.

StatusInfo Structure: Size = 47 Bytes

```
struct  {
    BYTE BoardType;
    BYTE MajorVersion;
    BYTE MinorVersion;
    BYTE LockStatus;
    BYTE NumberOfChannels;
    WORD SpsRate;
Note: 1
    ULONG CrcErrors;
    ULONG PacketsSent;
    ULONG RetransmissionCount;
    ULONG RetransmissionErrors;
    ULONG PacketsReceived;
    TimeInfo timeInfo;
} StatusInfo;
```

Note: 1 Starting with version 3.3 of the DLL/Library the SpsRate rate member has changed from a BYTE to a WORD

Structure Member Information:

BoardType:

ADC Board type. 0 = SDR
Server, 1 = V1 Board, 2 = V2
Board and 3 = VM Board

MajorVersion:

Major firmware software
version number running on
the ADC board.

MinorVersion:

Minor firmware software version number running on the ADC board.

LockedStatus:

Indicates the current time reference lock status. This member will be 0 (zero) if the time reference is not locked to the time reference. A 1 indicates that the time is not locked to a time reference, but was previously locked a few hours ago. A 2 indicates the time on the ADC board is locked to the time reference.

NumberOfChannels:

Current number of channels being recorded by the ADC board.

SpsRate:

Current sample rate.

CrcErrors:

Number of incoming CRC errors.

PacketsSent:

Number of packets sent by the ADC board.

RetransmissionCount:

Number of retransmission processed. V1 board only.

RetransmissionErrors:

Number of retransmission errors. V1 board only.

PacketsReceived:

Number of data packets received by the DLL/Library.

TimeInfo:

See the TimeInfo structure above.

PSN-ACCEL / PSN-ADC24 Information:

Our new PSN-ACCEL Strong Motion Senor board and PSN-ADC24 digitizer board uses an addition structure to set the programmable gain on the Sigma-Delta ADC chips used on the sensor board. The gain on the ADC chips can be set to: 1 (default), 2, 4, 8, 16, 32 or 64. This structure is sent to the board using the *PSNSendBoardCommand()* function with the Command Type set to 70 decimal. This command should be sent to the DLL/Library after calling *PSNOpenBoard()* and before calling the *PSNConfigBoard()* function.

AdcConfig Structure: Size = 12 Bytes

Structure Member Information:

```
struct  {
     double ReferenceVolts;
     BYTE AdcGains[ 4 ];
} AdcConfig;
ReferenceVolts:
      Currently not used by the
      DLL/Library or the PSN-ACCEL/PSN-
      ADC24 boards.
AdcGains:
      This array of 4 bytes is used to
      set the ADC chip Gain. It can be:
      1, 2, 4, 8, 16, 32 or 64
```

-End of Document