The AMTI USB Device Software Development Kit

Reference Manual

Version 1.1

Notice

AMTI does not warrant that the AMTI USB Device DLL software will function properly in every hardware software environment. This software is inherently complex, and users are cautioned to verify the results of their work.

AMTI has tested the software and reviewed the documentation. AMTI MAKES NO WARANTY OR REPRESENTATION, EITHER EXPRESSED OR IMPLIED, WITH RESPECT TO THIS SOFTWARE OR DOCUMENTATION, THEIR QUALITY, PERFORMANCE, MERCHANTIBILITY, OR FITNESS FOR A PARTICULAR PURPOSE. AS A RESULT, THIS SOFTWARE AND DOCUMENTATION ARE LICENSED “AS IS” AND YOU, THE LICENSEE ARE ASSUMING THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE.

IN NO EVENT WILL AMTI BE LIABLE FOR DIRECT, INDIRECT, SPECIAL, OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE SOFTWARE OR DOCUMENTATION, even if advised of the possibility of such damages. In particular, AMTI shall have no liability for any programs or data stored or used with AMTI software, including the costs of recovering such programs or data.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

Redistributions of source code must retain the above copyright notice, this list of conditions and the above disclaimer.

Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the above disclaimer listed in this license in the documentation and/or other materials provided with the distribution.

Neither the name of the copyright holders nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

AMTI Contact Information

Advanced Mechanical Technology
176 Waltham St.
Watertown, MA 02478
Phone (617)926-6700
Website www.amti.biz
Email helpdesk@amtimail.com

1.0 Introduction

The AMTI USB Device Software Development Kit (SDK) is designed to assist third party vendors integrate one or multiple AMTI Gen 5 signal conditioners into their applications. The AMTI SDK allows vendors to communicate with AMTI hardware through a USB 2.0 interface.

The SDK consists of a regular dynamic linking library (DLL) named AMTIUSBDevice.dll, a library file named AMTIUSBDevice.lib, and a header file named AMTIUSBDevice.h. To get started you must include these three files in your project as you would when integrating any regular DLL.

The DLL was written in Visual C++, Visual Studio 2008. It is for the Win 32 platform. It is compatible with Windows 7 Wow64. The reader is expected to be familiar with Dynamic Link Libraries and their uses.

2.0 The Gen 5 Signal Conditioner

The overall function of the GEN 5 Signal Conditioner is to condition data from six strain gauge inputs and output the results as six analog channels and/or a six channel digital data stream. The analog outputs are high level and suitable as inputs to a multi-channel Analog to Digital Converter (ADC). The digital data are transmitted to a host Personal Computer (PC) via a Universal Serial Bus (USB) connection. The USB port is also used to send and receive control and status information used by the GEN 5.

The overall GEN 5 function can be divided as follows:

1. Provide analog signal conditioning for six strain gauge inputs including production of six independently selectable strain gauge excitation voltages, bridge balancing with independently selectable offsets, filtering and amplification at independently selectable gains.

2. Perform periodic sampling of the six conditioned analog signals at selectable rates.

3. Perform numerical processing of digitized signals including conversion to engineering units.

4. Convert numerically processed data to high-level analog signals suitable for an ADC via a Digital to Analog Converter (DAC) and analog signal conditioning.

5. Provide an industry standard USB port for data transmission and reception.

6. Provide non-volatile memory for the storage of calibration and configuration data.

7. Provide for the reading of calibration coefficients and other data from AMTI Smart Platforms equipped with Read Only Memory (ROM).

The above functions are implemented by the GEN 5 with analog circuitry and two MCU's. A Silicon Laboratories C8051F120A mixed signal MCU with FLASH and its peripheral circuits perform all functions except the USB port implementation. A Cypress Semiconductor Corporation CY7C68013A-128AC (EZ-USB FX2LP) single-chip USB MCU implements the industry standard USB port.

3.0 Software Development Strategy

When considering integrating the AMTI SDK with your application there are two options, full integration or partial integration.

Full integration would involve integrating most of the features of this SDK into the application. This would give the application full control of the signal conditioners.

Partial integration involves integrating only data collection portion of the SDK.

The AMTI System Configuration program is a utility program which ships with every Gen 5. It is used to setup and configure both the DLL and the Gen 5's. For a partial integration strategy decide to use the AMTI System Configuration program for signal conditioner setup and configuration. Then only integrate the data collection processes into the third party application. To do this you need only familiarize yourself with the sections: Initializing and Configuring the DLL, and Data Collection. For most users this will be the way to go.

4.0 LabView Compatibility

When integrating this DLL with LabView, AMTI recommends using the program Gen 5 Configuration for setting up and configuring the DLL and Gen 5's. Then only integrate the data collection processes into the LabView application. To do this you need only familiarize yourself with the sections: Initializing and Configuring the DLL, and Data Collection.

We recommend using the polling method of data collection. For the data transfer function the ***fmDLLGetTheFloatDataLBVStyle*** function must be used.

AMTI does provide starter source code for a simple LabView data collection program.

The recommended data collection method above has been tested for LabView compatibility.

5.0 Definitions

AD – Analog to digital conversion

DAC – Digital to analog conversion

MCU – microcontroller unit

Platform – In this manual the word platform may also be substituted with transducer, load cell, any six channel strain gage multi-axis measurement device.

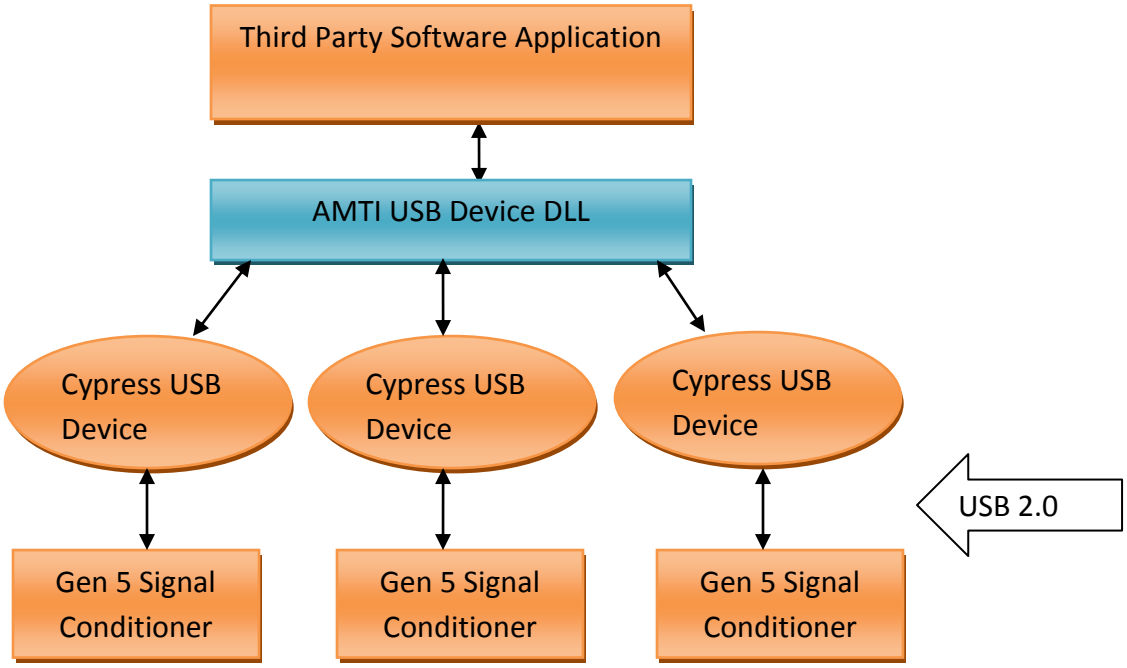
Electrical range – In this manual electrical range is the maximum and minimum measurement capacity of the signal conditioner expressed as engineering units.

Analog output range – For a Gen 5 the analog output range is always +- 5 volts. When we discuss the anlog output range in this manual we are frequently refferring to it in engineering units.

6.0 Software System Overview

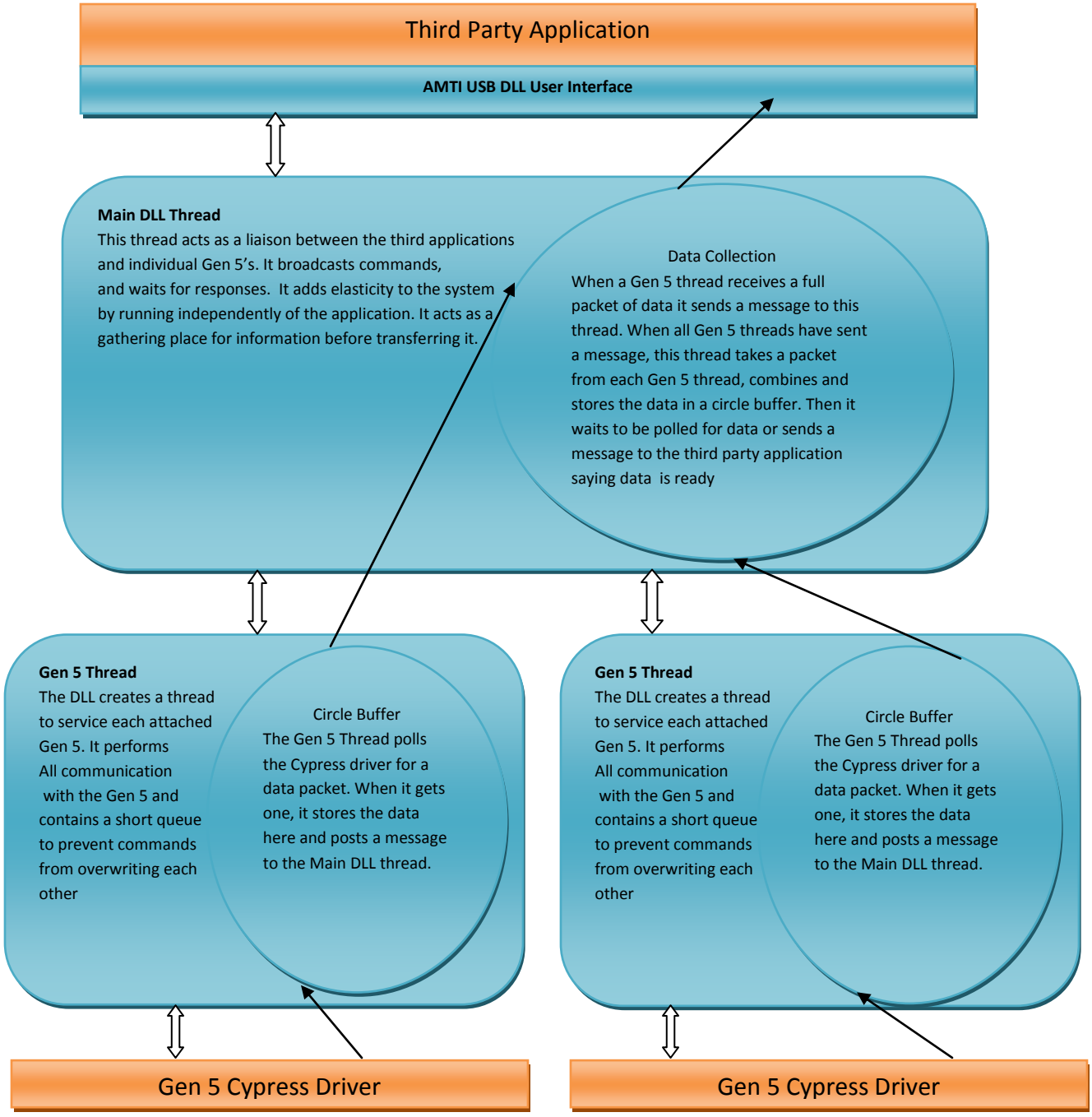
The diagram below illustrates the relationship between the AMTI USB Device DLL and the rest of the system. The DLL handles all communication between the third party application and the Gen 5 Cypress device drivers. When initialized the DLL will find and communicate with each Gen 5. Each Gen 5 is connected to the PC through a USB 2.0 port.

Figure 1 - Software System Overview



As illustrated in Figure 2, the DLL creates a thread to act as a liaison between the third party application and all of the Gen 5 signal conditioners. The DLL spawns additional threads to service each connected Gen 5.

Figure 2 – Software System Architecture



6.1 Understanding the Different Function Types

There are three different types of functions in the SDK. The function prefix distinguishes them.

Function Prefix	Function Prefix Meaning
fmBroadcast	The ***fmBroadcast*** prefix indicates the function is a global command. The function broadcasts the command to all currently connected signal conditioners.
fmDLL	The ***fmDLL*** prefix indicates the function has to do with the current configuration settings of the DLL, not the Gen 5 signal conditioners.
Fm	The ***fm*** *prefix,* excluding the ***fmBroadcast*** and ***fmDLL*** types, is concerned with only one Gen 5. In order to communicate with a specific signal conditioner the function ***fmDLLSelectDeviceIndex*** must be called to select that signal conditioner. The selected signal conditioner remains the selected signal conditioner until another signal conditioner is selected or the DLL terminates.

6.2 Selecting a Device

Before communicating with a specific Gen 5, the device must be selected first. The ***fmDLLSelectDeviceIndex*** function selects the Gen 5 by its device index. Once a signal conditioner has been selected you may communicate with it through of the ***Fm*** prefix type functions.

The device indexes are ordered 0 to the number of Gen 5's minus one. They are always ordered in the platform data collection order stored in the DLL configuration file. Use f***mDLLGetDeviceCount*** to know how many devices are connected.

To find out what signal conditioner is associated with a device index call GetAmplifierSerialNumber.

6.3 Understanding the Gen 5 Channel Order

The Gen 5 is a 6 channel data collection device. It collects data from force plates which measure forces and moments. The channel order is always the three forces followed by the three moments.

Fx – The force vector along the x axis of a platform
Fy – The force vector along the y axis of a platform
Fz – the force vector along the z axis of a platform
Mx – The moment around the x axis of a platform
My – The moment around the y axis of a platform
Mz – The moment around the z axis of a platform

Channel	Forces			Moments		
Index	0	1	2	3	4	5
Row	Fx	Fy	Fz	Mx	My	Mz

When uploading or downloading parameters the channel order as shown in the above table is always maintained. If we are uploading or downloading a table with multiple entries for each channel the channel order is still maintained as in the following table where each channel has three entries. The table is always packed in row, column order.

Index	0	1	2	3	4	5
Row 1	Fx	Fy	Fz	Mx	My	Mz
Index	6	7	8	9	10	11
Row 2	Fx	Fy	Fz	Mx	My	Mz
Index	12	13	14	15	16	17
Row 3	Fx	Fy	Fz	Mx	My	Mz

6.4 Applying and Saving Parameters

There are multiple functions for applying and saving parameters. To avoid confusion they are each described below.

Applying Gen 5 Signal Conditioner Settings

When new calibration tables or configuration parameters are downloaded to the Gen 5's they are not automatically applied. The functions ***fmBroadcastResetSoftware*** and ***fmResetSoftware*** are used to apply Gen 5 parameters. The exceptions to this rule are the start, stop, zero, blink and set acquisition rate commands. Additionally these two functions do not apply to the DLL configuration settings.

The two reset software functions require a time delay after being called. The internal Gen 5 software resets itself and the signal conditioner will not accept commands while resetting.

It is suggested the user makes all of their configuration changes and then call these functions when done.

The reset software functions do not save the parameter changes to permanent flash memory.

Saving Gen 5 Signal Conditioner Settings

Each Gen 5 signal conditioner maintains its own calibration tables, platform calibration tables, and current configuration settings all within its own internal flash memory. To save the current Gen 5 configuration settings to permanent flash memory the functions ***fmBroadcastSave*** or ***fmSave*** must be used.

Saving the DLL Configuration Settings

To save the current DLL settings you must call ***fmDLLSaveConfiguration***. For a full list of DLL configuration settings please see the section titled, The DLL Configuration File.

7.0 Initializing and Configuring the DLL

Initializing the DLL

The first DLL function called in your application will always be ***DLLInit***.

The ***DLLInit*** function does the following:

1) The DLL searches for the signal conditioners. When a signal conditioner is found it uploads all of the signal conditioner parameters and stores them in local memory. These parameters include both calibration tables and configuration settings. It does this so it can retrieve parameters instantly without having to query the signal conditioner. Additionally all future parameter changes update both DLL memory and signal conditioner memory.

2) The DLL then loads the configuration file. It compares the saved configuration to the current configuration. It checks to see that the same number of signal conditioners is present as the last time it was run. It compares the serial numbers and makes sure the serial numbers match. It sets up the data collection order of the platforms to insure the data is presented in the same platform order as saved in the last configuration. If some Gen 5's are not present the platform order will be maintained.

3) When the DLL has completed initializing it will set a flag. The flag can be checked by calling ***fmDLLIsDeviceInitComplete***.

4) After the DLL has finished initializing call ***fmDLLSetupCheck***. That will return an error number to alert you to configuration differences between the current configuration and the last saved configuration.

The DLL Configuration File

The DLL configuration file is AMTIUSBSetup.cfg. This file is located in the C:\AMTI\CFG folder.

The configuration file maintains the last saved DLL configuration settings listed in the table below. In addition to this list it maintains the data collection order of the platforms.

Table 1 – The DLL configuration Settings

Global Settings	**Description , Range, or possible values**
The configuration file version number	12345 (current version)
The signal conditioner count	0-15 (range of possible values)

The acquisition rate	0-2000 (range of possible values)
The run mode	0-4 (Range of possible values)
The genlock state	0-2 (Range of possible values)
Signal Conditioner Settings	**Always saved in data collection order**
Signal conditioner serial number	
Signal conditioner model number	
Platform serial number	
Platform model number	

The acquisition rate, run mode and genlock settings stored in the configuration file are the last broadcast settings downloaded before the configuration file was last saved. It does not mean that all of the signal conditioners are configured to these settings. To be sure all signal conditioners are configured to the same settings it is recommended that these setting be re – broadcast after the DLL has initialized.

The function ***fmDLLSaveConfiguration*** saves the DLL configuration file. It is not automatically updated upon closing the DLL.

Re-initializing the DLL

You may use the ***DLLInit*** function to reinitialize the DLL. Simply call it again and it will re-initialize. The reason to do this is to find signal conditioners that were either unplugged or added after the application has started.

If a signal conditioner is removed while the application is running the DLL should be re-initialized. It will no longer function correctly if a signal conditioner is not present.

DLL Cleanup

In order to shut down the DLL, ***fmDLLShutDown*** must be called. Calling this function terminates all running threads and performs cleanup for the DLL. After calling this command adequate time must be allotted for cleanup before closing your application. This time increases per signal conditioner however 250 ms should be more than adequate.

If you are planning on reinitializing the DLL to search for additional signal conditioners and not terminating the application just call ***fmDLLInit*** again. Do not call ***fmDLLShutDown***.

8.0 Collecting Data

This section describes the decisions which must be made to set up the data acquisition process. Each function presents data acquisition options which must be considered. Consider each one and configure the DLL accordingly.

Choosing a Data Collection Method

There are three ways to collect digital data

1. The first is polling; simply poll the DLL continuously to see if data is available. The data transfer function will either give you data or return FALSE if no new data is available.

2. The second is to have the DLL post a message to the application main window every time data is ready. Upon receiving the message the application can use the data transfer function to receive the data. To set up for windows messaging use the functions ***fmDLLPostDataReadyMessages*** and ***fmDLLPostWindowMessages***.

3. The third is to have the DLL post a message to a user thread every time data is ready. Upon receiving the message the application can use the data transfer function to receive the data. To set up for user thread messaging use the functions ***fmDLLPostDataReadyMessages*** and ***fmDLLPostUserThreadMessages***.

Note: You cannot have the DLL post data ready messages to both a user thread and a window at the same time.

Choosing the Data Transfer Function

The data transfer functions check to see if data is available and if so return the data. There are two data transfer functions. One is developed for C programming, the other is recommended for Labview.

The function for C programming is ***fmDLLTransferFloatData***.

The function for Labview programming is ***fmDLLGetTheFloatDataLBVStyle***.

Setting the Data Format

You must decide on a data format. There are two data formats. Your application can receive each dataset as eight channel or six channel format. A six channel dataset will consist solely of

the force and moment channels. An 8 channel format will have two additional channels, a dataset counter and a trigger state. To set the data format call ***fmDLLSetDataFormat.***

Setting the Data Packet Size

The DLL collects data in packets. Currently there is only one packet size and that is 512 bytes. To set the packet size call ***fmDLLSetUSBPacketSize*** and set it to 512.

Setting the Data Units

There are two ways to receive data from the Gen 5 signal conditioner. One is through the digital outputs; the other is through analog outputs. For digital outputs the choices are bits, English units, or metric units. For analog outputs the choices are fully conditioned and MSA 6 compatible. To set the data collection type call ***fmBroadcastRunMode***.

Setting the Acquisition Rate

The DLL can collect digital data at different rates. You must call ***fmBroadcastAcquisitionRate*** to set the acquisition rate. The new acquisition rate will take affect with the next Start command.

Alternatively you may time the data collection using an external Genlock signal. See the section, Implementing a Genlock Signal, for more information. If not using the Genlock feature, you should call ***fmBroadcastGenlock*** to make sure it is off.

Zeroing the Platform

Before collecting data the platform should be zeroed in an unloaded state. The function ***fmBroadcastZero*** sends a zero command to all of the signal conditioners. This function may be called before or during acquisition.

The ***fmBroadcastZero*** function performs both a hardware zero and software tare on the platform.

Starting Acquisition

To start data acquisition the function ***fmBroadcastStart*** must be called. This function sends a start command to all connected signal conditioners.

This function only starts digital data collection. The analog outputs of the Gen 5 are always running.

When this function has been called additional start commands will be ignored until a stop command has been received.

Data collection will be automatically stopped if any other commands are sent to the signal conditioners after the start command has been broadcast. This is necessary to maintain the integrity of the synchronization between signal conditioners. The exception to this is the broadcast zero command. The zero command is the only command which may be broadcast during data collection that will not stop data collection.

Stopping Acquisition

To stop data acquisition, call the function ***fmBroadcastStop.***

Data collection will be automatically stopped if any other DLL commands are sent to the signal conditioners after the start command has been broadcast. This is necessary to maintain the integrity of the synchronization scheme between signal conditioners.

The List of Data Collection Functions

fmDLLSetUSBPacketSize
fmBroadcastRunMode
fmDLLGetRunMode
fmGetRunMode
fmBroadcastGenlock
fmDLLGetGenlock
fmBroadcastAcquisitionRate
fmDLLGetAcquisitionRate
fmGetAcquisitionRate
fmBroadcastStart
fmBroadcastStop
fmBroadcastZero
fmDLLPostDataReadyMessages
fmDLLPostWindowMessages
fmDLLPostUserThreadMessages
fmDLLSetDataFormat
fmDLLTransferFloatData
fmDLLGetTheFloatDataLBVStyle

9.0 Using the Gen 5 Configuration Functions

In order to use a Gen 5 it must be configured for use. The following table describes the configuration choices which must be made. You may have to consult the Gen 5 user manual for additional information.

Table 2 - The Gen 5 Configuration Parameter List

Current Gain	For each channel an amplifier gain must be selected. The choices are 500, 1000, 2000 or 4000.
Current Excitation	For each channel a strain gage excitation voltage must be selected. The choices are 2.5, 5.0, or 10.0 volts.
Matrix Mode	The signal conditioner can either use the full platform calibration matrix when processing data or just the main diagonal terms of the matrix.
Channel Offset	For each channel a channel offset must be set. The channel offset allows the mechanical range of the signal conditioner to be offset by +- 90 %. It must be entered as a value between -0.9 and 0.9. The default setting is always 0.
Platform Rotation	A single value entered in degrees instructing the platform to perform a coordinate transformation on the data.
Cable Length	A single value giving the length of the cable between the platform and the signal conditioner in feet.
DAC Sensitivities	For each channel, a DAC conversion value must be entered. It is used for scaling the analog outputs to a user supplied conversion factor. This factor is only applied when the analog outputs are set to fully conditioned mode.
Run Mode	A single value selecting the output modes of the signal conditioner. For digital outputs the choices are, English, metric, or bits. For analog outputs the choices are fully conditioned or MSA 6 compatible.
Acquisition Rate	The digital, data collection rate of the signal conditioner represented in datasets per second.

Genlock	A single value turning genlock mode on or off. The default is off.
Product Type	You can ask the attached device what product type it is. Currently the only Product Types are 100 for a Gen 5 and 300 for an Optima.

The following is the list of functions used to configure the signal conditioner

9.1 The List of Gen 5 Configuration Functions

fmSetCurrentGains
fmGetCurrentGains
fmSetCurrentExcitations
fmGetCurrentExcitations
fmSetMatrixMode
fmGetMatrixMode
fmSetChannelOffsetsTable
fmGetChannelOffsetsTable
fmSetPlatformRotation
fmGetPlatformRotation
fmSetCableLength
fmGetCableLength
fmSetDACSensitivityTable
fmGetDACSensitivities
fmDLLSetUSBPacketSize
fmBroadcastRunMode
fmDLLGetRunMode
fmGetRunMode
fmBroadcastGenlock
fmDLLGetGenlock
fmBroadcastAcquisitionRate
fmDLLGetAcquisitionRate
fmGetAcquisitionRate
fmGetProductType

10.0 Retrieving the Gen 5 Mechanical Limits

All force platforms have mechanical capacities which may not be exceeded. Each channel of the Gen 5 has an electrical range. Depending on the Gen 5 configuration the electrical range is mapped to different different mechanical ranges.

When a Gen 5 is first turned on it calculates the mechanical limits for each channel. The function ***fmGetMechanicalMaxAndMin*** returns the mechanical limits of the signal conditioner in engineering units.

If the functions ***fmBroadcastResetSoftware*** or ***fmResetSoftware*** are called to apply settings, the mechanical range is recalculated. To retrieve the recalculated mechanical range from the Gen 5 the function ***fmUpdateMechanicalMaxAndMin*** is required to upload the recalculated mechanical limits to the DLL. The function ***fmGetMechanicalMaxAndMin*** may then be called to retrieve the limits.

The digital output range of the signal conditioner is always the same as the Gen 5 mechanical range, not the platform capacity. The analog output range expressed in engineering units is always the same as the digital output range except when the analog output is in fully conditioned mode.

For a Gen 5 whose analog output is in fully conditioned mode the output range is scaled to a user supplied digital to analog conversion factor. The output range expressed in engineering units will always be either less than or equal to the Gen 5 mechanical range.

The analog output range in engineering units is calculated when the Gen 5 is first turned on. The function ***fmGetAnalogMaxAndMin*** returns the analog output range in engineering units. This function is indeterminate if the signal conditioner is not running in analog fully conditioned mode.

If the functions ***fmBroadcastResetSoftware*** or ***fmResetSoftware*** are called to apply settings the analog output range in engineering units is recalculated. To retrieve the recalculated range, the function ***fmUpdateAnalogMaxAndMin*** is required to upload the recalculated limits to the DLL. The function ***fmGetAnalogMaxAndMin*** may then be called to retrieve the new limits.

11.0 Determining the Platform Order

When more than one platform is installed it is important the the data is always presented in the same order. A dataset consists of a single sample of data concatenated together from each platform. The question is which platforms data should be presented first in each dataset.

The DLL allows the user to set and save the dataset platform order in the DLL configuration file. That way the dataset platform order is remembered from one session to the next. The function ***fmDLLSaveConfiguration*** saves the DLL configuration file.

There are two ways to set the dataset platform order. The platform order may be set manually or automatically.

Manually Setting the Dataset Platform Order

To manually set the platfrom order call the function ***fmDLLSetPlatformOrder***.

Auto-ordering the Dataset Platform Order

The second method for setting the dataset platfrom order is called auto-ordering. In this scenario the user steps on the platforms in the desired dataset platform order. Four functions are involved in this process: ***fmBroadcastPlatformOrderingThreshold, fmDLLStartPlatformOrdering, fmDLLIsPlatformOrderingComplete***, and ***fmDLLCancelPlatformOrdering.***

The way this works is the function ***fmBroadcastPlatformOrderingThreshold*** is called to set a load detection threshold in the DLL. When the function ***fmDLLStartPlatformOrdering*** is called the DLL goes into listen mode to detect the order in which the load detection threshold is triggered by a person stepping on each platform in the desired order. The function ***fmDLLIsPlatformOrderingComplete***, may then be called to confirm the platform ordering is complete. The function ***fmDLLCancelPlatformOrdering*** may be called at any time to cancel the operation.

The required steps for completing the platform auto-ordering are:

A. Call ***fmDLLSetDataFormat*** and set the format to parameter to 0; (6 channel vs 8 channel collection)

B. Call ***fmBroadcastRunMode*** to collect the digital data as bits.

C. Call ***fmBroadcastPlatformOrderingThreshold*** and set an appropriate platform load detection threshold in bits. The full scale bit range of the Gen 5 is always +-16384.

D. Call ***fmBroadcastResetSoftware*** to apply the changes

E. Use a Sleep command to allow the signal conditioners to reset.

F. Call ***fmBroadcastZero*** to zero the unloaded platforms.

G. Call ***fmDLLStartPlatformOrdering*** to put the DLL into listening mode.

H. Call ***fmBroadcastStart*** to start data collection. The Dll will now check all incoming platform data to detect the order in which the platform threshold is crossed. It will not stop listening until all platforms have had their threshold crossed. You can call ***fmDLLCancelPlatformOrdering*** to cancel the process.

I. The function ***fmDLLIsPlatformOrderingComplete*** is called to detect if the process has completed. We suggest either using a timer or a sleep function to periodically check for process completion.

J. Once the process has completed remember that this has changed the device index order of the signal conditioners, You will have to loop through each device and request the serial numbers to figure out the new order.

K. Call ***fmDLLSaveConfiguration*** if you would like to maintain the new order when the DLL is next initialized.

12.0 Using the Genlock Feature

Genlock is a common technique where the output of one source is used to synchronize multiple devices. The Gen 5 has a genlock input port. The function ***fmBroadcastGenlock*** is used to set the genlock state of all connected Gen 5's. When genlock is set to on the Gen 5 will collect a single dataset on either the rising or falling edge of an analog input signal, usually a square wave of some sort.

The genlock signal must be sent to all connected Gen 5's.

Important Note: If using Genlock, the signal should be started before the ***fmBroadcastStart*** command is sent and turned off after the ***FmBroadcastStop*** command is sent. After a start command is sent any data collection timeout period of more than 2 seconds causes the USB bulk transport to stall the data collection. If this event occurs the function ***fmBroadcastResetUSB*** must be called to reset the pipes.

The low state of the genlock input must be less than one volt. The high state must be greater than 3 volts but never more than 10 volts. The duration in either state must be greater than 20 microseconds to be detected.

13.0 Using the External Trigger

The Gen 5 signal conditioner has a trigger input port. The Gen 5 has eight channels of digital output one of which is the trigger signal. The function ***fmDLLSetDataFormat*** determines whether the DLL delivers the full 8 channels or only the 6 force and moment channels. To see the trigger signal the DLL must be set up to deliver the full eight channels of data. A trigger channel value of 1 indicates the trigger input is high, and a value of 0 indicates the trgger input is low.

The low state of the trigger input must be less than one volt. The high state must be greater than 3 volts but never more than 10 volts. The duration in either state should be greater than the duration between datasets.

14.0 Using the Gen 5 Calibration Functions

The Gen 5 arrives already calibrated from the factory. The signal conditioner maintains its calibration tables within its permanent flash memory. The following calibration information is stored within the Gen 5 signal conditioner. For further information about these settings refer to the Gen 5 user manual.

Table 3 – The Gen 5 Calibration Parameter List

Item	**Description**
Model Number	The model number of the signal conditioner
Serial Number	The serial number of the signal conditioner
Firmware Version	The firmware version of the signal conditioner
Calibration Date	Date the signal conditioner was last calibrated
Gain Table	A 24 element table containing the gain correction values for the 4 possible AD gain settings for each of the 6 channels.
Excitation Table	An 18 element table containing the excitation correction values for the 3 possible excitation settings for each of the 6 channels.
DAC Gains Table	A 6 element table containing the DAC gain corrections for each of the 6 analog output channels.
DAC Offsets Table	A 6 element table containing the DAC offset corrections for each of the 6 analog output channels.
ADRef	The nominal AD reference voltage

The List of Gen 5 Calibration Functions

The following functions are used to set or retrieve the calibration settings listed in the table above. Since all of these settings were set at the factory it is not recommended that you overwrite them.

fmGetAmplifierModelNumber
fmGetAmplifierSerialNumber
fmGetAmplifierFirmwareVersion
fmGetAmplifierDate
fmGetGainTable
fmGetExcitationTable
fmGetDACGainsTable
fmGetDACOffsetTable
fmGetADRef

15.0 Using the Platform Calibration Functions

The Gen 5 delivers fully processed data to the PC through the USB connection. In order to do that, it must have the calibration tables for the attached platform available to it. The Gen 5 has space allocated within its permanent flash memory for storing calibration information about the attached platform. The table below describes all of the platform calibration information the Gen 5 should maintain.

Newer AMTI platforms come with smart chips embedded in them which contain the platform calibration information. If the attached platform is a smart platform the Gen 5 will read the smart chip and load the calibration settings from it. When the Gen 5 is turned on it looks for a smart platform, if it does not find one it will use its locally saved settings.

Table 4 – The Platform Calibration Parameter List

Item	**Description**
Platform Date	Date the platform was last calibrated
Model Number	The model number of the platform
Serial Number	The serial number of the platform
Length	The length of the platform in inches
Width	The width of the platform in inches
X,Y,Z Offsets	A 3 element table giving the spatial coordinates of the X, Y and Z axis, electrical centers
Platform Capacity	A 6 element table containing the platform capacity for each of the 3 forces and 3 moments in English units.
Bridge Resistances	A 6 element table containing the strain gage bridge resistance for each platform channel in Ohms.
Inverted Sensitivity Matrix	A 36 element table containing the inverted sensitivity matrix. The matrix must be the English version which comes with the platform.

The List of Gen 5 Platform Calibration functions

The following functions are used to configure and retrieve the platform calibration settings.

fmSetPlatformDate
fmGetPlatformDate
fmSetPlatformModelNumber
fmGetPlatformModelNumber
fmSetPlatformSerialNumber
fmGetPlatformSerialNumber
fmSetPlatformLengthAndWidth
fmGetPlatformLengthAndWidth
fmSetPlatformXYZOffsets
fmGetPlatformXYZOffsets
fmSetPlatformCapacity
fmGetPlatformCapacity
fmSetPlatformBridgeResistance
fmGetPlatformBridgeResistance
fmSetInvertedSensitivityMatrix
fmGetInvertedSensitivityMatrix

16.0 Data Synchronization and the Gen 5

The DLL handles all data synchronization between signal conditioners. When using a single USB hub the is skew is approximately +- 1.5 microseconds between signal conditioners. If using multiple hubs the skew is less than +-125 microseconds between hubs.

17.0 The AMTI Smart Platform and the Gen 5

An AMTI smart platform contains all of its calibration information embedded within the platform. When a Gen 5 is turned on it checks to determine if it is connected to a smart platform. If it is connected it uploads the smart platforms calibration information and uses it.

NOTE: If a platform is hot swapped to a running Gen 5, the Gen 5 must be power-cycled to detect the smart platform. AMTI does not recommend hot swapping equipment.

18.0 Troubleshooting Tips

Question: The DLL is not delivering data after sending the start command.

Answer: The individual signal conditioners may be set for different acquisition rates, unit types and genlock states. This happens because some Gen 5's have been turned off for a while or a new one is introduced. When first stating up broadcast the desired acquisition rate, genlock state and unit types to prevent this.

Question: While running in Genlock mode the DLL is not delivering data after sending a start command.

Answer: The genlock signal should be running before the start command is sent. See the section, Operating in Genlock Mode.

19.0 Function Definitions

The following Sections contain the definitions for each of the DLL functions. They are grouped according to following categories:

DLL Initialization
Data Collection
Apply and Save
Gen 5 Configuration
Retrieving the Gen 5 Mechanical Limits
Platform Ordering
Gen 5 Calibration
Gen 5 Platform Calibration

20.0 The DLL Initialization Function Definitions

fmDLLInit

Description

This function must be called first. After calling it the program should either set a timer or sleep for 250 milliseconds. Then follow it by a call to ***fmDLLIsDeviceInitComplete*** to see if the DLL is loaded and the devices ready. If ***fmDLLIsDeviceInitComplete*** returns 0 the initialization is not complete. Reset the timer or go back to sleep and try again.

When called the AMTI USB DEVICE DLL conducts a search for connected signal conditioners. For any connected signal conditioners, it uploads the settings to the DLL for instant access.

The DLL loads the last saved configuration file, AMTIUsbSetup.cfg, and compares the previous configuration against the current setup to detect whether all the signal conditioners are present. By calling ***fmDLLSetupCheck*** you may determine whether the current setup matches the configuration file.

The DLL always uses the platform data collection order from the configuration file. It will maintain that dataset platform order even if some signal conditioners are not present.

Format

void fmDLLInit(void);

Related Functions

fmDLLIsDeviceInitComplete
fmDLLSetupCheck

fmDLLIsDeviceInitComplete

Description

This function works in conjunction with ***fmDLLInit***. After ***fmDLLInit*** has been called, call ***fmDLLIsDeviceInitComplete*** to see if the DLL has completed initialization. See ***fmDLLInit*** for more information.

Format

int fmDLLIsDeviceInitComplete(void);

Returns

Returns	Description
0	Have not completed initializing the DLL
1	The DLL is initialized, no signal conditioners are present
2	The DLL is initialized

Related Functions

fmDLLInit

fmDLLSetupCheck

Description

This function should be called after the DLL initialization has been completed and confirmed by ***fmDLLIsDeviceInitComplete*.** The function ***fmDLLSetupCheck*** compares the last saved DLL configuration file to the current DLL setup and notes any changes or discrepancies which may need attending.

Format

int fmDLLSetupCheck(void);

Returns

Integer

Return	Description
0	No Gen 5's were found
1	The Gen 5 setup is the same as the last saved configuration.
211	The configuration file was not found
213	A configuration file was found but for the wrong version of the software
214	The configuration has changed. A different number of Gen 5's were detected than the previously saved setup.
215	The configuration has changed. The serial numbers don't match the previously saved setup.

Related Functions

fmDLLInit

fmDLLSetUSBPacketSize

Description

Set the size of a packet being sent from the Gen 5 signal conditioner to the PC

The current size of a packet is 512 bytes. Each packet has 16 datasets with 8 elements in each dataset. Each element uses 4 bytes IEEE float. The 8 elements consist of a dataset counter, 6 data channels, and a trigger channel.

Format

void fmDLLSetUSBPacketSize(int cnt);

Arguments

integer cnt

Currently, cnt must always be set to 512

fmDLLGetDeviceCount

Description

This function returns the current number of connected signal conditioners

Format

int fmDLLGetDeviceCount(void)

Returns

Integer

Return	Description
0	No signal conditioners found
>0	Number of signal conditioners found

Related Functions

fmDLLSelectDeviceIndex

fmDLLSelectDeviceIndex

Description

Before communicating with a specific signal conditioner, the device must be selected first. The ***fmDLLSelectDeviceIndex*** function selects a specific signal conditioner by its device index. Once a signal conditioner has been selected you may communicate with it through any of the ***fm*** prefix type functions. The device indexes are ordered 0 to the number of signal conditioners minus one. They are always ordered in the platform data collection order stored in the DLL configuration file. Use ***fmDLLGetDeviceCount*** to know how many devices are connected.

To find out what signal conditioner is associated with a device index call ***fmGetAmplifierSerialNumber***.

The functions using ***the fmBroadcast***, or ***fmDLL*** prefix in their names do not require this function as they are general functions not specific to any signal conditioner. The ***fmBroadcast*** prefixed functions broadcast commands to all connected signal conditioners. The ***fmDLL*** prefixed functions are commands which concern DLL settings and are not specific to signal conditioners.

Format

void fmDLLSelectDeviceIndex(int device_index)

Arguments

Integer device_index

The index of the signal conditioner you want to communicate with.

Related Functions

fmDLLGetDeviceCount
fmDLLGetDeviceIndex

fmDLLGetDeviceIndex

Description

This function returns the device index of the currently selected signal conditioner. Use ***fmDLLSelectDeviceIndex*** to select a signal conditioner as the currently selected device.

Format

int fmDLLGetDeviceIndex(void)

Returns

integer

This function returns the device index of the currently selected signal conditioner.

Related Functions

fmDLLSelectDeviceIndex

fmDLLSaveConfiguration

Description

This function saves the current DLL settings to a configuration file stored in the Windows32 directory. The configuration file is named AMTIUSBSetup.cfg.

The configuration file contains the following:

Global Settings	Description , Range, or possible values
The configuration file version number	12345 (current version)
The signal conditioner count	0-15 (range of possible values)
The acquisition rate	0-2000 (range of possible values)
The run mode	0-5 (Range of possible values)
The genlock state	0-2 (Range of possible values)
Signal Conditioner Settings	Always saved in data collection order
Signal conditioner serial number	
Signal conditioner model number	
Platform serial number	
Platform model number	

Format

int fmDLLSaveConfiguration (void);

Related Functions

fmDLLInit

fmDLLShutDown

Description

In order to shut down the DLL, ***fmDLLShutDown*** must be called. Calling this function terminates all running threads and performs cleanup for the DLL. After calling this command adequate time must be allotted for cleanup before closing your application. This time increases per signal conditioner however 500 ms should be more than adequate.

If you are planning on reinitializing the DLL to search for additional signal conditioners and not really terminating the application just call ***fmDLLInit*** again. Do not call ***fmDLLShutDown***.

Format

int fmDLLShutDown(void)

Related Functions

fmDLLInit

21.0 The Data Collection Function Definitions

fmBroadcastRunMode

Description

This function sets the data output of the signal conditioner. For digital USB data, the choices are English units, metric units, or bits. For analog data the choices are fully conditioned and MSA 6 compatible.

For digital data if the units are metric the forces are Newton's and the moments are Newton-meters. If the units are English the forces are pounds and the moments are foot-pounds. If the units are bits the full scale range is +-16384 bits

In MSA 6 compatible analog output mode the signal conditioner performs as a traditional analog amplifier with software selectable gains of 500, 1000, 2000, or 4000. Calibration corrections are applied for channel excitations, channel gains, cable length and bridge resistances.

In fully conditioned analog output mode calibration corrections are applied for channel excitations, channel gains, cable length and bridge resistances, and a platform sensitivity matrix is used to correct crosstalk. A user supplied conversion factor is used to scale the analog outputs.

Format

void fmBroadcastRunMode(int mode);

Arguments

Integer mode

Mode	Digital	Analog Volts
0	Metric	MSA 6 Compatible
1	Metric	Fully Conditioned
2	English	MSA 6 Compatible
3	English	Fully Conditioned
4	Bits	MSA 6 Compatible

Related Functions

fmDLLGetRunMode

fmDLLGetRunMode

Description

This function returns the last broadcast data output mode of the DLL. For digital USB data, the choices are English units, metric units, or bits. For analog data the choices are MSA 6 compatible and fully conditioned.

For digital data, if the units are metric the forces are Newton's and the moments are Newton-meters. If the units are English the forces are pounds and the moments are foot-pounds. If the units are bits the full scale range is +-16384 bits

In MSA 6 compatible analog output mode the signal conditioner performs as a traditional analog amplifier with software selectable gains of 500, 1000, 2000, or 4000. Calibration corrections are applied for channel excitations, channel gains, cable length and bridge resistances.

In fully conditioned analog output mode calibration corrections are applied for channel excitations, channel gains, cable length and bridge resistances, and a platform sensitivity matrix is used to correct crosstalk. A user supplied conversion factor is used to scale the analog outputs.

Format

int fmDLLGetRunMode(void)

Returns

Integer

Return	Digital	Analog Volts
0	Metric	MSA 6 Compatible
1	Metric	Fully Conditioned
2	English	MSA 6 Compatible
3	English	Fully Conditioned
4	Bits	MSA 6 Compatible

Related Functions

fmBroadcastRunMode
fmGetRunMode

fmGetRunMode

Description

This function returns the run mode of the currently selected signal conditioner. For digital USB data, the choices are English units, metric units, or bits. For analog data the choices are MSA 6 compatible and fully conditioned.

For digital data if the units are metric the forces are Newton's and the moments are Newton-meters. If the units are English the forces are lbs and the moments are ft-lbs. If the units are bits the full scale range is +-16384.

In MSA 6 compatible analog output mode the signal conditioner performs as a traditional analog amplifier with software selectable gains of 500, 1000, 2000, or 4000. Calibration corrections are applied for channel excitations, channel gains, cable length and bridge resistances.

In fully conditioned analog output mode calibration corrections are applied for channel excitations, channel gains, cable length and bridge resistances, and a platform sensitivity matrix is used to correct crosstalk. A user supplied conversion factor is used to scale the analog outputs.

Format

int fmGetRunMode(void)

Returns

Integer

Return	Digital	Analog Volts
0	Metric	MSA 6 Compatible
1	Metric	Fully Conditioned
2	English	MSA 6 Compatible
3	English	Fully Conditioned
4	Bits	MSA 6 Compatible

Related Functions

fmBroadcastRunMode
fmDLLGetRunMode

fmBroadcastGenlock

Description

Call this function to set the genlock mode.

In genlock mode, the signal conditioner collects a dataset only on the rising or falling edge of an electrical signal input into the genlock port of the signal conditioner. For more information refer to the section, Using the Genlock Signal, and the Gen 5 user manual.

You must still call ***fmBroadcastStart*** to start data collection.

Format

void fmBroadcastGenlock (int value)

Arguments

Integer value

Return	Description
0	Genlock off
1	collect datasets on rising edge
2	collect datasets on falling edge

Related Functions

fmDLLGetGenlock

fmDLLGetGenlock

Description

This function asks the DLL whether what the last genlock configuration setting was.

In genlock mode, the signal conditioner collects a dataset only on the rising or falling edge of an electrical signal input into the genlock port of the signal conditioner. For more information refer to the section, Using the Genlock Signal, and the Gen 5 user manual.

Format

int fmDLLGetGenlock(void)

Returns

Integer

Return	Description
0	Genlock mode is off
1	Collect dataset on rising edge
2	Collect dataset on falling edge

Related Functions

fmBroadcastGenlock

fmBroadcastAcquisitionRate

Description

This function broadcasts an acquisition rate, in datasets per second, to all connected signal conditioners.

Format

void fmBroadcastAcquisitionRate(int idata)

Arguments

Integeridata

The following acquisition rates are permissible. If the acquisition rate is not recognized it will default to 500.

Acquisition Rates									
2000	1800	1500	1200	1000	900	800	600	500	450
400	360	300	250	240	225	200	180	150	125
120	100	90	80	75	60	50	45	40	30
25	20	15	10						

Related Functions

fmDLLGetAcquisitionRate
fmGetAcquisitionRate

fmDLLGetAcquisitionRate

Description

This function returns the last broadcast acquisition rate stored in the DLL, The acquisition rate is in datasets per second.

Format

int fmDLLGetAcquisitionRate(void)

Returns

Integer

Acquisition Rates									
2000	1800	1500	1200	1000	900	800	600	500	450
400	360	300	250	240	225	200	180	150	125
120	100	90	80	75	60	50	45	40	30
25	20	15	10						

Related Functions

fmBroadcastAcquisitionRate
fmGetAcquisitionRate

fmGetAcquisitionRate

Description

This function returns the acquisition rate of the currently selected signal conditioner, in datasets per second.

Format

int fmGetAcquisitionRate(void)

Returns

Integer

Acquisition Rates									
2000	1800	1500	1200	1000	900	800	600	500	450
400	360	300	250	240	225	200	180	150	125
120	100	90	80	75	60	50	45	40	30
25	20	15	10						

Related Functions

fmBroadcastAcquisitionRate
fmDLLGetAcquisitionRate

fmBroadcastStart

Description

Call this function to start data acquisition from all connected Gen 5's.

Any other SDK function called after ***fmBroadcastStart*** except for ***fmBroadcastZero*** will automatically stop acquisition. This is to preserve signal conditioner synchronization. The formal stop acquisition function is ***fmBroadcastStop.***

This function does not affect the analog outputs, as they are always on.

Format

void fmBroadcastStart (void)

Related Functions

fmBroadcastStop

fmBroadcastStop

Description

Call this function to stop data acquisition from all connected Gen 5's.

This function does not affect analog outputs, as they are always on.

Format

void fmBroadcastStop (void)

Related Functions

fmBroadcastStart

fmBroadcastZero

Description

This function tells all connected Gen 5's to zero their platforms. This function may be called before or after the data collection start command. Data collected while the zero process is taking place will consist of all zero's. If this function is called after the start command it will not cause data collection to stop unlike some other DLL functions.

Format

void fmBroadcastZero(void)

Related Functions

fmBroadcastStart
fmBroadcastStop

fmDLLPostDataReadyMessages

Description

There are three ways to receive data. The first is by repeatedly polling for ready data. The second is to have the DLL send a message to the main application window each time a data buffer is ready. The third is to have the DLL send a message to a user thread each time a data buffer is ready. If you would like to send a message to a window or user thread indicating data is ready set ***fmDLLPostDataReadyMessages*** to 1.

If messages are to be used you must also call either ***fmDLLPostUserThreadMessages*** or ***fmDLLPostWindowMessages***. You cannot post to both the main application window and user threads at the same time.

Format

fmDLLPostDataReadyMessages(int bMessages)

Arguments

int bMessages

integer	Description
0	Do not post data ready messages
1	Post data ready messages

Related Functions

fmDLLPostUserThreadMessages
fmDLLPostWindowMessages

fmDLLPostWindowMessages

Description

If you have chosen to receive data by having the DLL post a message to the application main window each time data is ready you must first call ***fmDLLPostDataReadyMessages*** and then ***fmDLLPostWindowMessages***.
The function ***fmDLLPostWindowMessages*** passes the windows handle to the DLL.

If you are using the Microsoft Foundation Classes the GetSafeHwnd function will return a handle to the window.

The window message identifier must always be (WM_USER + 108)

Format

void fmDLLPostWindowMessages(HWND handle)

Arguments

HWND handle
A handle to the window

Related Functions

fmDLLPostDataReadyMessages
fmDLLPostUserThreadMessages

fmDLLPostUserThreadMessages

Description

If you have chosen to receive data by having the DLL post a message to a user thread you must first call ***fmDLLPostDataReadyMessages***. You must then call ***fmDLLPostUserThreadMessages*** to send the user thread ID to the DLL.

The thread message identifier must always be WM_USER + 109

Format

void fmDLLPostUserThreadMessages(unsigned int threadID)

Arguments

unsigned int threadID

ID of the thread attached to the CWinThread, m_nThreadID is a member of the CWinThread class.

Related Functions

fmDLLPostDataReadyMessages
fmDLLPostUserThreadMessages

fmDLLSetDataFormat

Description

There are two data formats. The user can receive each dataset in eight element or six element format. A six element dataset will consist solely of the force and moment channels. An 8 element dataset will have two additional channels, a dataset counter and a trigger state. The dataset counter records the number of datasets after the start command was received. The dataset counter rolls over at 16,777,215 ($2^{24} - 1$). The trigger state will be either 0 or 1 depending on the input state of the trigger port on the signal conditioner.

Eight Element format

Channel	0	1	2	3	4	5	6	7
Element	Counter	Fx	Fy	Fz	Mx	My	Mz	Trigger

Six Element Format

Channel	0	1	2	3	4	5
Element	Fx	Fy	Fz	Mx	My	Mz

Format

void fmDLLSetDataFormat(int DataFormat)

Arguments

int DataFormat

integer	Description
0	Six channel format
1	Eight channel format

Related Functions

FmDLLTransferFloatData
FmDLLGetTheFloatDataLBVStyle

fmDLLTransferFloatData

Description

This function is used to receive incoming data. If your development environment is Visual C++ or some other C language this is the recommended data collection function. If you are developing in Labview or Matlab you may have to use ***fmDLLGetTheFloatDataLBVStyle*** function.

If data is available the ptr argument will return pointing to a full data buffer. The function does not return partial data buffers.

The data buffer consists of 16 datasets from each connected signal conditioner. For one signal conditioner the data buffer consists of 16 datasets, For two signal conditioners are data buffer consist of 16 datasets from signal conditioner one, and 16 datasets from signal conditioner two etc.

A single Gen 5 dataset will consist of either 6 or 8 elements of data depending on the selected data format. Each data element is of the type float. The data format is set by calling ***fmDLLSetDataFormat***. A six element dataset will consist solely of the force and moment channels. An 8 element dataset will have two additional channels, a dataset counter and a trigger state. The dataset counter records the number of dataset after the start command was received. The trigger state will be either 0 or 1 depending on the input state of the trigger port on the signal conditioner.

Dataset Format								
Channel index	0	1	2	3	4	5	6	7
6 element	Fx	Fy	Fz	Mx	My	Mz		
8 element	Data counter	Fx	Fy	Fz	Mx	My	Mz	Trigger state

The order of the datasets in the data buffer must be considered. If a data buffer contains data from three signal conditioners the first dataset in the data buffer would be from Gen 5 one, the second dataset from Gen 5 two etc.

The formulating for calculating the size of the data buffer is the following.

Whereas:

DBS = the data buffer size
NCD = the number of channels per dataset

NOSC = the number of signal conditioners
16 = the number datasets from each signal conditioner in every packet

DBS = NCD * NOSC * 16

Format

int fmDLLTransferFloatData(float *&ptr)

Arguments

float *&data

The function requires a pointer to a float data type. If data is available the pointer will return pointing to a full data buffer of type float. If no data is available the pointer will be unchanged.

Returns

Returns	Description
0	No new data available
1	Data returned

Related Functions

fmDLLGetDeviceCount
fmDLLSetDataFormat
fmDLLGetTheFloatDataLBVStyle

fmDLLGetTheFloatDataLBVStyle

Description

This function is used to receive incoming data. If your development environment is LabView or Matlab this is the recommended data collection function. If you are developing in Visual C++ or some other C language you should to use ***fmDLLTransferFloatData***.

The difference between the two data transfer funtions is the ***fmDLLGetTheFloatDataLBVStyle*** passes in an array to be filled. The ***fmDLLTransferFloatData*** function simply passes in a pointer to a float which returns pointing to an array of floats.

If data is available the data argument will return with a full data buffer. The function does not return partial data buffers.

The data buffer consists of 16 datasets from each connected signal conditioner. For one signal conditioner the data buffer consists of 16 datasets, For two signal conditioners are data buffer consist of 16 datasets from signal conditioner one, and 16 datasets from signal conditioner two etc.

A single Gen 5 dataset will consist of either 6 or 8 elements of data depending on the selected data format. Each data element is of the type float. The data format is set by calling ***fmDLLSetDataFormat***. A six element dataset will consist solely of the force and moment channels. An 8 element dataset will have two additional channels, a dataset counter and a trigger state. The dataset counter records the number of dataset after the start command was received. The trigger state will be either 0 or 1 depending on the input state of the trigger port on the signal conditioner.

Dataset Format								
Channel index	0	1	2	3	4	5	6	7
6 element	Fx	Fy	Fz	Mx	My	Mz		
8 element	Data counter	Fx	Fy	Fz	Mx	My	Mz	Trigger state

The order of the datasets in the data buffer must be considered. If a data buffer contains data from three signal conditioners the first dataset in the data buffer would be from Gen 5 one, the second dataset from Gen 5 two etc.

The formulating for calculating the size of the data buffer is the following.

Whereas:

DBS = the data buffer size
NCD = the number of channels per dataset
NOSC = the number of signal conditioners
16 = the number datasets from each signal conditioner in every packet

DBS = NCD * NOSC * 16

Format

int fmDLLGetTheFloatDataLBVStyle(float *data, int size)

Arguments

float *data

The function requires a pointer to an array of type float. The array size should be calculated according to the formula above.

If data is available the array will returned filled. If no data is available the array will return unchanged.

Int size

The size of the array calculated by the formula above

Returns

Returns	Description
0	No new data available
1	Data returned

Related Functions

fmDLLGetDeviceCount
fmDLLSetDataFormat
fmDLLTransferFloatData

22.0 The Apply and Save Function Definitions

fmBroadcastResetSoftware

Description

This function resets the software of all connected Gen 5 signal conditioners.

When new signal conditioner settings are downloaded, the changes are not implemented until this function is called. First make all the configuration changes (excitations, gains, acquisition rate, etcetera). Then call this function for the changes to be applied. When you call this function do not follow it directly with another function call as the Gen 5 will go into an indeterminate state when resetting; pause for at least 250 milliseconds. This function does not save the changes to flash memory. Power cycling the signal conditioner will reset the last saved settings.

The function ***fmBroadcastAcquisitionRate*** does not need an ***fmResetsoftware*** function call to be applied. It is applied on the next ***fmBroadcastStart*** command.

Format

void fmBroadcastResetSoftware(void)

Related Functions

fmBroadcastSave
fmSave
fmResetSoftware

fmResetSoftware

Description

This function resets the software of the currently selected signal conditioner

When new signal conditioner settings are downloaded, the changes are not implemented until this function is called. First make all the configuration changes (excitations, gains, acquisition rate, etcetera). Then call this function for the changes to be applied. When you call this function do not follow it directly with another function call as the Gen 5 will go into an indeterminate state when resetting; pause for at least 250 milliseconds. This function does not save the changes to flash memory. Power cycling the signal conditioner will reset the last saved settings.

The function ***fmBroadcastAcquisitionRate*** does not need a f***mResetsoftware*** function call to be applied. It is applied on the next ***fmBroadcastStart*** command.

Format

void fmResetSoftware(void)

Related Functions

fmBroadcastResetSoftware
fmResetSoftware
fmBroadcastSave
fmSave

fmBroadcastSave

Description

This function saves the current Gen 5 settings to the Gen 5 non volatile memory. The saved settings are restored whenever the Gen 5 is powered on.

It takes a fair amount of time to write to flash. Do not send any signal conditioner commands for at least 250 milliseconds after calling this function as the signal conditioner is busy.

Format

void fmBroadcastSave(void)

Related Functions

fmBroadcastResetSoftware
fmResetSoftware
fmSave

fmSave

Description

This function saves the current Gen 5 settings to the Gen 5. The saved settings are restored whenever the Gen 5 is powered on.

It takes a fair amount of time to write to flash. Do not send any signal conditioner commands for at least 250 milliseconds after calling this function as the signal conditioner is busy. Since this flash chip is rated for 20000 to 50000 writes you want the user to make all his configuration changes and then save.

Format

void fmSave(void)

Related Functions

fmBroadcastSave
fmBroadcastResetSoftware

fmApplyLimited

Description

This function saves the current hardware zero settings to the Gen 5 flash memory. When the Gen 5 is powered on these zero settings will automatically be loaded.

Format

void fmApplyLimited(void)

Related Functions

fmBroadcastSave
fmBroadcastResetSoftware

23.0 The Gen 5 Configuration Function Definitions

fmSetCurrentGains

Description

This function tells the signal conditioner what gain setting to use for each channel. The function requires a 6 element array of type long integer. The possible values for each element are (0, 1, 2, 3) corresponding to the gains shown in the table below.

Array Setting	Corresponding Gains
0	500
1	1000
2	2000
3	4000

Format

void fmSetCurrentGains(long *ldata);

Arguments

long *ldata

A pointer to an array of 6 long integers

Related Functions

fmGetCurrentGains

fmGetCurrentGains

Description

This function retrieves the current gain setting of each channel of the currently selected signal conditioner. The function requires a 6 element array of type long integer. The possible values for each element are (0, 1, 2, 3) which correspond to the gains shown in the table below.

Array Setting	Corresponding Gains
0	500
1	1000
2	2000
3	4000

Format

void fmGetCurrentGains(long *ldata)

Arguments

long *ldata

A pointer to an array of 6 long integers

Related Functions

fmSetCurrentGains

fmSetCurrentExcitations

Description

This function tells the signal conditioner what excitation setting to use for each channel. The function requires a 6 element array of type long integer. The possible values for each element are (0, 1, 2 corresponding to the excitations shown in the table below.

Array Setting	Corresponding Excitation
0	2.5
1	5.0
2	10.0

Format

void fmSetCurrentExcitations(long *ldata)

Arguments

long *ldata

A pointer to an array of 6 long integers

Related Functions

fmGetCurrentExcitations

fmGetCurrentExcitations

Description

This function retrieves the current excitation setting of each channel of the currently selected signal conditioner. The function requires a 6 element array of type long integer. The possible values for each element are (0, 1, 2) which correspond to the excitations shown in the table below.

Array Setting	Corresponding Excitation
0	2.5
1	5.0
2	10.0

Format

void fmGetCurrentExcitations(long *ldata)

Arguments

long *ldata

A pointer to an array of 6 long integers

Related Functions

fmSetCurrentExcitation

fmSetChannelOffsetsTable

Description

The channel offset parameter allows the user to offset the mechanical range of the signal conditioner to better adapt to the test being conducted. The channel offsets table is a 6 element array of type float. The value for each channel must lie between -0.99 and 0.99. Zero is the default value.

Say for instance that a test involves jumping on a platform. The expected physical range of channel Fz platform loading may be between -25 and 1500 Newton's. Traditionally the electrical range of the signal conditioner would need to be -2000 to +2000 Newton's in order to encompass the physical load range. However a better signal conditioner resolution could be accomplished by doubling the gain and offsetting the load range from -250 to 1750 Newton's.

The channel offset table allows the user to set a zero offset. The tables below illustrates the effects of three different zero offset settings for a single channel on a signal conditioner with an electrical range configured for +-1000 Newton's. The first table is referring to the digital outputs and the second table is referring to the analog outputs.

Digital Output in Newton's			
channel offset	0	0.75	-0.75
maximum electrical range	1000	250	1750
zero load output	0	0	0
minimum electrical range	-1000	-1750	-250

Analog Output in Volts			
channel offset	0	0.75	-0.75
maximum output range	5.0	5.0	5.0
zero load output	0.0	3.75	-3.75
minimum output range	--5.0	-5.0	-5.0

The ***fmSetChannelOffsetsTable*** function downloads the channel offsets table to the currently selected Gen 5. The array should be loaded in channel order as in the following table.

	Channel Offset Table					
Channel	**Fx**	**Fy**	**Fz**	**Mx**	**My**	**Mz**
Index	**0**	**1**	**2**	**3**	**4**	**5**
Range	**Nominal Values**					
(-0.99 to 0.99)	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**

Format

void fmSetChannelOffsetsTable(float *data)

Arguments

float *data

A pointer to an array of 6 floats

Related Functions

fmGetChannelOffsetsTable
fmUpdateMechanicalMaxAndMin
fmGetMechanicalMaxAndMin

fmGetChannelOffsetsTable

Description

This function retrieves the channel offsets table from the currently selected signal conditioner. The array will be in channel order as in the following table. The values should all lie between +-0.9.

	Channel Offset Table					
Channel	Fx	Fy	Fz	Mx	My	Mz
Index	0	1	2	3	4	5
Range	Nominal Values					
(-0.99 to 0.99)	0.0	0.0	0.0	0.0	0.0	0.0

Format

void fmGetChannelOffsetsTable(float *data)

Arguments

float *data

A pointer to an array of 6 floats

Related Functions

fmSetChannelOffsetsTable

fmSetCableLength

Description

This function sets the cable length between the platform and the currently selected signal conditioner. It must be in set in feet.

The strength of the electrical signal will drop in proportion to the cable length. By entering the cable length the signal conditioner can apply a correction factor.

Format

void fmSetCableLength(float fdata)

Arguments

float fdata

The cable length in feet between the platform and the signal conditioner

Related Functions

fmGetCableLength

fmGetCableLength

Description

This function retrieves the cable length parameter from the Gen 5. It will always be in feet.

The strength of the electrical signal drops in proportion to the cable length. By entering the cable length the signal conditioner can apply a correction factor.

Format

float fmGetCableLength(void)

Return

float

The cable length in feet between the platform and the signal conditioner

Related Functions

fmSetCableLength

fmSetMatrixMode

Description

The inverted sensitivity table is a 36 element array of type float; This 6 x 6 calibration matrix is used to eliminate crosstalk. it consists of calibration coefficients which convert microvolts to engineering units. Occasionally the user may only want to use the main diagonal terms as opposed to the full calibration matrix.. The ***fmSetMarixMode*** function tells the signal conditioner to use either the full array or only the main diagonal terms.

	Sample inverted Sensitivity Matrix					
Channel	0	1	2	3	4	5
	VFx	VFy	VFz	VMx	VMy	VMz
	Input to channel i(lb,in-lb) is B(I,j)times the electrical output j(uV,Vex)					
	BP 400600-2000					
Fx	0.6519	-0.0068	-0.0019	0.0009	-0.0017	-0.0003
Fy	0.0090	0.6515	-0.0037	0.0009	0.0005	0.0010
Fz	0.0018	0.0017	2.5523	-0.0062	0.0001	0.0026
Mx	-0.0044	-0.0032	0.0003	12.8281	0.0108	-0.0138
My	0.0725	-0.0032	0.0003	0.0058	10.1358	-0.0140
Mz	0.0649	0.0821	0.0792	0.0123	0.0340	5.4451

Format

fmSetMatrixMode(long ldata)

Arguments

long ldata

Long ldata	Description
1	Use full matrix
0	Use main diagonal terms only

Related Functions

fmSetInvertedSensitivityMatrix
fmGetInvertedSensitivityMatrix
fmGetMatrixMode

fmGetMatrixMode

Description

The ***fmGetMatrixMode*** function returns the current matrix mode of the signal conditioner.

Format

long fmGetMatrixMode(void)

Returns

long

Return	Description
1	The Gen 5 is currently set to use the full calibration matrix.
0	The Gen 5 is currently set to use the calibration matrix main diagonal terms only.

Related Functions

fmSetInvertedSensitivityMatrix
fmGetInvertedSensitivityMatrix
fmSetMatrixMode

fmSetPlatformRotation

Description

This function allows the signal conditioner to perform a rotational transformation on the data. Sometimes a platform must be rotated from its original orientation to get the cable connectors out of the way. This function allows the user to change the platform orientation while maintaining the X, Y axis orientation. The rotation must be entered in degrees (0 to 360). The default setting is zero.

The transformation cannot apply to all run modes.

Output modes		Transformation applied
Digital	English	Yes
	Metric	Yes
	Bits	No
Analog	Fully Conditioned	Yes
	MSA 6 Compatible	No

Format

void fmSetPlatformRotation(float data)

Arguments

float data

A number from 0 to 360

Related Functions

fmGetPlatformRotation
fmBroadcastRunMode

fmGetPlatformRotation

Description

This function retrieves the current rotational transformation setting for the signal conditioner. The rotation will be from 0 to 360 degrees. The default rotation is zero.

Format

float fmGetPlatformRotation(void)

Returns

float

A number from 0 to 360

Related Functions

fmSetPlatformRotation
fmBroadcastRunMode

24.0 Retrieving the Gen 5 Mechanical Limit Function Definitions

fmUpdateMechanicalMaxAndMin

Description

This function uploads the last calculated mechanical range of the Gen 5 signal conditioner under its current configuration to the DLL. The mechanical range is recalculated every time the signal conditioner is reset. The functions ***fmBroadcastResetSoftware*** and ***fmResetSoftware*** reset the signal conditioner.

NOTE: This function is uploads the currently configured mechanical limits of the signal conditioner, not that of the attached platform.

Format

void fmUpdateMechanicalMaxAndMin(void)

Related Functions

fmGetMechanicalMaxAndMin

fmGetMechanicalMaxAndMin

Description

This function retrieves the mechanical maximum and minimum for each channel under the current signal conditioner configuration. The mechanical max and min table is a 12 element array of type float. The array will be loaded in row, column order, the first row being mechanical maximums and the second row being mechanical minimums. The values will be in either English or metric units depending on the current run mode selection.

The function ***fmUpdateMechanicalMaxAndMin*** must be called prior to ***fmGetMechanicalMaxAndMin*** unless no parameters have been modified after initializing the DLL.

NOTE: This function retrieves the currently configured mechanical limits of the signal conditioner, not that of the attached platform.

Format

int fmGetMechanicalMaxAndMin(float *data)

Parameter

A pointer to a 12 element array of type float.

Return

Integer

Return	Description
0	Please wait, The DLL is currently uploading the mechanical range after a call to ***fmUpdateMechanicalMaxAndMin*** was made.
1	This is the last updated mechanical range

Related Functions

fmUpdateMechanicalMaxAndMin
fmDLLGetRunMode

fmUpdateAnalogMaxAndMin

Description

This function uploads the last calculated analog output range, in engineering units, of the Gen 5 signal conditioner to the DLL. The mechanical range is recalculated every time the signal conditioner is reset. The functions ***fmBroadcastResetSoftware*** and ***fmResetSoftware*** reset the signal conditioner. Upon DLL initialization ***fmUpdateAnalogMaxAndMin*** is automatically called.

The maximum output is calculated by dividing the channel DAC sensitivity value by 5.0. The minimum output is calculated by dividing the channel DAC sensitivity by -5.0. The DAC sensitivity values are always in millivolts per pound for forces and millivolts per inch pound for moments.

If the analog output range is greater than the configured signal conditioner mechanical range the analog output range will be constrained by the mechanical range.

This function is for informational purposes only.

NOTE: When the analog outputs are set to MSA 6 compatible mode this function is indeterminate. The analog output range is then nominally the same as the electrical range.

Format

void fmUpdateAnalogMaxAndMin(void)

Related Functions

fmGetAnalogMaxAndMin
fmUpdateMechanicalMaxAndMin
fmGetMechanicalMaxAndMin

fmGetAnalogMaxAndMin

Description

This function retrieves the analog output range, in engineering units, of the Gen 5, from the DLL. The analog output maximum and minimum table is a 12 element array of type float. The first 6 elements are the analog maximums; the last 6 elements are the analog minimums. The values will be in either English or metric units depending on the current run mode selection.

The function ***fmUpdateAnalogMaxAndMin*** must be called prior to ***fmGetAnalogMaxAndMin*** unless the DAC Sensitivities have not been modified after initializing the DLL.

NOTE: When the analog outputs are set to MSA 6 compatible mode this function is indeterminate. The analog output range is then nominally the same as the electrical range.

Format

int fmGetAnalogMaxAndMin(float *data);

Parameter

A pointer to a 12 element array of type float

Returns

Integer

Return	Description
0	Please wait, The DLL is currently uploading the analog range after a call to ***fmUpdateAnalogMaxAndMin*** was made.
1	This is the last updated analog output range

Related Functions

fmUpdateAnalogMaxAndMin
fmDLLGetRunMode
FmBroadcastResetSoftware
fmSetDACSensitivityTable

25.0 The Platform Ordering Function Definitions

fmDLLSetPlatformOrder

Description

This function sets a new platform data collection order. The platform data collection order is important because when analyzing data files it is necessary for the data to always appear in the correct columns. When using multiple signal conditioners the platform data order must be defined.

To use this function you must already know the current order of the platforms. To find the order of the platforms do the following. First call ***fmDLLGetDeviceCount*** to get the number of signal conditioners. Then create a loop to cycle through the signal conditioners. Use the functions ***fmDLLSelectDeviceIndex*** and ***fmGetAmplifierSerialNumber*** to get the serial number of each signal conditioner. Once you know the serial number for each device index simply map a new device index order into an array of integers and pass a pointer to the array into the ***fmDLLSetPlatformOrder*** function.

Format

void fmDLLSetPlatformOrder(int *DeviceImdexMap)

Arguments

Integer *DeviceImdexMap

The function accepts a pointer to an array of integers. Each array element will contain the index order of a signal conditioner we would like the current index order to change to. The integer array must be of size at least equal to the current number of signal conditioners.

The user will provide each element of the array with a current Gen 5 device index. After ***fmDLLSetPlatformOrder*** is called the index of each array element will become the new device index.

Related Functions

fmDLLGetPlatformOrder

fmBroadcastPlatformOrderingThreshold

Description

This function sets the platform threshold used for auto-ordering platforms. When auto ordering is used the DLL is set to detect when each platform is stepped on. The order in which the platforms are stepped on determines the platform order in the collected data. The platform threshold value is a value which is crossed when a user steps on the platform. Be sure it is not set so low as to be triggered by vibration or noise. It is to be entered in bits. The full scale range of the signal conditioner in bits is +-16384.

Format

void fmBroadcastPlatformOrderingThreshold(float value)

Arguments

float value

Valid values are 0 to 16384 bits. 0 usually indicates zero platform load.

Related Functions

fmDLLStartPlatformOrdering
fmDLLCancelPlatformOrdering

fmDLLStartPlatformOrdering

Description

The data collection order is the order in which each platform appears in a collected dataset. One way to set the data collection order is to have a person walk on the platforms in the desired order.

When ***fmDLLStartPlatformOrdering*** is called the AMTIUSBDevice DLL listens to all platforms to detect if an FZ force threshold is crossed. The order in which each platforms threshold is crossed determines the data collection order.

Once all platforms have been detected the new platform order is set.

Format

void fmDLLStartPlatformOrdering(void)

Related Functions

fmDLLIsPlatformOrderingComplete
fmDLLCancelPlatformOrdering
fmBroadcastPlatformOrderingThreshold

fmDLLIsPlatformOrderingComplete

Description

Call **fmDLLIsPlatformOrderingComplete** to determine if the auto platform ordering process has been completed.

The data collection order is the order in which each platform appears in a collected dataset. One way to set the data collection order is to have a person walk on the platforms in the desired order.

After ***fmDLLStartPlatformOrdering*** is called the AMTIUSBDevice DLL listens to all platforms to detect if an FZ force threshold is crossed. The order in which each platforms threshold is crossed determines the data collection order.

Once all platforms have been detected the new platform order is set.

Format

int fmDLLIsPlatformOrderingComplete(void)

Returns

integer

Return	Description
0	Platform Ordering is not complete
1	Platform Ordering is complete

Related Functions

fmBroadcastPlatformOrderingThreshold
fmDLLStartPlatformOrdering
fmDLLCancelPlatformOrdering

fmDLLCancelPlatformOrdering

Description

Call ***fmDLLCancelPlatformOrdering*** to cancel the auto platform ordering process before it has completed.

The data collection order is the order in which each platform appears in a collected dataset. One way to set the data collection order is to have a person walk on the platforms in the desired order.

After ***fmDLLStartPlatformOrdering*** is called the AMTI USB Device DLL listens to all platforms to detect if an FZ force threshold is crossed. The order in which each platforms threshold is crossed determines the data collection order.

Once all platforms have been detected the new platform order is set.

Format

void fmDLLCancelPlatformOrdering(void)

Related Function

fmBroadcastPlatformOrderingThreshold
fmDLLStartPlatformOrdering
fmDLLCancelPlatformOrdering

26.0 The Gen 5 Calibration Function Definitions

fmGetAmplifierFirmwareVersion

Description

This function retrieves the firmware version of the currently selected signal conditioner.

Format

void fmGetAmplifierFirmwareVersion(char *Cdata)

Arguments

char *Cdata

A pointer to a 16 element array of type char

fmGetAmplifierModelNumber

Description

This function retrieves the current model number of the currently selected signal conditioner.

Format

void fmGetAmplifierModelNumber(char *Cdata)

Arguments

char *Cdata

A pointer to a element array of type char

Related Functions

fmGetAmplifierSerialNumber

fmGetAmplifierSerialNumber

Description

This function retrieves the current serial number of the currently selected signal conditioner.

Format

void fmGetAmplifierSerialNumber(char *Cdata)

Arguments

char *Cdata

A pointer to a 16 element array of type char

Related Functions

fmGetAmplifierModelNumber

fmGetAmplifierDate

Description

This function retrieves the last calibration date of the currently selected signal conditioner.

Format

void fmGetAmplifierDate(char *Cdata)

Arguments

char *Cdata

A pointer to a 12 element array of type char

Related Functions

fmGetAmplifierModelNumber

fmGetGainTable

Description

This function retrieves the gain correction table of the currently selected signal conditioner. The gain table is a 24 element array of type float. The array will be retrieved in row, column order as in the table below.

The nominal values presented below represent approximations for the correct values.

Format

void fmGetGainTable(float *data)

Arguments

float *data

A pointer to a 24 element array of type float

	Gain Table					
Channel	**0**	**1**	**2**	**3**	**4**	**5**
Gain	**Nominal Values**					
500	1.0	1.0	1.0	1.0	1.0	1.0
1000	2.0	2.0	2.0	2.0	2.0	2.0
2000	4.0	4.0	4.0	4.0	4.0	4.0
4000	8.0	8.0	8.0	8.0	8.0	8.0

Related Functions

fmSetGainTable

fmGetExcitationTable

Description

This function retrieves the excitation correction table for the currently selected signal conditioner. The excitation table is an 18 element array of type float. The array will be retrieved in row, column order as in the table below.

The nominal values presented below represent approximations for the correct values.

Format

void fmGetExcitationTable(float *data)

Arguments

float *data

A pointer to an 18 element array of type float

	Excitation Table					
Channel	**0**	**1**	**2**	**3**	**4**	**5**
Excitation	**Nominal Values**					
2.5	2.5	2.5	2.5	2.5	2.5	2.5
5.0	5.0	5.0	5.0	5.0	5.0	5.0
10.0	10.0	10.0	10.0	10.0	10.0	10.0

Related Functions

fmSetExcitationTable

fmGetDACGainsTable

Description

This function retrieves the digital to analog converter (DAC) gain correction table for the currently selected signal conditioner. The gain correction table is a 6 element array of type float. The array will be retrieved in channel order as in the table below.

The nominal values presented below represent approximations for the correct values.

Format

void fmGetDACGainsTable(float *data);

Arguments

float *data

A pointer to a 6 element array of type float

	DAC Gains Table					
Channel	**0**	**1**	**2**	**3**	**4**	**5**
	Nominal Values					
	-2.49423	-2.49423	-2.49423	-2.49423	-2.49423	-2.49423

Related Functions

fmSetDACGainsTable
fmSetCurrentGains
fmGetCurrentGains

fmGetDACOffsetTable

Description

This function retrieves the digital to analog converter (DAC) offset correction table for the currently selected signal conditioner. The offset correction table is a 6 element array of type float. The array will be retrieved channel order as in the table below.

The nominal values presented below represent approximations for the correct values.

Format

void fmGetDACOffsetTable(float *data)

Arguments

float *data

A pointer to a 6 element array of type floats

	DAC Offset Table					
Channel	0	1	2	3	4	5
	Nominal Values					
	0.0	0.0	0.0	0.0	0.0	0.0

Related Functions

fmSetDACGainsTable

fmGetDACSensitivities

Description

This function retrieves the digital to analog conversion (DAC) sensitivity table.

The digital to analog conversion (DAC) table contains conversion factors, one for each channel. This conversion factor is used to convert Gen 5 internally calculated digital force and moment values into analog output volts.

The force channel conversions are always millivolts per pound. The moment channel conversions are always millivolts per inch pound.

The DAC sensitivity Table is only applied when the analog outputs are set to fully conditioned mode. These same conversion values or the metric equivalents must be entered in the user application to convert the analog signal to engineering units within the PC.

Format

void fmGetDACSensitivities(float *data)

Arguments

float *data

A pointer to an 6 element array of type float

Related Functions

fmSetDACSensitivityTable
fmBroadcastRunMode

fmGetADRef

Description

This function retrieves the nominal analog to digital reference voltage value for the currently selected signal conditioner.

Format

float fmGetADRef(void)

Returns

Float

The nominal reference voltage value for the currently selected signal conditioner

27.0 The Gen 5 Platform Calibration Function Definitions

fmSetPlatformDate

Description

This function sets the platform calibration date of the currently selected signal conditioner.

Format

void fmSetPlatformDate (char *Cdata)

Arguments

char *Cdata

A pointer to a 12 element array of type char

Related Functions

fmGetPlatformDate

fmGetPlatformDate

Description

This function retrieves the platform calibration date of the currently selected signal conditioner.

Format

void fmGetPlatfromDate(char *Cdata)

Arguments

char *Cdata

A pointer to a 12 element array of type char

Related Functions

fmSetPlatformDate

fmSetPlatformModelNumber

Description

This function sets the model number of the currently selected signal conditioner.

Format

void fmSetPlatformModelNumber(char *Cdata)

Arguments

char *Cdata

A pointer to a 28 element array of type char

Returns

Related Functions

fmGetPlatformModelNumber

fmGetPlatformModelNumber

Description

This function retrieves the platform model number of the currently selected signal conditioner.

Format

void fmGetPlatformModelNumber(char *Cdata)

Arguments

char *Cdata

A pointer to a 28 element array of type char

Related Functions

fmSetPlatformModelNumber

fmSetPlatfromSerialNumber

Description

This function sets the platform serial number of the currently selected signal conditioner.

Format

void fmSetPlatformSerialNumber(char *Cdata)

Arguments

char *Cdata

A pointer to a 16 element array of type char

Related Functions

fmGetPlatfromSerialNumber

fmGetPlatfromSerialNumber

Description

This function retrieves the current platform serial number of the currently selected signal conditioner.

Format

void fmGetPlatformSerialNumber(char *Cdata)

Arguments

char *Cdata

A pointer to a 16 element array of type char

Related Functions

fmSetPlatfromSerialNumber

fmSetPlatformLengthAndWidth

Description

This function sets the platform length and width of the currently selected signal conditioner. The signal conditioner has no internal use for this information.

Format

void fmSetPlatformLengthAndWidth(char *Ldata, char *Wdata);

Arguments

char *Ldata

A 16 element array of type char

char *Wdata

A 16 element array of type char

Related Functions

fmGetPlatformLengthAndWidth

fmGetPlatformLengthAndWidth

Description

This function retrieves the platform length and width of the currently selected signal conditioner.

Format

void fmGetPlatformLengthAndWidth(char *Ldata, char *Wdata)

Arguments

char *Ldata

A pointer to a 16 element array of type char

char *Wdata

A pointer to a 16 element array of type char

Related Functions

fmSetPlatformLengthAndWidth

fmSetPlatformXYZOffsets

Description

This function sets the x, y, and z platform offsets of the currently selected signal conditioner. The signal conditioner has no internal use for these values.

Each platform has an electrical center which has a physical location somewhere within the platform. These offsets represent that location. For more information refer to the platform calibration information and manual.

Format

void fmSetPlatformXYZOffsets(float *data)

Arguments

float *data

A pointer to a 3 element array of type float

Array Element	0	1	2
Value	X offset	Y offset	Z offset

Related Functions

fmGetPlatformXYZOffsets

fmGetPlatformXYZOffsets

Description

This function retrieves the x, y, and z platform offsets of the currently selected signal conditioner.

Format

void fmGetPlatformXYZOffsets(float *data)

Arguments

float *data

A pointer to a 3 element array of type float

Array Element	0	1	2
Value	X offset	Y offset	Z offset

Related Functions

fmSetPlatformXYZOffsets

fmSetPlatformCapacity

Description

This function sets the platform capacities for the platform attached to the signal conditioner. The function requires a 6 element array of type float. These parameters are stored in English units.

These parameters are always shipped with the platform. These values have no internal usey are not actually used for anything other than letting the user know the platform capacity.

NOTE: If the Gen 5 is attached to a smart platform the Gen 5 will overwrite these parameters with those of the smart platform.

	Platform Capacity Table					
Channel	**Fx**	**Fy**	**Fz**	**Mx**	**My**	**Mz**
Units	**Lb**	**lb**	**Lb**	**In-lb**	**In-lb**	**In-lb**
Index	**0**	**1**	**2**	**3**	**4**	**5**
	Nominal Capacity Values for a 1000 lb OR6-7					
Capacity	500	500	1000	10000	10000	5000

Format

void fmSetPlatformCapacity(float *data)

Arguments

float *data

A pointer to an array of 6 elements of type float

Returns

Related Functions

fmGetPlatformCapacity

fmGetPlatformCapacity

Description

This function retrieves the platform capacities for the platform attached to the signal conditioner. The function requires a 6 element array of type float. Each array element corresponds to a signal conditioner channel.

These parameters are stored in the signal conditioner in English units.

The signal conditioner has no internal use for these values.

NOTE: If the Gen 5 is attached to a smart platform the Gen 5 will overwrite these parameters with those from the smart platform.

	Platform Capacity Table					
Channel	**Fx**	**Fy**	**Fz**	**Mx**	**My**	**Mz**
Units	**Lb**	**lb**	**lb**	**In-lb**	**In-lb**	**In-lb**
Index	**0**	**1**	**2**	**3**	**4**	**5**
	Nominal Capacity Values for a 1000 lb OR6-7					
Capacity	500	500	1000	10000	10000	5000

Format

void fmGetPlatformCapacity(float *data;

Arguments

float *data

A pointer to an array of 6 elements of type float

Related Functions

fmSetPlatformCapacity

fmSetPlatformBridgeResistance

Description

This function sets the bridge resistances of the platform attached to the signal conditioner. The function requires a 6 element array of type float.

These parameters are stored in the signal conditioner in ohms.

These parameters are always shipped with the platform.

NOTE: If the Gen 5 is attached to a smart platform the Gen 5 will overwrite these parameters with those of the smart platform.

	Platform Capacity Table					
Channel	**Fx**	**Fy**	**Fz**	**Mx**	**My**	**Mz**
Units	**ohms**	**ohms**	**Ohms**	**ohms**	**ohms**	**ohms**
Index	**0**	**1**	**2**	**3**	**4**	**5**
	Nominal Bridge Resistance Values for an OR6-7					
Bridge Resistance	700	700	350	700	700	700

Format

void fmSetPlatformBridgeResistance(float *data)

Arguments

float *data

A pointer to a 6 element array of type float

Related Functions

fmGetPlatformBridgeResistance

fmGetPlatformBridgeResistance

Description

This function retrieves the bridge resistance values for the platform attached to the signal conditioner. The function requires a 6 element array of type float. Each array element corresponds to a signal conditioner channel.

These parameters are stored in the signal conditioner in ohms.

	Platform Bridge Resistance Table					
Channel	**Fx**	**Fy**	**Fz**	**Mx**	**My**	**Mz**
Units	**ohms**	**ohms**	**ohms**	**ohms**	**ohms**	**ohms**
Index	**0**	**1**	**2**	**3**	**4**	**5**
	Nominal Bridge Resistance Values for an OR6-7					
Bridge Resistance	700	700	350	700	700	700

Format

void fmGetPlatformBridgeResistance(float *data)

Arguments

float *data

A pointer to a 6 element array of type float

Related Functions

fmSetPlatformBridgeResistance

fmSetInvertedSensitivityMatrix

Description

The inverted sensitivity array is used to convert micro volts to engineering units and eliminate crosstalk.

This function downloads the inverted sensitivity table to the currently selected signal conditioner. The inverted sensitivity table is a 36 element array of type float. The array should be loaded in row, column order as in the table below.

The English version of the array should be used. This matrix is supplied with every platform shipped.

	Sample inverted Sensitivity Matrix					
Channel	**0**	**1**	**2**	**3**	**4**	**5**
	VFx	**VFy**	**VFz**	**VMx**	**VMy**	**VMz**
	Input to channel i(lb,in-lb) is B(I,j)times the electrical output j(uV,Vex)					
	BP 400600-2000					
Fx	0.6519	-0.0068	-0.0019	0.0009	-0.0017	-0.0003
Fy	0.0090	0.6515	-0.0037	0.0009	0.0005	0.0010
Fz	0.0018	0.0017	2.5523	-0.0062	0.0001	0.0026
Mx	-0.0044	-0.0032	0.0003	12.8281	0.0108	-0.0138
My	0.0725	-0.0032	0.0003	0.0058	10.1358	-0.0140
Mz	0.0649	0.0821	0.0792	0.0123	0.0340	5.4451

NOTE: If the Gen 5 is attached to a smart platform the Gen 5 will automatically load these parameters from the smart platform.

Format

void fmSetInvertedSensitivityMatrix(float *data)

Arguments

float *data

A pointer to a 36 element array of type float

Related Functions

fmSetInvertedSensitivityMatrix

fmGetInvertedSensitivityMatrix

Description

The inverted sensitivity array is used to convert volts to engineering units.

This function retrieves the inverted sensitivity table of the currently selected signal conditioner. the inverted sensitivity table is a 36 element array of type float. The array will be ordered in row, column order as in the table below.

The array is stored in the signal conditioner in English units. A matrix is supplied for every platform shipped.

	Sample inverted Sensitivity Matrix					
Channel	**0**	**1**	**2**	**3**	**4**	**5**
	VFx	**VFy**	**VFz**	**VMx**	**VMy**	**VMz**
	Input to channel i(Ib,in-lb) is B(I,j)times the electrical output j(uV,Vex)					
	BP 400600-2000					
Fx	0.6519	-0.0068	-0.0019	0.0009	-0.0017	-0.0003
Fy	0.0090	0.6515	-0.0037	0.0009	0.0005	0.0010
Fz	0.0018	0.0017	2.5523	-0.0062	0.0001	0.0026
Mx	-0.0044	-0.0032	0.0003	12.8281	0.0108	-0.0138
My	0.0725	-0.0032	0.0003	0.0058	10.1358	-0.0140
Mz	0.0649	0.0821	0.0792	0.0123	0.0340	5.4451

NOTE: If the Gen 5 is attached to a smart platform the Gen 5 will automatically load these parameters from the smart platform.

Format

void fmGetInvertedSensitivityMatrix(float *data)

Arguments

float *data

A pointer to an a36 element array of type float

Related Functions

fmSetInvertedSensitivityMatrix

28.0 The Gen 5 Hardware Function Definitions

fmSetBlink

Description

This function tells the currently selected signal conditioner to blink. The amber light will blink for ten seconds.

Format

void fmSetBlink(void)

fmResetHardware

Description

This function power cycles the currently selected signal conditioner. The signal conditioner will not lose its USB connection during this process.

Format

void fmResetHardware(void)

Related Functions

fmBroadcastResetSoftware
fmResetSoftware

fmBroadcastResetUSB

Description

This function resets the USB pipes from the PC to the signal conditioner for all connect signal conditioners.

Its only current use is resetting the data collection after a two second data collection pause caused by an extended Genlock timeout.

Format

void fmBroadcastResetUSB(void)

Related Functions

fmBroadcastGenlock

29.0 Sample Code

The following sample code is all written in the Microsoft Foundation Classes using Visual Studio 2008. To use any DLL functions AMTIUSBDeviceDefinitions.h must be included as a class header.

```
#include "AMTIUSBDeviceDefinitions.h"
```

DLL Initialization Using a Sleep Statement

```
#include "AMTIUSBDeviceDefinitions.h"

void USBDeviceDlg::InitializeDeviceDLL(void)
{

	fmDLLInit();
	Sleep(250);
	while(fmDLLIsDeviceInitComplete() == 0)
	{
		Sleep(250);
	}

	ret = fmDLLSetupCheck();
		// If return is not 1 configuration has changed
		// Go to funtion description for more information

	ConfigureDataCollection();

}
```

DLL Initialization Using an MFC timer

```
#include "AMTIUSBDeviceDefinitions.h"

void USBDeviceDlg::InitializeDeviceDLL(void)
{

	fmDLLInit();
	TimerID = SetTimer( TimerIDUSBInit, 250, NULL );
}

void USBDeviceDlg::OnTimer(UINT_PTR nIDEvent)
{
	int ret;
	int i;
	ret = 0;

	if(nIDEvent == TimerIDUSBInit)
	{

		ret = fmDLLIsDeviceInitComplete();
		// ret = 0 Wait still initializing
		// ret = 1 Finished, No devices found
		// ret = 2 Finished, device found
		if(ret != 0 )
		{
			KillTimer(TimerIDUSBInit);
			ret = fmDLLSetupCheck();
			// If return is not 1 configuration has changed
			// Go to funtion description for more information

			ConfigureDataCollection();

		}
		else
		{
			timerCount++;
			if(timerCount > MAX_TIMER_ITERATIONS)
			{
				KillTimer(TimerIDUSBInit);
				AfxMessageBox("USB DLL Timeout");

			}
			else
			{
				SetTimer( TimerIDUSBInit, 250, NULL );
			}
		}
	}

}
```

The Acquisition Rate being Broadcast to the Gen 5's

```
#include "AMTIUSBDeviceDefinitions.h"

int acqRate;

acqRate = 1000;
fmBroadcastAcquisitionRate(acqRate);
```

The Platforms being Zeroed

```
#include "AMTIUSBDeviceDefinitions.h"

fmBroadcastZero();
```

Starting Acquisition

```
#include "AMTIUSBDeviceDefinitions.h"

fmBroadcastStart();
```

Stopping Acquisition

```
#include "AMTIUSBDeviceDefinitions.h"

fmBroadcastStop();
```

An MFC dialog class being setup to do data collection using windows messaging

```
#include "AMTIUSBDeviceDefinitions.h"

//Standard Data collection settings that should be set
void USBDeviceDlg::ConfigureDataCollection(void)
{

	HWND h_Wnd;

	//Decided to post messages to a window
	fmDLLPostDataReadyMessages(TRUE);
	h_Wnd = GetSafeHwnd();
	fmDLLPostWindowMessages((HWND) h_Wnd);

	fmDLLSetUSBPacketSize(512 ); // Set the packet size to 512

	fmBroadcastGenlock(0); // Make sure Genlock is off

	fmBroadcastRunMode(0); // Set collection mode to metric

	fmBroadcastResetSoftware(); //Apply the settings
	Sleep(250);

}
```

Windows messaging being set up to do data collection

```
//Setting up the data collection function in the dialog class header message
map to receive widows messages.

Must always use (WM_USER + 108) as an identifier
#define WM_BUFFER_READY                    (WM_USER + 108) //Add this line
…….
…….
afx_msg void OnPaint();
afx_msg HCURSOR OnQueryDragIcon();
long OnBufferReady( DWORD wParam, long lParam );//Add this line
DECLARE_MESSAGE_MAP()
…..
…….


//Setting up the data collection function in the dialog class main body
message map
…….
…….
ON_BN_CLICKED(IDC_B_START, &CdummygenDlg::OnBnClickedBStart)
	ON_BN_CLICKED(IDC_B_STOP, &CdummygenDlg::OnBnClickedBStop)
	ON_MESSAGE( WM_BUFFER_READY,(LRESULT(AFX_MSG_CALL CWnd::*)(WPARAM,
LPARAM))  OnBufferReady //Add this line
	ON_BN_CLICKED(IDC_B_SHUTDOWN, &CdummygenDlg::OnBnClickedBShutdown)
```

```
      ON_BN_CLICKED(IDC_B_BLINK, &CdummygenDlg::OnBnClickedBBlink)
END_MESSAGE_MAP()
…….
……..
```

Collecting Data When a Windows message is recieved

```
// The data collection function received a message indicating data is ready

long USBDeviceDlg::OnBufferReady( DWORD wParam, long lParam )
{
      float *ptr;
      int ret,i;
      CString str,dum;

      //getting the Data
      ret = fmDLLTransferFloatData((float *&)ptr);

      if(ret == 0 )
      {
            return 0;
      }

      str = "";
      dum = "";
      for(i = 0;i < 16;i++)
      {
            dum.Format("%6.3f, %6.3f, %6.3f, %6.3f, %6.3f, %6.3f, %6.3f,
            %6.3f \r\n", ptr[0], ptr[1], ptr[2], ptr[3], ptr[4], ptr[5],
            ptr[6], ptr[7]);
            ptr += 8;
            str+= dum;
      }

      UpdateData(TRUE);
      m_Data = str; //Data being copied to the display
      UpdateData(FALSE);


      return 0;
}
```

User Thread Messaging being set up to do data collection

Sample_4 - This example shows an MFC User thread function being created for data collection using thread messaging.

```
//Setting up the data collection function in the user thread header (.h)
//Always use WM_USER + 109 for the message identifier


#define WM_GENFIVE_THREAD_BUFFER_READY  (WM_USER + 109)//Add this line
…….
……..
……..
……
……
void Cleanup(void);
int SetMessageDestinationWindow( HWND hWnd );
long OnGenDataReadyMsg(DWORD lParem,long rParem); //Add this line
long  OnCommandDispatch(DWORD lParem,long rParem);
……..
……


//Setting up the data collection function in the user thread main body (.cpp)
message map
……..
……
BEGIN_MESSAGE_MAP(CGenThread, CWinThread)
      ON_THREAD_MESSAGE( WM_COMMAND_DISPATCH, (void (AFX_MSG_CALL
CWinThread::*)(WPARAM, LPARAM)) OnCommandDispatch )
    ON_THREAD_MESSAGE( WM_KILL_THREAD, (void (AFX_MSG_CALL
CWinThread::*)(WPARAM, LPARAM)) OnKillThread )
      ON_THREAD_MESSAGE( WM_GENFIVE_THREAD_BUFFER_READY, (void (AFX_MSG_CALL
CWinThread::*)(WPARAM, LPARAM)) OnGenDataReadyMsg )//Add this line
END_MESSAGE_MAP()

…….

……
```

Collecting Data When a User Thread message is recieved

```
//The user thread data collection function

long  CGenThread::OnGenDataReadyMsg(DWORD lParem,long rParem)
{
      float *pSrc;
      float *pSrcA;

      //Get a pointer to the data
      ret = fmDLLTransferFloatData(pSrcA);

      if(ret == 0 )
      {
            return 0;
      }

      //Unload the data

      return( 0 );
}
```

Downloading Some Parameters

```
#include "AMTIUSBDeviceDefinitions.h"

int numdevices
int currentGain[6];
int currentExc[6];
float zeroOffset[6];
int cableLen;


for(i = 0; i<6; i++)
{
      currentGain[i] = 2;  //2000
      currentExc[i]= 0;    //2.5
      zeroOffset[i] = 0.0;
}

numDevices = fmDLLGetDeviceCount();

for(i = 0;i < numDevices;i++)
{
      fmDLLSelectDeviceIndex(i);
      fmSetCurrentExcitations(curentExc);
      fmSetCurrentGains(currentGain);
      fmSetChannelOffsetsTable((float*)zeroOffset);
}

fmSetCableLength(cableLen);

fmResetSoftware();
```

Retrieving Some Parameters

```
#include "AMTIUSBDeviceDefinitions.h"

int i;

char len[30];
char width[30];
char model[30];
char serial[30];
char theDate[30];

float lenWdth[2];
float offsets[3];
float sen[36];
float bridgeResis[6];

memset(len,'\0',30);
memset(width,'\0',30);
memset(model,'\0',30);
memset(serial,'\0',30);
memset(theDate,'\0',30);

offsets[0] = 0.0;
offsets[1] = 0.0;
offsets[2] = 0.0;

for(i = 0;i < 6;i++)
{
      bridgeResis[i] = 0.0;
}

for(i =0;i< 36;i++)
{
      sen[i] = 0.0;
}

fmDLLSelectDeviceIndex(0);
fmGetPlatformModelNumber(model);
fmGetPlatformSerialNumber(serial);
fmGetPlatformDate(theDate);

fmGetPlatformLengthAndWidth((char *)len,(char *) width);

fmGetPlatformXYZOffsets(offsets);

fmGetPlatformBridgeResistance(bridgeResis);
fmGetInvertedSensitivityMatrix(sen);
```

Auto-Ordering the Dataset Platform Order using an MFC timer

```
#include "AMTIUSBDeviceDefinitions.h"


int oldRunMode;

void USBDeviceDlg::StartPlatformOrdering(void)
{
	fmDLLSetDataFormat(0);
	oldRunMode = fmDLLGetRunMode();
	fmBroadcastRunMode(4);
	fmBroadcastPlatformOrderingThreshold(30);
	fmBroadcastResetSoftware();

	Sleep(1000);
	fmBroadcastZero();
	Sleep(500);
	fmDLLStartPlatformOrdering();
	fmBroadcastStart();
	TimerID = SetTimer( TimerIDPltfrmOrder, 500, NULL );

}

void USBDeviceDlg::OnTimer(UINT_PTR nIDEvent)
{

	if(fmDLLIsPlatformOrderingComplete())
	{

		KillTimer(TimerIDPltfrmOrder);

		fmBroadcastRunMode(oldRunMode);
		fmBroadcastResetSoftware();
		Sleep(500);


	}
	else
	{
		SetTimer(TimerIDPltfrmOrder, 250, NULL );
	}
}
```

Appendix A – Integration of the AMTI Optima Signal Conditioner into the USB Device SDK

Introduction

In September, 2011 AMTI launched the Optima line of force plate systems introducing an unparalleled level of accuracy for biomechanics force platform measurement. The following section covers additions to the USB Device SDK for integrating the Optima line of signal conditioners.

All third party software modifications to integrate the new Optima signal conditioner will occur in the signal conditioner initialization section of their code.

The Optima Binary Calibration File

An AMTI binary calibration file (*.bcf), is shipped with every Optima force platform. The calibration file is too large to store on the platform smart chip and therefore, has to be written from the PC to the signal conditioner whenever a new Optima signal conditioner, platform combination is detected.

Optima binary calibration files should be installed on the PC through the use of the AMTI System Configuration program. The AMTI System Configuration program is a utility program which ships with all AMTI USB Devices. It is used to configure both the USB Device DLL and all AMTI USB devices. When this program encounters a new Optima signal conditioner, platform combination, it requests the appropriate binary calibration file from the user. It then stores the calibration file on the PC while simultaneously downloading it to the signal conditioner. If later the DLL needs the binary file again it can simply retrieve it from the storage location on the PC.

When storing the initial binary calibration file the AMTI System Configuration program creates a storage folder and records the folder location in the system registry.

For Windows 7 – 64 the registry location is:

HKEY_CURRENT_CONFIG->SOFTWARE->Wow6432Node ->AMTI->HPS

For Windows XP - 32 the registry location is:

HKEY_CURRENT_CONFIG->SOFTWARE->AMTI->HPS

The current PC folder location for storing binary calibration files is C:\AMTI\HPS

How the AMTI USB Device DLL Initializes an Optima Signal Conditioner

When an Optima signal conditioner is powered on it queries the connected force platform to determine if the platform is equipped with smart chip technology. If the platform is so equipped the signal conditioner uploads platform identification information, including the platform serial number. The signal conditioner compares the serial number of the platform to that of the calibration table last stored in local memory. If the serial numbers match the process ends; if there is no match the signal conditioner will require the correct calibration file to be downloaded from the PC.

When the USB Device DLL initializes it detects all connected AMTI USB devices. If an Optima signal conditioner is detected, the SDK will query the signal conditioner to determine if the correct platform calibration table is present. If the correct calibration table is not present the DLL will check the PC registry to obtain the folder location of the Optima calibration files. If the folder location exists, the DLL will search the folder for the correct binary calibration file (*.bcf). If the file is found, the DLL will automatically download it to the Optima. The calibration file download can take up to 15 seconds, meaning the initialization process of the USB Device DLL can take up to 15 seconds.

There are four new SDK functions associated with Optima technology. If these functions are not integrated into third party applications, their software will run fine provided the AMTI System Configuration software was previously used to install the Optima signal conditioners with the correct binary calibration files. If this has not taken place, depending on implementation, the potential 15 second file download time could be problematic.

Gen 5 Compatibility

All Gen 5 functions apply to the Optima signal conditioner except for the following

fmSetPlatformDate
fmSetPlatformModelNumber
fmSetPlatformSerialNumber
fmSetPlatformLengthAndWidth
fmSetPlatformXYZOffsets
fmGetPlatformXYZOffsets
fmSetPlatformCapacity
fmSetPlatformBridgeResistance

fmGetInvertedSensitivityMatrix
fmSetInvertedSensitivityMatrix

The Optima signal conditioner doesn't have as full a range of acquisition rates as the Gen 5. The acquisition rates highlighted in orange are common to both the Gen 5 and Optima. You will note the Gen 5 has three additional high speed rates highlighted in blue.

Acquisition Rates									
2000	1800	1500	1200	1000	900	800	600	500	450
400	360	300	250	240	225	200	180	150	125
120	100	90	80	75	60	50	45	40	30
25	20	15	10						

Optima Only Functions

The following list of functions applies to the Optima signal conditioner only.

fmBroadcastCheckOptima
fmOptimaGetStatus
fmOptimaDownloadCalFile
fmIsOptimaDownloadComplete

fmBroadcastCheckOptima

Description

This function checks all connected Optima signal conditioners and reports whether they are ready. It is primarily checking for correct calibration files. If any Optima's are not ready the function returns the number of Optima's not ready to run and their current device indexes.

The function should be called shortly after ***fmDLLIsDeviceInitComplete*** returns success indicating successful SDK initialization.

Format

```
long fmBroadcastCheckOptima(long *data );
```

Arguments

long *data

The argument is a pointer to a 16 element array of type long, each element containing the device index for an Optima signal conditioner that is not ready. The array will only fill as many elements as indicated in the function return parameter. Call ***fmDLLSelectDeviceIndex*** and ***fmOptimaGetStatus*** to determine why any individual Optima are not ready.

Return

Long

Returns the number of Optima signal conditioners which are not ready. If zero all Optima's are ready to run.

Related Functions

fmDLLIsDeviceInitComplete
fmDLLSelectDeviceIndex
fmOptimaGetStatus

fmOptimaGetStatus

Description

The function ***fmOptimaGetStatus*** should be called after ***fmBroadcastCheckOptima*** to check the status code of any Optima signal conditioner that is not ready.

Format

```
long fmOptimaGetStatus(void);
```

Returns

Long

Returns the status of the currently selected device

0: The Signal conditioner is a Gen 5
1: The signal conditioner is an Optima and the calibration file is correct
2: Bad CRC check – the calibration file is corrupted
3: The calibration file does not match the platform
4: The Optima signal conditioner is using factory default settings not a calibration file.
5: The Optima signal conditioner is not attached to an Optima platform.

Related Functions

fmBroadcastCheckOptima
fmDLLSelectDeviceIndex

fmOptimaDownloadCalFile

Description

This function reads the system registry to determine the folder in which Optima binary calibration files (*.bcf) are stored on the PC. It then searches the folder for the binary calibration file required for the currently selected Optima signal conditioner. If the file is found, the function will begin the process of downloading the calibration file to the signal conditioner. It can take up to 15 seconds to download a binary calibration file from the PC to the signal conditioner. When a binary calibration file is downloaded it's written directly to the flash. It does not need to be saved.

Format

```
long fmOptimaDownloadCalFile(BOOL bDownload)
```

Arguments

BOOL bDownload

Currently this argument is not being utilized. Always set to 1.

Returns

Long

0 – The file download has begun
1 – There are no connected devices
2 – The connected device is not an Optima signal conditioner
3 – No binary calibration file (*.bcf) was found

Related Functions

fmIsOptimaDownloadComplete
fmDLLSelectDeviceIndex

fmIsOptimaDownloadComplete

Description

This function should be called to determine if the binary calibration file (*.bcf) download which began with a call to ***fmOptimaDownloadCalFile*** has completed.

Format

```
long fmIsOptimaDownloadComplete( void );
```

Returns

Long

0 – File download in process
1 – File download is complete
2 – File not found on PC
3 – File version not supported
4 – Bad CRC Check
5 – Other

Related Functions

fmDLLSelectDeviceIndex
fmOptimaDownloadCalFile