

3DM-GX2[®] Orientation SDK

for Inertia-Link[®] and 3DM-GX2[®]



©2008 by MicroStrain, Inc.
459 Hurricane Lane, Suite 102
Williston, VT 05495
Phone 802-862-6629
Fax 802-863-4093
www.microstrain.com
support@microstrain.com

ISSUED: 19 November 2008

Table of Contents

OVERVIEW	4
DISCLAIMER.....	5
SDK SUPPORT	5
DATA COMMUNICATIONS PROTOCOL	6
ANSI C IMPLEMENTATION.....	8
ANSI C DEFINITION	8
IMPLEMENTATION	8
Main Implementation.....	8
Visual Studio C++ Project	9
Release	10
API.....	11
Command Set functions contained in i3dmgx2_Cmd.c	11
Communication functions contained in i3dmgx2SerialWin.c	13
Utility functions contained in i3dmgx2Utils.c.....	14
IMPLEMENTATION SAMPLE.....	16
CONTROL FLOW	18
I. Overview, file identifiers, for Windows serial port connections with 3dmgx2.....	18
II. Setting up a Serial Comm Port for Windows.....	18
III. Executing a 3dmgx2 command using Windows Serial Driver	19
IV. Closing the Windows Serial Driver.....	20
VISUAL STUDIO C++ 2003/2008 EXPRESS CODE SAMPLES	21
LABVIEW 7.1.1 CODE SAMPLES.....	25
VISUAL STUDIO VISUAL BASIC 2005 CODE SAMPLES	27
VISUAL BASIC 6.0 CODE SAMPLES.....	34
CP210X USB TO UART BRIDGE CONTROLLER	40
SUGGESTED DEBUGGING TOOLS	41
LOOKRS232	41
SERIAL PORT MONITOR.....	42
COMM OPERATOR	42
SUPPORT	43
OVERVIEW.....	43
WEB	43
EMAIL.....	43
TELEPHONE	43
SKYPE	43

Overview

The MicroStrain 3DM-GX2[®] Orientation Software Development Kit (SDK) is a set of development tools that allows a user to create applications which communicate with MicroStrain's Inertia-Link[®] and 3DM-GX2[®] orientation sensors.

The SDK is targeted at providing the user with the tools required to build Windows applications for the RS-232, USB and RS-422 (wired) physical communication interfaces of the MicroStrain orientation sensors.*

The SDK consists of the following major components:

- Data Communications Protocol manual
- ANSI C Implementation
- Application Programming Interface (API)
- Sample Code:
 - o Visual Studio C++ 2003/2008 Express
 - o LabVIEW 7.1.1
 - o Visual Studio Visual Basic 2005
 - o Visual Basic 6.0

It is expected that the user will have:

- a working knowledge of the MicroStrain Inertia-Link[®] and/or 3DM-GX2[®] orientation sensor;
- experience coding in the programming language in use;
- a working knowledge of communication ports.

Please review the Disclaimer and SDK Support on the following page.

Good luck and good coding.

* The MicroStrain wireless 2.4 GHz physical communication interface for the orientation sensors is elsewhere supported.

Disclaimer

The SDK is provided “as is” and any expressed or implied warranties, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose are disclaimed. In no event shall MicroStrain or its contributors be liable for any direct, indirect, incidental, special, exemplary, or consequential damages (including, but not limited to, procurement of substitute goods or services; loss of use, data, or profits; or business interruption) however caused and on any theory of liability, whether in contract, strict liability, or tort (including negligence or otherwise) arising in any way out of the use of this SDK, even if advised of the possibility of such damage. MicroStrain will make every effort to amplify the instructions contained in the SDK but will neither undertake to detail the functioning of the hardware or firmware in the orientation sensor family nor debug the user’s code.

SDK Support

MicroStrain will only accept requests for support for this SDK via email to support@microstrain.com. Please include the following information to expedite the support:

- Your name
- Your company or organization
- Model name, serial number, firmware version of the orientation sensor(s)
- Physical interface
- Coding language
- Hardware platform
- Operating System
- Your issue(s) in detail including screen shots, data files, links and any other attachments that will aid in understanding

Data Communications Protocol

The Data Communications Protocol is a separate document which describes how to communicate with the MicroStrain Inertia-Link[®] and 3DM-GX2[®] orientation sensors. The protocol changes from time-to-time in lockstep with firmware changes on the various orientation sensors. The most current protocol is maintained on the MicroStrain web site at: <http://www.microstrain.com/pdf/dcp/Inertia-Link-3DM-GX2-data-communications-protocol.pdf>. The Data Communications Protocol manual contains a version number on the cover, a revision date on the second page and a statement concerning which firmwares are supported. The user should confirm the firmware version of the particular orientation sensor being used.

The MicroStrain Inertia-Link[®] and 3DM-GX2[®] orientation sensors are available with many communication interface options including USB, RS-232, RS-422, and wireless 802.15.4 (2.4 GHz). The Data Communications Protocol supports communication with all of these interfaces; however, this SDK supports communication with only wired versions (USB, RS-232, RS-422) of the sensors. Wireless support is elsewhere provided.

The Data Communications Protocol is a set of serial commands and responses designed specifically for MicroStrain's orientation sensors. For the most part, the communications protocol consists of simple single byte binary commands with fixed length binary data records as replies. Most replies include an "echo" byte and a checksum to do simple data integrity checks. All communications with the sensors are accomplished using a standard serial "COM" port. A typical command and response, as an example Euler angles, is demonstrated in **Table 1** below. A single byte 0xCE is sent by the host to the sensor. The sensor responds with a 19 byte packet containing the header (echo) byte, 4 bytes representing the Roll value as a float, 4 bytes for Pitch, 4 bytes for Yaw, 4 bytes indicating the sample time from the sensor's on-board clock, and a 2 byte checksum to determine packet integrity.

Euler Angles (0xCE) –Single Byte Form

Command	1 byte
Command Byte	0xCE
Response	19 bytes
Byte 1	Header = 0xCE
Bytes 2-5	Roll
Bytes 6-9	Pitch
Bytes 10-13	Yaw
Bytes 14-17	Timer
Bytes 18-19	Checksum

Table 1

The Data Communications Protocol is the basis of this Software Development Kit. The ANSI C Implementation, the Application Programming Interface and the various sample codes, all later described and detailed in this manual, are all constructed directly from the Data Communications Protocol.

We emphasize that careful reading of the Data Communications Protocol manual is good practice and will speed the user's understanding and success in creating applications.

ANSI C Implementation

ANSI C Definition

http://en.wikipedia.org/wiki/ANSI_C

ANSI C is the standard published by the American National Standards Institute (ANSI) for the C programming language. Software developers writing in C are encouraged to conform to the requirements in the standard, as it encourages easily portable code. ANSI C is now supported by almost all the widely used compilers. The vast majority of all C language code being produced adheres to the ANSI C standard. Any program written *only* in standard C and without any hardware dependent assumptions is virtually guaranteed to compile correctly on any platform with a conforming C implementation. Without such precautions, most programs may compile only on a certain platform or with a particular compiler, due, for example, to the use of non-standard libraries, such as GUI libraries, or to the reliance on compiler- or platform-specific attributes such as the exact size of certain data types and byte endianness.

Implementation

The 3DM-GX2[®] Orientation SDK ANSI C Implementation contains C source files, C header files, compiling batch files, readme files, Visual Studio C++ project files, executables, and build log files. All C source files and header files are well commented.

In the Main Implementation and from a hierarchical point of view, **[project_dir]main.c** is the presentation module. It requests and receives data from the Inertia-Link[®] or 3DM-GX2[®] by calling Command Set functions from **i3dmgx2_Cmd.c**, Communication functions from **i3dmgx2_SerialWin.c** and Parsing functions from **i3dmgx2_Utils.h**. Error Handling is contained within most functions and is supported by **i3dmgx2_Errors.h** and **i3dmgx2_Utils.h**. A detailed operation chart can be found in the *Control Flow* section of this manual.

A detailed listing of all files follows:

Main Implementation

File	compileWin.bat
Type	MS-DOS Batch File
Purpose	Batch file to compile MS-Windows application

File	[project_dir]main.c
Type	C Source
Purpose	Presentation of data

File	i3dmgx2.exe
Type	Application
Purpose	Executable program.

File	i3dmgx2_Cmd.c
Type	C Source
Purpose	Command Set functions

File	i3dmgx2_Cmd.h
Type	C/C++ Header
Purpose	Declarations in support of Command Set functions

File	i3dmgx2_Errors.h
Type	C/C++ Header
Purpose	Declarations in support of Error Handling functions

File	i3dmgx2_Serial.h
Type	C/C++ Header
Purpose	Declarations in support of Communications functions

File	i3dmgx2_SerialWin.c
Type	C Source
Purpose	Communication functions

File	i3dmgx2_Utils.c
Type	C Source
Purpose	Parsing and Error Handling functions

File	i3dmgx2_Utils.h
Type	C/C++ Header
Purpose	Declarations in support of Parsing and Error Handling functions

File	readme.txt
Type	Read Me
Purpose	Provides general information about the implementation.

Table 2**Visual Studio C++ Project**

File	i3dmgx2.ncb
Type	VC++ Intellisense Database
Purpose	Identifies dynamic user code information features of Visual C++ such as Quick Information, Auto Completion, and Parameter Help.

File	i3dmgx2.sln
Type	Microsoft Visual Studio Solution
Purpose	Stores information about the projects that make up this solution, source control settings, and other global settings that apply to all of the projects in the solution.

File	i3dmgx2.suo
Type	Visual Studio Solution User Option
Purpose	Stores user settings.

File	i3dmgx2.vcproj
Type	VC++ Project
Purpose	Stores items that represent the references, data connections, folders, and files to create an application.

File	readme.txt
Type	Read Me
Purpose	Project overview

Table 3**Release**

File	BuildLog.htm
Type	HTML Document
Purpose	A log of the Command lines, Output Window and Results of the project build.

File	i3dmgx2.exe
Type	Application
Purpose	Executable program.

File	vc70.idb
Type	VC++ Minimum Rebuild Dependency File
Purpose	Stores release info in binary format

Table 4

API

The Orientation Application Programming Interface (API) is a set of functions that the ANSI C Implementation provides to support software development for the Inertia-Link[®] and 3DM-GX2[®]. The API is C language dependent; it uses the syntax and elements of the C programming language to make the API convenient to use in this particular language. The API is constructed in such a fashion as to allow compilation for the Microsoft Windows operating system.

A detailed listing of all API functions follows:

Command Set functions contained in i3dmgx2_Cmd.c

Function	int i3dmgx2_openPort (int portNum, int baudrate, int size, int parity, int stopbits, int inputBuff, int outputBuff)
Use	Open a serial communications port.
Return	Port number

Function	void i3dmgx2_closeDevice (int portNum)
Use	Closes the device.
Return	No return

Function	int i3dmgx2_RawSensor (int portNum, BYTE* pRecord)
Use	Executes the orientation sensor 0xC1 command request
Return	31 byte 0xC1 data packet

Function	int i3dmgx2_AccelAndAngRate (int portNum, BYTE* pRecord)
Use	Executes the orientation sensor 0xC2 command request
Return	31 byte 0xC2 data packet

Function	int i3dmgx2_DeltaAngAndVelocity (int portNum, BYTE* Bresponse)
Use	Executes the orientation sensor 0xC3 command request
Return	31 byte 0xC3 data packet

Function	int i3dmgx2_OrientMatrix (int portNum, BYTE* Bresponse)
Use	Executes the orientation sensor 0xC5 command request
Return	43 byte 0xC5 data packet

Function	int i3dmgx2_OrientUpMatrix (int portNum, BYTE* Bresponse)
Use	Executes the orientation sensor 0xC6 command request
Return	43 byte 0xC6 data packet

Function	int i3dmgx2_ScaledMagVec (int portNum, BYTE* Bresponse)
Use	Executes the orientation sensor 0xC7 command request
Return	19 byte 0xC7 data packet

Function	int i3dmgx2_AccelAngOreint (int portNum, BYTE* Bresponse)
Use	Executes the orientation sensor 0xC8 command request
Return	67 byte 0xC8 data packet

Function	int i3dmgx2_AccAngMagRate (int portNum, BYTE* Bresponse)
Use	Executes the orientation sensor 0xCB command request
Return	43 byte 0xCB data packet

Function	int i3dmgx2_GetFullMatrix (int portNum, BYTE* Bresponse)
Use	Executes the orientation sensor 0xCC command request
Return	79 byte 0xCC data packet

Function	int i3dmgx2_captureGyroBias (int portNum, short sampt, BYTE *BiasBuff, BYTE* Bresponse)
Use	Executes the orientation sensor 0xCD command request
Return	19 byte 0xCD data packet

Function	int i3dmgx2_WriteGyroBias (int portNum, BYTE *BiasBuff, BYTE* Bresponse)
Use	Executes the orientation sensor 0xCA command request
Return	19 byte 0xCA data packet

Function	int i3dmgx2_EulerAngles (int portNum, BYTE* Bresponse)
Use	Executes the orientation sensor 0xCE command request
Return	19 byte 0xCE data packet

Function	int i3dmgx2_EulerAngRate (int portNum, BYTE* Bresponse)
Use	Executes the orientation sensor 0xCF command request
Return	31 byte 0xCF data packet

Function	int i3dmgx2_TransferNonVolatile (int portNum, int transfer, BYTE *Bresponse)
Use	Executes the orientation sensor 0xD0 command request
Return	9 byte 0xD0 data packet

Function	int i3dmgx2_Tempatures (int portNum, BYTE* Bresponse)
Use	Executes the orientation sensor 0xD1 command request
Return	15 byte 0xD1 data packet

Function	int i3dmgx2_GyroStab (int portNum, BYTE* Bresponse)
Use	Executes the orientation sensor 0xD2 command request
Return	43 byte 0xD2 data packet

Function	int i3dmgx2_DeltaAngVel (int portNum, BYTE* Bresponse)
Use	Executes the orientation sensor 0xD3 command request
Return	43 byte 0xD3 data packet

Function	int i3dmgx2_getEEPROMValue (int portNum, char address, int readFlag, int *readval)
Use	Executes the orientation sensor 0xE5 command request
Return	5 byte 0xE5 data packet

Function	int i3dmgx2_getDeviceIdentiy (int portNum, char flag, BYTE* Bresponse)
Use	Executes the orientation sensor 0xEA command request
Return	20 byte 0xEA data packet

Function	int i3dmgx2_getFirmwareVersion (int portNum, char *firmware)
Use	Executes the orientation sensor 0xE9 command request
Return	7 byte 0xE9 data packet

Table 5**Communication functions contained in i3dmgx2SerialWin.c**

Function	int openPort (int portNum, int inputBuff, int outputBuff)
Use	Opens port and initializes internal buffer sizes
Return	Port status

Function	void closePort (int portNum)
Use	Closes port
Return	No return

Function	int setCommParameters (int portNum, int baudrate, int charsize, int parity, int stopbits)
Use	Initializes port settings
Return	Port status

Function	int setCommTimeouts (int portNum, int readTimeout, int writeTimeout)
Use	Sets port internal timeouts
Return	Port status

Function	int sendBuffData (int portNum, char *command, int commandLength)
Use	Sends command to device
Return	Device status

Function	int receiveData (int portNum, char *response, int responseLength)
Use	Receives data packet of size specified by response length
Return	Device status and data packet

Function	int purge_port (int portNum)
Use	Clears internal port input and output buffers
Return	Port status

Table 6**Utility functions contained in i3dmgx2Utils.c**

Function	int calcChecksum (char* buffer, int length)
Use	Calculate checksum on a received data buffer.
Return	Checksum value

Function	unsigned short i3dmgx2_Checksum (const BYTE* pBytes, int count)
Use	Calculate checksum on a received data buffer.
Return	Checksum status

Function	int TestByteOrder ()
Use	Tests byte alignment to determine Endian Format of local host.
Return	Endianness of host

Function	float FloatFromBytes (const BYTE* pBytes)
Use	Converts bytes to Float.
Return	Floating point value

Function	short convert2short (char* buffer)
Use	Convert two adjacent bytes to an integer.
Return	Signed Short 2 byte integer

Function	unsigned short convert2ushort (char* buffer)
Use	Convert two adjacent bytes to a short.
Return	Unsigned short 2 byte integer

Function	long convert2long (BYTE* plbyte)
Use	Convert four adjacent bytes to a signed long.
Return	Signed long 4 byte integer

Function	unsigned long convert2ulong (BYTE* plbyte)
Use	Convert four adjacent bytes to a unsigned long.
Return	Unsigned long 4 byte integer

Function	char * explainError (int errornum)
Use	Provide a text explanation for an error code.
Return	Text

Table 7

Implementation Sample

By way of further explanation, we detail the steps we take to build the API. We are presented with the command (for example), Raw Accelerometer and Angular Rate Sensor Outputs in the Data Communications Protocol (see **Table 8**). From that we develop a function written in C code (see **Code Sample 1**). From that we add the function to our API (see **Table 9**).

Raw Accelerometer and Angular Rate Sensor Outputs (0xC1)

Function:	The Inertia-Link® or 3DM-GX2™ will output a data record containing the raw sensor voltages in the form of A/D converter codes (0 to 65535).
Command Byte:	0xC1
Command Data:	None
Response:	31 bytes defined as follows
Byte 1	<i>Header</i> = 0xC1
Bytes 2-5	<i>RawAccel</i> ₁
Bytes 6-9	<i>RawAccel</i> ₂
Bytes 10-13	<i>RawAccel</i> ₃
Bytes 14-17	<i>RawAngRate</i> ₁
Bytes 18-21	<i>RawAngRate</i> ₂
Bytes 22-25	<i>RawAngRate</i> ₃
Bytes 26-29	<i>Timer</i>
Bytes 30-31	<i>Checksum</i>

Table 8

```

/* -----
-
* i3dmgx2_RawSensor  0xC1
*
* parameters    portNum : the number of the sensor device (1..n)
*                pI3Record : struct to receive floating point values for
*                           raw Accel (x,y,z) and raw Ang (x,y,z)
*
* returns:      errorCode : I3DMGX2_OK if succeeded, otherwise returns
*                           an error code.
* -----
*/
int i3dmgx2_RawSensor(int portNum, BYTE* pRecord) {
    int responseLength = 31;
    int status;
    unsigned short wChecksum = 0;
    unsigned short wCalculatedChecksum = 0;
    char cmd = CMD_RAW_ACCELEROMETER; /* value is 0xC1 */
    int i = 0;

    status = sendBuffData(portNum, &cmd, 1);

```

```

if (DEBUG) printf("i3dmgx2_send: tx status : %d\n", status);
if (status == I3DMGX2_COMM_OK) {
    status = receiveData(portNum, &pRecord[0], responseLength);
if (DEBUG) printf("Raw Sensor i3dmgx2_send: rx status : %d and
    responseLength %d\n", status, responseLength);

if (status==I3DMGX2_COMM_OK) {
    wChecksum = convert2ushort(&pRecord[responseLength-2]);
    wCalculatedCheckSum = i3dmgx2_Checksum(&pRecord[0],
        responseLength-2);
        //calculate the checksum, 29 = 31-2 don't include the
        checksum bytes
    if(wChecksum != wCalculatedCheckSum)
        return status = I3DMGX2_CHECKSUM_ERROR;
    }else
        return status = I3DMGX2_COMM_READ_ERROR;
}else
    status = I3DMGX2_COMM_WRITE_ERROR;
return status;

```

Sample Code 1

Function	int i3dmgx2_RawSensor (int portNum, BYTE* pRecord)
Use	Executes the orientation sensor 0xC1 command request
Return	31 byte 0xC1 data packet

Table 9

Control Flow

I. Overview, file identifiers, for Windows serial port connections with 3dmgx2

[project_dir]main.c Main() controls the steps and flows, initiates the setup of the serial port and each of the 3dmgx2 commands.

i3dmgx2_CMD.c calls for the 3dmgx2 commands this program handles the formatting for input and output to and from the serial driver in order to execute the 3dmgx2 command set.

i3dmgx2_SerialWin.c handles the Windows serial port commands: opens, configures, reads, writes and closes the serial port. The calls in this file are specific to the Windows operating system.

i3dmgx2_Utils.c handles conversion functions from Big Endian to Little Endian, supports the windows Little Endian formats.

File names are in **bold** and underlined Windows specific Serial functions are underlined:
<---- and ----> identifies the direction of control which is passed to program

II. Setting up a Serial Comm Port for Windows

[project_dir]main.c (get serial port identifier passed to program on command line by user performs check to insure its a valid port number then calls
----->

i3dmgx2_cmd.c i3dmgx2_openPort with the port number
----->

i3dmgx2_SerialWin.c ----> calls the Windows specific serial port commands,
CreateFile (with specified port number and the following values:
GENERIC_READ | GENERIC_WRITE,
0, // exclusive access
NULL, // no security
OPEN_EXISTING,
0, // no overlapped I/O
NULL); // null template
if successful it calls
SetupComm with buffer sizes 128, 128

Returns port handle on success error on failure
<-----

i3dmgx2_Cmd.c If success setup the port handle with the baud rate
wordlength stopbits and parity with setCommParameter
----->

i3dmgx2_SerialWin.c -----> calls GetCommState to obtain the
Windows DCB serial port struc and sets the baudrate, charsize,
parity and stopbits with SetCommState returns either success or
error
<-----

i3dmgx2_Cmd.c If success, calls setCommTimeouts with read and write timeout
values
----->

i3dmgx2_SerialWin.c-----> calls Windows GetCommTimeouts to
obtain timeout struct sets the timeouts with windows
SetCommTimeouts returns either success or error

<-----

<-----

[project_dir]main.c if success prints the port number, on error exits as it was unable to obtain a
valid port number

On success the serial port has been opened and initialized, the application is ready to
execute the 3dmgx2 command set

III. Executing a 3dmgx2 command using Windows Serial Driver

Example uses the command Raw Sensor Output 0xC1:

[project_dir]main.c Get the Raw Sensor Output rawAccel x y z and RawAng x y z
i3dmgx2_accelAndAngRate(portNum, &i3Record)
----->

i3dmgx2_cmd.c RawSensor function:
calls sendBuffData(portNum, 0xC1, 1)
----->

i3dmgx2_SerialWin.c -----> Executes Windows Serial command
WriteFile(portHandle, command, length
and gets #of bytes written)
returns success or failure

<-----

i3dmgx2_cmd.c If Success:
calls receiveData(portNum, gets receiveBuff,
length_to_receive)
(If error returns control to i3dmgx2.c for handling)
----->

i3dmgx2_SerialWin.c -----> Executes Windows Serial command
ReadFile(portHandle, get_response,
length, and gets #of bytes read)

returns success or failure

 $\angle$ -----

i3dmgx2_cmd.c

If Success:

verify checksum, if success:

convert buffer into floating point values, NOTE data is received

in Big Endian format and on a Windows

platform Little Endian format is used, The conversion is handled

in the file `i3dmgx2Utils.c`

prints formatted response and returns success or failure

 $\angle$ -----

[project_dir\]main.c

Print out error or success and execute the next command in list

IV. Closing the Windows Serial Driver

[\project_dir\]main.c

Calls the function `i3dmgx2_closeDevice(portNum)`

----->

i3dmgx2_cmd.c

calls the function `closePort(portNum)`

obtains the port number from the device map.

----->

i3dmngx2_SerialWin.c

---> Executes Windows Serial command

CloseHandle(portHandle)

<-----

i3dmgx2_cmd.c

< -----

[project_dir]\main.c

exits

Visual Studio C++ 2003/2008 Express Code Samples

The C++ Code Samples were constructed with the Microsoft C++ Development Environment 2003 version 7.1.6030 with Microsoft .NET Framework 1.1 version 1.1.4322 SP1 and Microsoft C++ 2008 Express Edition version 9.0.30729.1 SP with Microsoft .NET Framework version 3.5 SP1 using only objects and components native to the IDEs. No third party components or objects were used.

The Visual Studio C++ 2003/2008 Express Code Samples are written directly from the Data Communications Protocol.

The code shown and explained below is reflective of the entire code sample offering.

The code sample spawns a Console window as shown in **Figure 1**. The Console window indicates the command being demonstrated (in this case, the 0xC2 packet). The Console window indicates which comm port is being used, whether the host is using Big Endian or Little Endian byte order, the output values of the sensor and the timestamp.

```

C:\toss\SDK_Master\Windows\Code\C\Examples\Read Acceleration & Angular Rate\projectV...
3DM-GX2 Read Acceleration and Angular Rate
No port specified. Scanning for device...
Found device on COM port #10
Using COM Port #10 Device Number is #1
<Local Host is in Little Endian format>

0xC2 Accel and Ang Output
      Accel X      Accel Y      Accel Z
      -0.011912      0.003623      -0.993416
      Ang X      Ang Y      Ang Z
      0.028888      -0.000719      -0.007737

Time Stamp: 58568377
Hit return to exit...
  
```

Figure 1

Pseudo code

- The application is given either a single argument or run without argument.
- If it is given an argument, it assumes it is a comm port number, attempts to connect to that port number and checks that it is a valid port between 1 and 256.
- If no argument is given, it will scan all ports, looking for valid ports and attempt a connection.
- Upon connection, the application configures the port with baud rate, parity, stop bits, timeout, etc.
- If successful, it sends a polling command to the sensor.

- If a data packet is returned from the sensor, the application will perform a checksum.
- If the checksum is successful, it will parse the data packet into the X, Y, Z acceleration and X, Y, Z angular rate values and display them to the screen.
- The application will terminate.

```

/*-----
* Read Acceleration and Angular Rate
*-----
*/

#include <stdio.h>
#include "ms_basic_type.h"
#include "i3dmgx2Errors.h"
#include "i3dmgx2Serial.h"
#include "i3dmgx2_Cmd.h"
#include "i3dmgx2Utils.h"
#include "win_scanports.h"

/*-----
*/
int main(int argc, char **argv) {

    s16 portNum;
    s16 portNum = 0;
    s16 i;
    u16 value=0;
    s16 id_flag = 0;
    s16 errorCode;
    s16 tryPortNum = 1;
    unsigned char Record[79]; //record returned from device read
                             //where max size is 79
    C2Accel_AngRecord Accel_AngRecord;

    printf("\n  3DM-GX2 Read Acceleration and Angular Rate\n");

    /*----- If user specifies a port, then use it */
    if (argc > 1) {
        tryPortNum = atoi(argv[1]);
        if (tryPortNum < 1 || tryPortNum > 256) {
            printf("  usage: i3dmgx2 <portNumber>\n");
            printf("    valid ports are 1..256, default is\n");
            printf("                                1\n");
            exit(1);
        }
    }
    /*----- If no port specified, then scan ports to see if we can
                             find a connected device */
    else {
        printf("\n  No port specified. Scanning for device...\n");
        tryPortNum = scanports();
        if (tryPortNum < 1 || tryPortNum > 256) {
            printf("\n  No recognized devices attached!\n");
            goto Exit;
        }
    }
}

```

```

    }
    else {
        printf("\n    Found device on COM port %#d\n",
               tryPortNum);
    }
}

/*----- open a port, map a device */
portNum = i3dmgx2_openPort(tryPortNum, 115200, 8, 0, 1, 1024,
                           1024);

if (portNum<0) {
    printf("    port open failed.\n");
    printf("    Comm error %d, %s: ", portNum,
           explainError(portNum));
    goto Exit;
}
printf("\n    Using COM Port %#d \n", portNum);

/*----- Set Comm Timeout values */
errorCode = setCommTimeouts(portNum, 1, 1); /* Read & Write
                                              timeout values */

if (errorCode!=I3DMGX2_COMM_OK) {
    printf("    setCommTimeouts failed on port:%d with
           errorcode:%d\n",portNum,errorCode);
    goto Exit;
}

/*----- Disclose the byte order of host */
if( TestByteOrder() !=BIG_ENDIAN)
    printf("    (Local Host is in Little Endian format)\n");
else
    printf("    (Local Host is in Big Endian format)\n");
printf("\n");

/*----- 0xC2 Accel and Ang rate Output --- Accel x y z and
                                              AngRate x y z */
printf("\n    0xC2 Accel and Ang Output \n");
errorCode = i3dmgx2_AccelAndAngRate(portNum, &Record[0]);
if (errorCode < 0)
    printf("    Error Accel and AngRate - : %s\n",
           explainError(errorCode));
else{
    for (i=0; i<3; i++) {
        Accel_AngRecord.Accel[i] = FloatFromBytes
            (&Record[1 + i*4]);
        // extract float from byte array
        Accel_AngRecord.AngRt[i] = FloatFromBytes(
            &Record[13 + i*4]);
        // extract float from byte array
    }
    printf("\n\tAccel X\t\tAccel Y\t\tAccel Z\n");
    printf("    \t%f\t%f\t%f\n", Accel_AngRecord.Accel[0],
           Accel_AngRecord.Accel[1], Accel_AngRecord.Accel[2]);
    printf("\n\tAng X\t\tAng Y\t\tAng Z\n");
    printf("    \t%f\t%f\t%f\n", Accel_AngRecord.AngRt[0],
           Accel_AngRecord.AngRt[1], Accel_AngRecord.AngRt[2]);
}

```

```
        Accel_AngRecord.timer = convert2ulong(&Record[25]);
        printf("\n    Time Stamp: %u\n", Accel_AngRecord.timer);
    }

Exit:
    /*----- close device */
    if (portNum >= 0)
        i3dmgx2_closeDevice(portNum);

    /*----- wait for user to respond before exiting */
    printf("\nHit return to exit...\n");
    while (getchar() != EOF);
    return(0);
}
```

Code Sample 2

LabVIEW 7.1.1 Code Samples

The LabVIEW Code Samples were constructed with the National Instruments LabVIEW version 7.1.1 Interface Development Environment (IDE) using only objects and components native to the IDE. No third party components or objects were used.

The LabVIEW Code Samples are written directly from the Data Communications Protocol.

The code shown and explained below is reflective of the entire code sample offering.

The code sample contains a Front Panel as shown in **Figure 2**. The Front Panel allows selection of a serial port and baud rate. The Front Panel displays the data packet breakout (in this case, the 0xC2 packet). The Form has a red/green checksum indicator and an 'Error Output' box which displays messages to the user.

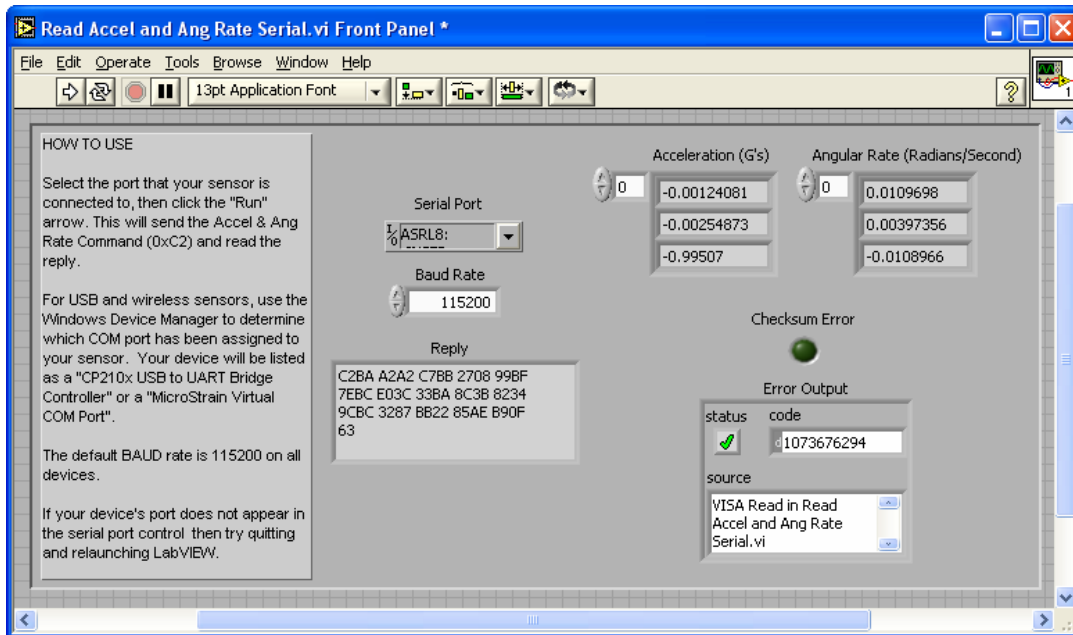


Figure 2

The Block Diagram is shown in **Figure 3**. The wiring path is linear. Baud rate and serial port are selected. The Run arrow is clicked. The serial port is configured and opened. The polling command is sent, the receive threshold is met, and the returned bytes are read from the serial port. The packet checksum is evaluated, the packet is parsed and scaled, and the results are displayed on the Front Panel.

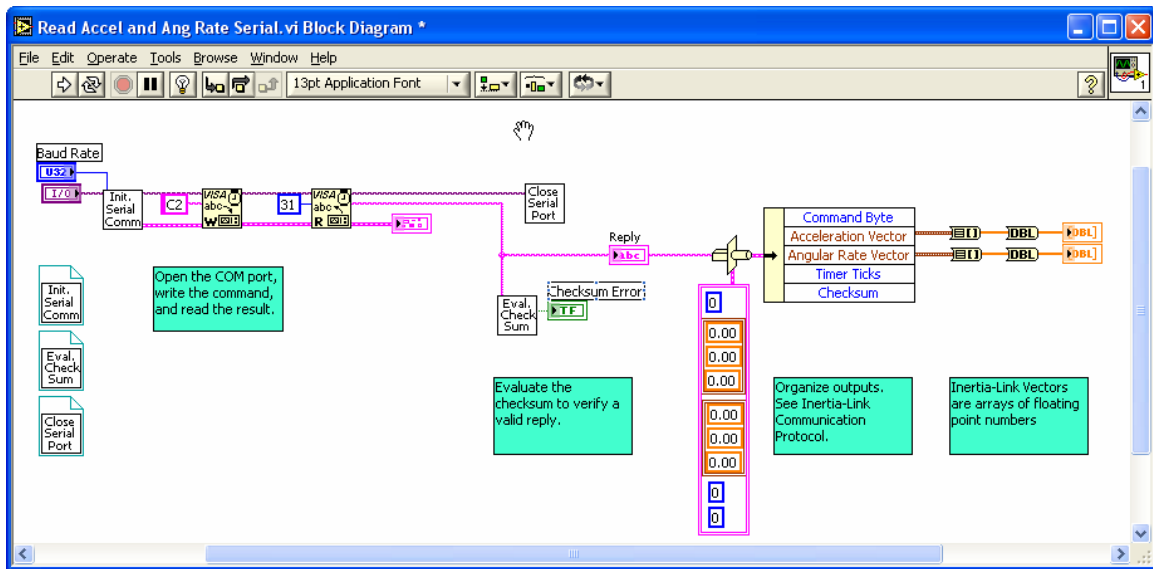


Figure 3

Visual Studio Visual Basic 2005 Code Samples

The Visual Basic 2005 Code Samples were constructed with the Microsoft Visual Basic 2005 version 8.0.50727.42 Interface Development Environment (IDE) and Microsoft .NET Framework version 2.0.50727 SP1 using only objects and components native to the IDE. No third party components or objects were used.

The Visual Basic 2005 Code Samples are written directly from the Data Communications Protocol.

The code shown and explained below is reflective of the entire code sample offering.

The code sample contains a Form as shown in **Figure 4**. The Form allows selection of a communication port. The Form displays the data packet breakout (in this case, the 0xC2 packet). The Form has a red/green checksum indicator and an 'Event' textbox which displays messages to the user.

Figure 4

Pseudo code

- On **Main_Load**, the combo box for the comm port selection is populated.
- The user clicks File and Sample on the menu bar.
- **SampleToolStripMenuItem_Click** configures the comm port and sends the first polling command.

- **SerialPort1_DataReceived** fires as the result of the receive threshold hitting 31 bytes, checks the data packet integrity and sends the packet to the **Checksum** function.
- **Checksum** tests the checksum and if good, parses and scales the data and returns it to **SerialPort1_DataReceived**. If **Checksum** fails, the packet is not parsed.
- Upon **Checksum** return, **SerialPort1_DataReceived** either displays the good packet, flashes the checksum indicator green and sends the next polling command, or flashes the checksum indicator red and sends the next polling command.
- The cycle continues until the user intervenes by clicking File and Sample to stop the process.
- The **ReportEvent** function is used to populate user messages in the Event textbox. **ExitToolStripMenuItem_Click** terminates the application.

Public Class Main

```
Private Sub ExitToolStripMenuItem_Click(ByVal sender As
System.Object, ByVal e As System.EventArgs) Handles
ExitToolStripMenuItem.Click
    'quit app
End
End Sub
```

```
Private Sub SampleToolStripMenuItem_Click(ByVal sender As
System.Object, ByVal e As System.EventArgs) Handles
SampleToolStripMenuItem.Click
    'set error handler
    On Error GoTo ErrorHandler

    'test for comm port selection
    If cmbCommPort.Text = "" Then
        'event
        Call ReportEvent("Comm Port not selected")
        'exit
        Exit Sub
    End If

    'determine action based on menu check
    If SampleToolStripMenuItem.Checked = False Then
        'event
        Call ReportEvent("Sampling started")
        'set menu check
        SampleToolStripMenuItem.Checked = True
        'disable combo
        cmbCommPort.Enabled = False
        'zero elapsed time counter
        ElapsedTime = 0
        'zero previous time counter
        PreviousTime = 0
        'set flag
        flgFirstPass = 0
        'set serial port number
```

```

        SerialPort1.PortName() = "COM" + Trim(cmbCommPort.Text)
        'set receive threshold
        SerialPort1.ReceivedBytesThreshold = 31
        'open serial port
        SerialPort1.Open()
        'populate byte array with polling command for 0xC2
        ReDim arrBytes(0 To 0)
        arrBytes(0) = 194
        'send command
        SerialPort1.Write(arrBytes, 0, 1)
    ElseIf SampleToolStripMenuItem.Checked = True Then
        'open serial port
        SerialPort1.Close()
        'event
        Call ReportEvent("Sampling stopped")
        'set menu check
        SampleToolStripMenuItem.Checked = False
        'set checksum indicator
        lblChecksum.ForeColor = Color.Red
        'enable combo
        cmbCommPort.Enabled = True
    End If

Exit Sub
ErrorHandler:
    'open serial port
    If SerialPort1.IsOpen = True Then SerialPort1.Close()
    'event
    Call ReportEvent("Sampling errored: " + Err.Description)
    'set menu check
    SampleToolStripMenuItem.Checked = False
    'set checksum indicator
    lblChecksum.ForeColor = Color.Red
    'enable combo
    cmbCommPort.Enabled = True
    'exit
Exit Sub
End Sub

Private Sub Main_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
    'populate comm port combo box
    Dim X As Integer
    For X = 1 To 32
        cmbCommPort.Items.Add(Trim(Str(X)))
    Next

    'Sets a value indicating whether to catch calls on the wrong
thread that access a control's Handle property
    txtHeader.CheckForIllegalCrossThreadCalls = False
    txtAccelX.CheckForIllegalCrossThreadCalls = False
    txtAccelY.CheckForIllegalCrossThreadCalls = False
    txtAccelZ.CheckForIllegalCrossThreadCalls = False
    txtAngRateX.CheckForIllegalCrossThreadCalls = False
    txtAngRateY.CheckForIllegalCrossThreadCalls = False
    txtAngRateZ.CheckForIllegalCrossThreadCalls = False
    txtTimerTick.CheckForIllegalCrossThreadCalls = False

```

```

txtElapsed.CheckForIllegalCrossThreadCalls = False
txtMyCheckSum.CheckForIllegalCrossThreadCalls = False
txtTheirCheckSum.CheckForIllegalCrossThreadCalls = False

```

```
End Sub
```

```

Private Function ReportEvent(ByVal MyEvent As String) As Boolean
    'add event to text
    If txtEvent.Text = "" Then
        txtEvent.Text = MyEvent
    Else
        txtEvent.Text = txtEvent.Text + Chr(13) + Chr(10) + MyEvent
    End If
End Function

```

```

Private Function CheckSum() As Boolean
    'set error handler
    On Error GoTo ErrorHandler

    '-----calc TheirCheckSum
    'dim
    Dim X As Double
    'zero variable
    TheirCheckSum = 0
    'calc TheirCheckSum
    For X = 0 To 28
        TheirCheckSum = TheirCheckSum + CDBl(arrBytes(X))
    Next X

    'handle CheckSum rollover
    TheirCheckSum = TheirCheckSum Mod 65536

    'calc MyCheckSum
    MyCheckSum = (CDBl(arrBytes(29)) * 256) + arrBytes(30)

    'compare checksums
    If MyCheckSum = TheirCheckSum Then
        'set checksum return
        CheckSum = True
        'calc datapoints
        MyHeader = arrBytes(0)
        'convert array of 32bit floating point values in IEEE-754
format
        arrTemp(0) = arrBytes(4)
        arrTemp(1) = arrBytes(3)
        arrTemp(2) = arrBytes(2)
        arrTemp(3) = arrBytes(1)
        AccelX = BitConverter.ToSingle(arrTemp, 0)
        'round
        AccelX = Math.Round(AccelX, 4)
        'convert array of 32bit floating point values in IEEE-754
format
        arrTemp(0) = arrBytes(8)
        arrTemp(1) = arrBytes(7)
        arrTemp(2) = arrBytes(6)
        arrTemp(3) = arrBytes(5)

```

```

        AccelY = BitConverter.ToSingle(arrTemp, 0)
        'round
        AccelY = Math.Round(AccelY, 4)
        'convert array of 32bit floating point values in IEEE-754
format
        arrTemp(0) = arrBytes(12)
        arrTemp(1) = arrBytes(11)
        arrTemp(2) = arrBytes(10)
        arrTemp(3) = arrBytes(9)
        AccelZ = BitConverter.ToSingle(arrTemp, 0)
        'round
        AccelZ = Math.Round(AccelZ, 4)
        'convert array of 32bit floating point values in IEEE-754
format
        arrTemp(0) = arrBytes(16)
        arrTemp(1) = arrBytes(15)
        arrTemp(2) = arrBytes(14)
        arrTemp(3) = arrBytes(13)
        AngRateX = BitConverter.ToSingle(arrTemp, 0)
        'round
        AngRateX = Math.Round(AngRateX, 4)
        'convert array of 32bit floating point values in IEEE-754
format
        arrTemp(0) = arrBytes(20)
        arrTemp(1) = arrBytes(19)
        arrTemp(2) = arrBytes(18)
        arrTemp(3) = arrBytes(17)
        AngRateY = BitConverter.ToSingle(arrTemp, 0)
        'round
        AngRateY = Math.Round(AngRateY, 4)
        'convert array of 32bit floating point values in IEEE-754
format
        arrTemp(0) = arrBytes(24)
        arrTemp(1) = arrBytes(23)
        arrTemp(2) = arrBytes(22)
        arrTemp(3) = arrBytes(21)
        AngRateZ = BitConverter.ToSingle(arrTemp, 0)
        'round
        AngRateZ = Math.Round(AngRateZ, 4)
        'calc timer tick and elapsed time
        TimerTick = TimerTickCalc(25)
Else
    'set checksum return
    CheckSum = False
End If

    'exit
    Exit Function
ErrorHandler:
    'set checksum return
    CheckSum = False
    'exit
    Exit Function
End Function

```

```

Private Sub SerialPort1_DataReceived(ByVal sender As System.Object,
ByVal e As System.IO.Ports.SerialDataReceivedEventArgs) Handles
SerialPort1.DataReceived
    'set error handler
    On Error GoTo ErrorHandler

    'set variable
    UpperLimitOfArray = SerialPort1.BytesToRead - 1
    'prep array
    ReDim arrBytes(0 To UpperLimitOfArray)
    'get comm buffer
    SerialPort1.Read(arrBytes, 0, SerialPort1.BytesToRead)

    'test for no byte return
    If UBound(arrBytes) = -1 Then
        'set checksum indicator
        lblCheckSum.ForeColor = Color.Red
        'populate byte array with polling command for 0xC2
        ReDim arrBytes(0 To 0)
        arrBytes(0) = 194
        'send command
        SerialPort1.Write(arrBytes, 0, 1)
        'exit
        Exit Sub
    End If

    'test 31 byte return and header
    If Not UBound(arrBytes) = 30 And arrBytes(0) = 194 Then
        'set checksum indicator
        lblCheckSum.ForeColor = Color.Red
        'populate byte array with polling command for 0xC2
        ReDim arrBytes(0 To 0)
        arrBytes(0) = 194
        'send command
        SerialPort1.Write(arrBytes, 0, 1)
        'exit
        Exit Sub
    End If

    'test checksum
    If CheckSum() = False Then
        'set checksum indicator
        lblCheckSum.ForeColor = Color.Red
        'populate byte array with polling command for 0xC2
        ReDim arrBytes(0 To 0)
        arrBytes(0) = 194
        'send command
        SerialPort1.Write(arrBytes, 0, 1)
        'exit
        Exit Sub
    End If

    'set checksum indicator
    lblCheckSum.ForeColor = Color.Green

    'display data

```



```
txtHeader.Text = MyHeader
txtAccelX.Text = AccelX
txtAccelY.Text = AccelY
txtAccelZ.Text = AccelZ
txtAngRateX.Text = AngRateX
txtAngRateY.Text = AngRateY
txtAngRateZ.Text = AngRateZ
txtTimerTick.Text = DisplayTimerTickCalc
txtElapsed.Text = ElapsedTime
txtMyChecksum.Text = MyChecksum
txtTheirChecksum.Text = TheirChecksum

'populate byte array with polling command for 0xC2
ReDim arrBytes(0 To 0)
arrBytes(0) = 194
'send command
SerialPort1.Write(arrBytes, 0, 1)

Exit Sub
ErrorHandler:

'close serial port
If SerialPort1.IsOpen = True Then SerialPort1.Close()

End Sub
```

```
End Class
```

Code Sample 3

Visual Basic 6.0 Code Samples

The Visual Basic 6.0 Code Samples were constructed with the Microsoft Visual Basic 6.0 Interface Development Environment (IDE), Service Pack 5, using only objects and components native to the IDE. No third party components or objects were used.

The Visual Basic 6.0 Code Samples are written directly from the Data Communications Protocol.

The code shown and explained below is reflective of the entire code sample offering.

The code sample contains a Form as shown in **Figure 5**. The Form allows selection of a communication port. The Form displays the data packet breakout (in this case, the 0xC2 packet). The Form has a red/green checksum indicator and an 'Event' textbox which displays messages to the user.

Figure 5

Pseudo code

- On **Form_Load**, the combo box for the comm port selection is populated.
- The user clicks File and Sample on the menu bar.
- **mnuSample_Click** configures the comm port and sends the first polling command.

- **MSComm1_OnComm** fires as the result of the receive threshold hitting 31 bytes, checks the data packet integrity and sends the packet to the **Checksum** function.
- **Checksum** tests the checksum and if good, parses and scales the data and returns it to **MSComm1_OnComm**. If **Checksum** fails, the packet is not parsed.
- Upon **Checksum** return, **MSComm1_OnComm** either displays the good packet, flashes the LED green and sends the next polling command, or flashes the LED red and sends the next polling command.
- The cycle continues until the user intervenes by clicking File and Sample to stop the process.
- The **ReportEvent** function is used to populate user messages in the Event textbox. **mnuExit_Click** terminates the application.

Private Sub Form_Load()

```
'populate comm port combo box
Dim X As Integer
For X = 1 To 16
    cmbCommPort.AddItem X
Next X
End Sub
```

Private Sub mnuExit_Click()

```
'quit app
End
End Sub
```

Private Sub mnuSample_Click()

```
'set error handler
On Error GoTo ErrorHandler

'test for comm port selection
If cmbCommPort.Text = "" Then
    'event
    Call ReportEvent("Comm Port not selected")
    'exit
    Exit Sub
End If

'determine action based on menu check
If mnuSample.Checked = False Then
    'event
    Call ReportEvent("Sampling started")
    'set menu check
    mnuSample.Checked = True
    'disable combo
    cmbCommPort.Enabled = False
```

```
'zero elapsed time counter
ElapsedTime = 0
'zero previous time holder
PreviousTime = 0
'set flag
flgFirstPass = False
'set comm port
MSComm1.CommPort = Val(cmbCommPort.Text)
'set receive threshold to 31 bytes
MSComm1.RThreshold = 31
'open comm port
MSComm1.PortOpen = True
'populate byte array with polling command for 0xC2
'decimal equivalent of 0xC2 = 194
arrBytes = ChrB(194)
'send polling command
MSComm1.Output = arrBytes
ElseIf mnuSample.Checked = True Then
    'event
    Call ReportEvent("Sampling stopped")
    'set menu check
    mnuSample.Checked = False
    'close comm port
    MSComm1.PortOpen = False
    'enable combo
    cmbCommPort.Enabled = True
    'set checksum indicator
    shpLED.FillColor = vbRed
End If

Exit Sub
ErrorHandler:
    'set menu check
    mnuSample.Checked = False
    'close comm port
    If MSComm1.PortOpen = True Then MSComm1.PortOpen = False
    'enable combo
    cmbCommPort.Enabled = True
    'event
    Call ReportEvent("Sampling errored: " + Err.Description)
    'set checksum indicator
    shpLED.FillColor = vbRed
    'exit
    Exit Sub
End Sub
```

Private Function ReportEvent(MyEvent As String)

```
'add event to text
If txtEvent.Text = "" Then
    txtEvent.Text = MyEvent
Else
    txtEvent.Text = txtEvent.Text + Chr(13) + Chr(10) + MyEvent
End If
End Function
```

Private Sub MSComm1_OnComm()

```
'get all bytes in comm buffer
arrBytes = MSComm1.Input

'test 31 byte return and header
If Not UBound(arrBytes) = 30 Or Not arrBytes(0) = 194 Then
    'set checksum indicator
    shpLED.FillColor = vbRed
    'populate byte array with polling command for 0xC2
    arrBytes = ChrB(194)
    'send polling command
    MSComm1.Output = arrBytes
    'exit sub
    Exit Sub
End If

'test checksum
If CheckSum = False Then
    'set checksum indicator
    shpLED.FillColor = vbRed
    'populate byte array with polling command for 0xC2
    arrBytes = ChrB(194)
    'send polling command
    MSComm1.Output = arrBytes
    'exit sub
    Exit Sub
End If

'set checksum indicator
shpLED.FillColor = vbGreen

'display data
txtHeader.Text = MyHeader
txtAccelX.Text = Round(AccelX, 4)
txtAccelY.Text = Round(AccelY, 4)
```

```
txtAccelZ.Text = Round(AccelZ, 4)
txtAngRateX.Text = Round(AngRateX, 4)
txtAngRateY.Text = Round(AngRateY, 4)
txtAngRateZ.Text = Round(AngRateZ, 4)
txtTimerTick.Text = DisplayTimerTickCalc
txtElapsed.Text = ElapsedTime
txtMyChecksum.Text = MyChecksum
txtTheirChecksum.Text = TheirChecksum

'populate byte array with polling command for 0xC2
arrBytes = ChrB(194)
'send polling command
MSComm1.Output = arrBytes
```

End Sub

Private Function CheckSum() As Boolean

```
'set error handler
On Error GoTo ErrorHandler

'-----calc TheirChecksum
'dim
Dim X As Double
'zero variable
TheirChecksum = 0
'calc TheirChecksum
For X = 0 To 28
    TheirChecksum = TheirChecksum + CDBl(arrBytes(X))
Next X

'handle CheckSum rollover
TheirChecksum = TheirChecksum Mod 65536

'calc MyChecksum
MyChecksum = (CDBl(arrBytes(29)) * 256) + arrBytes(30)

'compare checksums
If MyChecksum = TheirChecksum Then
    'set checksum return
    CheckSum = True
    'calc datapoints
    MyHeader = arrBytes(0)
    AccelX = FloatConversion(1)
    AccelY = FloatConversion(5)
    AccelZ = FloatConversion(9)
    AngRateX = FloatConversion(13)
```

```
    AngRateY = FloatConversion(17)
    AngRateZ = FloatConversion(21)
    TimerTick = TimerTickCalc(25)
Else
    'set checksum return
    CheckSum = False
End If

'exit
Exit Function
ErrorHandler:
    'set checksum return
    CheckSum = False
    'exit
    Exit Function
End Function
```

Code Sample 4

CP210x USB to UART Bridge Controller

The MicroStrain Inertia-Link[®] and 3DM-GX2[®] orientation sensors are available with many communication interfaces as previously stated. In particular the USB interface has a special driver installed in the operating system.

The USB interface, from a physical standpoint, resides in a communication cable connecting a Micro-D connector to the sensor and a standard USB connector to the host. The USB communication on the sensor is provided via a Silicon Laboratories CP210x USB to UART Bridge chip. The CP210x is a single-chip USB to UART bridge that converts data traffic between USB and UART formats. The chip includes a complete USB 2.0 full-speed function controller, bridge control logic and a UART interface with transmit/receive buffers and modem handshake signals. Specifications for the chip may be found at: <https://www.silabs.com/products/interface/usbtouart/Pages/default.aspx>

This physical architecture is supported by a Silicon Laboratories CP210x USB to UART Bridge Virtual COM Port (VCP) driver installed on the host. This driver is required for device operation as a Virtual COM Port to facilitate host communication. The driver is normally installed during installation of MicroStrain's data acquisition and display software. The driver may also be downloaded from Silicon Labs at: <https://www.silabs.com/products/mcu/Pages/USBtoUARTBridgeVCPDrivers.aspx>

With the installation of this driver, the developer will find that communication between host and sensor will be, for all intents and purposes, typical serial communication. The various coding languages will interact as if a standard serial port existed.

To determine if the driver is properly installed, connect the orientation sensor to the host and follow these steps:

- Click Start on the Windows desktop.
- Click Control Panel.
- Click System icon.
- System Properties window appears.
- Click Hardware tab.
- Click Device Manager.
- Device Manager window appears.
- Locate Ports(Com&LPT) in the tree.
- Double-click the + to further open the Ports tree.
- Do you see a device named 'CP210x USB to UART Bridge Controller (ComX)'? (with X representing a comm port number).
- If so, the driver is installed.
- If not, the orientation sensor is malfunctioning or the driver is not installed.

Suggested Debugging Tools

We have found that software port *sniffers* are incredibly valuable when building applications with serial communication. Here are some of our favorites; we highly recommend their use.

LookRS232

<http://www.lookrs232.com/>

- Look RS232 is a tool for debugging computer connection with peripheral devices using COM port, such as modem, mini-ATS, projector etc. Its easy to use interface guarantees an easy work with Com-port.
- Look RS232 can send data through COM port, receive data from an outer device through COM port, it has port indication as well.
- Look RS232 supports connection at the standard speed of 110...115200 kbit/s, it supports commands of COM port program control.
- Look RS232 supports data input in ASCII, BIN, HEX, OCT formats, keeps log of the sent and received data and commands in ASCII, BIN, HEX, OCT formats, it has an option of time link (relating to COM port activation).
- Look RS232 works with system initiated COM ports regardless of whether they are installed on motherboard or on supplementary in/output boards, e.g. VS-Com (Roadrunner) PCI-800H 8-port PCI.

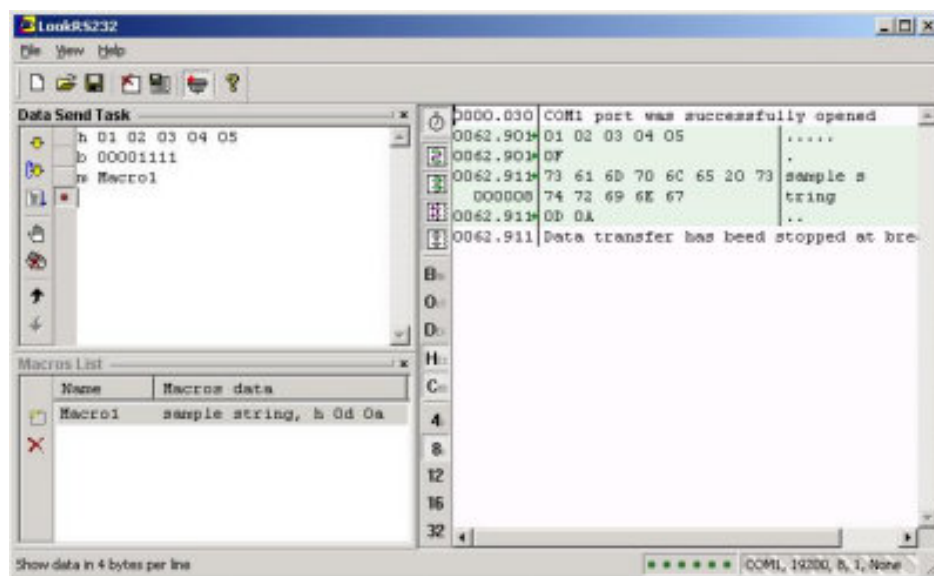


Figure 6

Serial Port Monitor

<http://www.serial-port-monitor.com/index.html>

Free Serial Port Monitor allows you to intercept, display and analyze all data exchanged between the Windows application and the serial device. It can be successfully used in application development, device driver or serial hardware development and offers the powerful platform for effective coding, testing and optimization.

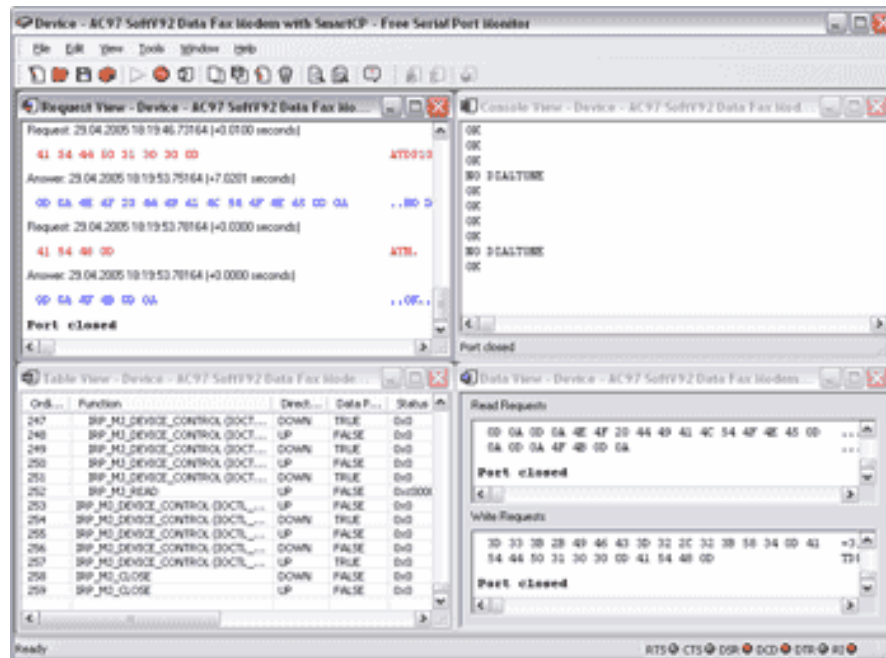


Figure 7

Comm Operator

<http://www.serialporttool.com/CommOpV2Info.htm>

- Comm Operator is a powerful tool for serial port communication. It can act as peripheral device emulator related RS232 COM port. It can also act as a debug tool for device's test and development.
- The built-in event driven auto send mechanics makes Comm Operator a full functioned device simulator with just a few mouse clicks. By this way, RS232 related software can be developed offsite, no need for device, only the rule script is enough.
- Comm Operator can send data through COM port, receive data from an outer device through COM port. The speed it supports range from 110 to 921600. It can detect all COM ports automatically regardless of whether they are real COM ports on motherboard or on supplementary in/output boards, or virtual COM ports created by special drivers.
- Comm Operator supports data input in ASCII, HEX as well as Decimal formats. It keeps log of the sent and received data in ASCII, HEX as well as Decimal formats. It also supports SYMBOL with which the invisible ASCII code became meaningful immediately. All setting can be saved to file and loaded later.

Support

Overview

- MicroStrain is committed to providing timely, knowledgeable, world-class support to its customers.
- We are open 24 X 7 through our web portal.
- We make every attempt to respond to your email the same business day.
- We are always available by telephone during business hours.
- We provide in-depth FAQs, manuals, quick start guides and technical notes.
- Firmware and software upgrades are made available on-line as they become available.
- Code samples in several languages are posted to aid your development.
- We support our customers as we would want to be supported.

Web

Our home page is at URL: www.microstrain.com

Our support page is at URL: http://www.microstrain.com/support_overview.aspx

Email

MicroStrain's Support Engineers make every attempt to respond to emails requesting product support within the same business day. The more detail you can provide, the quicker we will be able to understand your issues and find solutions. Data files, pictures, screen grabs, etc. are all very helpful in generating a well-thought-out solution.

Please email us at: support@microstrain.com

Telephone

MicroStrain's Support Engineers are available by phone Monday through Friday 9:00AM to 5:00PM local time. When calling MicroStrain, indicate to the receptionist that you are calling for product support and you will be promptly routed to a Support Engineer. Please have your equipment ready to test. Every attempt will be made to solve issues while you are on the line.

1.800.449.DVRT(3878) Toll Free in US
1.802.862.6629 telephone
1.802.863.4093 fax

Local time = GMT -05:00 (Eastern Time US & Canada)

SKYPE

MicroStrain's Support Engineers are available by SKYPE Monday through Friday 9:00AM to 5:00PM local. SKYPE name: **microstrain.orientation.support**