



ASCENSION TECHNOLOGY CORPORATION

---

*Making Minimally Invasive Possible*

# 3D Guidance medSAFE<sup>TM</sup>

## Installation and Operation Guide

# 3D Guidance **medSAFE**

## Installation and Operation Guide

---

### REVISION NUMBER:

940034 Rev F6

2/12/10

### TRADEMARKS

Microsoft Windows XP® and Windows Vista®  
are registered trademarks of Microsoft Corporation.  
All other products mentioned in this guide are trademarks  
or registered trademarks of their respective companies.

3DGuidance medSAFE is a trademark of Ascension Technology Corporation

© 2010 Ascension Technology Corporation. All rights reserved.

P.O. Box 527

Burlington, VT USA 05402

Phone (802) 893-6657 • Fax (802) 893-6659

# Table of Contents

<b>Introduction.....</b>	<b>i</b>
<b>About This Guide.....</b>	<b>i</b>
How this Guide is Organized .....	i
Guide Conventions .....	iii
<b>Getting Assistance.....</b>	<b>iii</b>
<b>Chapter 1: Preparing for Setup and Safe Performance.....</b>	<b>1</b>
System Requirements .....	1
Intended Use Statement .....	1
Software Requirements .....	1
Hardware Requirements.....	2
Unpacking the Tracker .....	2
Safe Performance & Handling Precautions .....	6
Environmental Conditions.....	8
Temperature .....	8
Humidity .....	8
<b>Chapter 2: Setup and Checkout .....</b>	<b>9</b>
Install the Software.....	9
Cable Connections.....	12
Installing the Driver.....	14
System Checkout.....	15
Running the USB Demo Software .....	15
Running the RS232 Demo Utility.....	17
<b>Chapter 3: Configuration and Basic Operation.....</b>	<b>20</b>
Default Configuration.....	20
Configurable Power-up Settings: .....	21
Default Reference Frames.....	24
Mid and Short-Range Transmitter Reference Frame .....	24
Flat Transmitters.....	25
Changing Your Settings.....	25
Mounting the Hardware.....	26
Mid-Range Transmitter Mounting and Location .....	26
Short-Range Transmitter Mounting and Location .....	27
Sensor Mounting.....	27
Locating Your Electronics Unit .....	28
Flat Transmitter Mounting .....	28
Pre-amplifier Mounting .....	28
Rear Panel Connectors .....	28
Power .....	28
RS232 .....	28
USB.....	29
SYNC.....	29

---

SWITCH .....	29
Basic Operation .....	29
Dipole Transmitter .....	29
Non-Dipole Transmitter .....	30
6DOF Sensor .....	30
5DOF Sensor .....	30
Pre-Amplifier .....	30
Electronics .....	30
Measurement Cycle .....	31
Calibration .....	31
Performance Factors .....	32
Electromagnetic and Other Interference in Tracking .....	32
Excessive Electrical Noise .....	32
Causes of Noise .....	32
Reducing Noise .....	33
Magnetic Distortion .....	33
Causes of Distortion .....	33
Reducing Distortion .....	34
Metal Detection .....	34
Tracker as the Cause of Interference .....	34
Factors in Tracker Accuracy .....	35
Warm-up .....	35
Default Measurement Rate .....	35
Equipment Alteration .....	35
Power Grid Magnetic Interference .....	35
Performance Motion Box .....	36
<b>Chapter 4: Software Operation: Tools for Successful Tracking .....</b>	<b>39</b>
Software Overview .....	39
3DGuidance API .....	39
Sample Programs .....	40
Ascension RS232 Interface .....	41
Ascension RS232 Driver .....	41
Direct Communication: Ascension's RS232 Protocol .....	41
RS232 Sample Program .....	42
<b>Chapter 5: 3DGuidance API Reference .....</b>	<b>43</b>
Using 3D Guidance medSAFE .....	43
Quick Reference .....	44
SYSTEM .....	44
SENSOR .....	45
BOARD .....	49
TRANSMITTER .....	49
System Initialization .....	50
ATC3DGm.ini File .....	51
System Setup .....	52
Sensor Setup .....	54
Transmitter Setup .....	56
Acquiring Tracking Data .....	57
Error Handling .....	58
3DGuidance API .....	59
3DGuidance API Functions .....	60

---

InitializeBIRDSystem.....	61
GetBIRDSystemConfiguration.....	63
GetTransmitterConfiguration.....	64
GetSensorConfiguration.....	65
GetBoardConfiguration.....	66
GetSystemParameter.....	67
GetSensorParameter.....	68
GetTransmitterParameter.....	70
GetBoardParameter.....	72
SetSystemParameter.....	74
SetSensorParameter.....	76
SetTransmitterParameter.....	78
SetBoardParameter.....	80
GetAsynchronousRecord.....	82
GetSynchronousRecord.....	84
GetBIRDError.....	87
GetErrorText.....	88
GetSensorStatus.....	90
GetTransmitterStatus.....	92
GetBoardStatus.....	94
GetSystemStatus.....	96
SaveSystemConfiguration.....	98
RestoreSystemConfiguration.....	100
CloseBIRDSystem.....	102
3D Guidance API Structures.....	103
SYSTEM_CONFIGURATION.....	104
TRANSMITTER_CONFIGURATION.....	106
SENSOR_CONFIGURATION.....	107
BOARD_CONFIGURATION.....	108
ADAPTIVE_PARAMETERS.....	110
QUALITY_PARAMETERS.....	111
VPD_COMMAND_PARAMETER.....	112
BOARD_REVISIONS.....	113
SHORT_POSITION_RECORD.....	115
SHORT_ANGLES_RECORD.....	117
SHORT_MATRIX_RECORD.....	119
SHORT_QUATERNIONS_RECORD.....	121
SHORT_POSITION_ANGLES_RECORD.....	122
SHORT_POSITION_MATRIX_RECORD.....	123
SHORT_POSITION_QUATERNION_RECORD.....	124
DOUBLE_POSITION_RECORD.....	125
DOUBLE_ANGLES_RECORD.....	127
DOUBLE_MATRIX_RECORD.....	128
DOUBLE_QUATERNIONS_RECORD.....	130
DOUBLE_POSITION_ANGLES_RECORD.....	131
DOUBLE_POSITION_MATRIX_RECORD.....	132
DOUBLE_POSITION_QUATERNION_RECORD.....	133
DOUBLE_POSITION_TIME_STAMP_RECORD.....	134
DOUBLE_ANGLES_TIME_STAMP_RECORD.....	135
DOUBLE_MATRIX_TIME_STAMP_RECORD.....	136
DOUBLE_QUATERNIONS_TIME_STAMP_RECORD.....	137
DOUBLE_POSITION_ANGLES_TIME_STAMP_RECORD.....	138
DOUBLE_POSITION_MATRIX_TIME_STAMP_RECORD.....	140

---

DOUBLE_POSITION_QUATERNION_TIME_STAMP_RECORD .....	141
DOUBLE_POSITION_TIME_Q_RECORD .....	142
DOUBLE_ANGLES_TIME_Q_RECORD.....	144
DOUBLE_MATRIX_TIME_Q_RECORD.....	145
DOUBLE_QUATERNIONS_TIME_Q_RECORD .....	146
DOUBLE_POSITION_ANGLES_TIME_Q_RECORD.....	147
DOUBLE_POSITION_MATRIX_TIME_Q_RECORD.....	149
DOUBLE_POSITION_QUATERNION_TIME_Q_RECORD .....	151
SHORT_ALL_RECORD .....	153
DOUBLE_ALL_RECORD .....	155
DOUBLE_ALL_TIME_STAMP_RECORD .....	157
DOUBLE_ALL_TIME_STAMP_Q_RECORD .....	159
DOUBLE_ALL_TIME_STAMP_Q_RAW_RECORD .....	161
DOUBLE_POSITION_ANGLES_TIME_Q_BUTTON_RECORD .....	163
DOUBLE_POSITION_MATRIX_TIME_Q_BUTTON_RECORD.....	165
DOUBLE_POSITION_QUATERNION_TIME_Q_BUTTON_RECORD .....	167
3D Guidance API Enumeration Types.....	169
BIRD_ERROR_CODES .....	170
SENSOR_PARAMETER_TYPE.....	175
MESSAGE_TYPE.....	181
TRANSMITTER_PARAMETER_TYPE .....	182
BOARD_PARAMETER_TYPE .....	184
SYSTEM_PARAMETER_TYPE .....	185
HEMISPHERE_TYPE .....	187
AGC_MODE_TYPE .....	188
DATA_FORMAT_TYPE.....	189
BOARD_TYPES .....	192
DEVICE_TYPES .....	193
3D Guidance API Status/Error Bit Definitions .....	195
ERRORCODE .....	196
DEVICE_STATUS .....	197
3D Guidance Initialization Files .....	198
3D Guidance Initialization File Format Reference .....	198
[System].....	199
[Sensorx].....	201
[Transmitterx] .....	203

## **Chapter 6: Ascension RS232 Interface Reference ..... 204**

RS232 Signal Description .....	204
Using the 'reset on CTS' feature.....	205
RS232 Commands .....	205
Command Summary .....	206
Command Utilization.....	208
Response Format.....	208
Position/Orientation Data Format .....	209
RS232 Command Reference .....	211
ANGLES.....	212
ANGLE ALIGN .....	213
BORESIGHT .....	214
BORESIGHT REMOVE .....	215
BUTTON MODE .....	216
BUTTON READ .....	217

---

CHANGE VALUE .....	218
EXAMINE VALUE.....	218
TRACKER STATUS .....	221
SOFTWARE REVISION NUMBER .....	221
TRACKER COMPUTER CRYSTAL SPEED.....	222
POSITION SCALING.....	222
FILTER ON/OFF STATUS .....	222
MEASUREMENT RATE .....	223
DISABLE/ENABLE DATA READY OUTPUT .....	224
SET DATA READY CHARACTER .....	224
ERROR CODE.....	224
DC FILTER TABLE $V_m$ .....	224
DC FILTER CONSTANT TABLE ALPHA_MAX.....	225
SUDDEN OUTPUT CHANGE LOCK.....	226
SYSTEM MODEL IDENTIFICATION .....	226
XYZ REFERENCE FRAME .....	227
FILTER LINE FREQUENCY.....	227
HEMISPHERE.....	227
ANGLE ALIGN .....	228
REFERENCE FRAME.....	228
TRACKER SERIAL NUMBER.....	228
SENSOR SERIAL NUMBER.....	228
TRANSMITTER SERIAL NUMBER .....	228
METAL .....	229
REPORT RATE .....	229
GROUP MODE.....	229
SYSTEM STATUS .....	230
AUTOCONFIG .....	230
SENSOR OFFSET .....	231
BOOT LOADER FIRMWARE REVISION .....	232
MDSP FIRMWARE REVISION .....	232
NON DIPOLE POSERVER FIRMWARE REVISION .....	232
FIVE DOF FIRMWARE REVISION .....	232
SIX DOF FIRMWARE REVISION.....	232
DIPOLE POSERVER FIRMWARE REVISION.....	232
HEMISPHERE .....	233
MATRIX.....	235
METAL.....	237
OFFSET.....	239
POINT.....	240
POSITION .....	241
POSITION/ANGLES .....	242
POSITION/MATRIX .....	243
POSITION/QUATERNION .....	244
QUATERNION .....	245
READ_VPD.....	246
REFERENCE FRAME.....	247
REPORT RATE.....	248
RESET .....	249
RS232 TO FBB.....	250
RUN.....	251
SLEEP.....	252
STREAM .....	253

---

STREAM STOP .....	254
WRITE_VPD .....	255
Error Reporting .....	256
Error Code Listing .....	257
<b>Chapter 7: Troubleshooting .....</b>	<b>259</b>
Error Codes .....	260
<b>Chapter 8: Maintenance, Repair and Disposal .....</b>	<b>261</b>
User Maintenance .....	261
Maintenance Prior to Each Use .....	261
Periodic Maintenance (As needed) .....	261
Cleaning and Disinfecting .....	262
Sensor Sterilization .....	262
Broad Guidelines When Considering the Cidex (Glutaraldehyde) Process .....	262
Broad Guidelines When Considering the EtO Sterilization Process .....	263
Software and Firmware Updates .....	263
Repair .....	263
Warranty .....	264
Disposal .....	264
<b>Chapter 9: Regulatory Information and Specifications .....</b>	<b>265</b>
EC Declaration of Conformity .....	266
FCC Compliance Statement .....	267
Product Specifications .....	268
Performance .....	268
Physical .....	269
<b>Appendix I: medSAFE Utility .....</b>	<b>271</b>
Running medSAFE Utility .....	271
Setup .....	271
Run the Utility .....	271
<b>Appendix II: Application Notes .....</b>	<b>274</b>
Computing Stylus Tip Coordinates .....	274

---



# Introduction

Congratulations on your purchase of our 3D Guidance medSAFE tracking device. We are proud of the quality of all our tracking products and want to meet your expectations. Please contact us immediately if you encounter any problems with its use.

medSAFE is a medically compliant guidance and localization device for instruments and tools. It employs pulsed DC magnetic field generation and sensing to safely and accurately track the position and orientation (five and six degrees-of-freedom) of a variety of sensors. Multiple field transmitter and sensors options are available to provide the best performance for various medical procedures. Tracking locations are determined in real time and are instantly reported to your host computer via a RS232 or USB interface.

This guide will help set up and install the medSAFE hardware and software and configure the system for optimal tracking.

## About This Guide

This *Installation and Operation Guide* contains all the information you need to install and run the tracker. It contains simple steps to operate, communicate with, and test your tracker. You'll also find many helpful tools to configure the tracker for your particular application.

## How this Guide is Organized

Information is presented in eight chapters.

### Chapter 1: Preparing for Setup and Safe Performance

- States the intended use.
- Lists system software and hardware requirements.
- Outlines the components of your system.
- Details safe performance and handling precautions.

### Chapter 2: Quick Start: Setup and Checkout

- Directs you on connecting your system components.

- Enables a quick checkout running our demonstration software.

### Chapter 3: Configuration and Basic Operation

- Outlines default configuration parameters and reference frames.
- Provides component mounting information.
- Discusses basic principles of tracker operation.
- Describes factors that affect tracker performance to include electromagnetic and other interference.

### Chapter 4: Software Operation – Tools for Successful Tracking

- Overviews the system software.
- Provides a few sample programs.

### Chapter 5: 3DGuidance API Reference

- Gives you an overview of the 3D Guidance API.
- Describes sample programs included on your CD-ROM that illustrate the tracker's communication structure.
- Details the 3D Guidance Application Programming Interface (API) for communicating with the tracker using USB.

### Chapter 6: Ascension's RS232 Interface Reference

- Describes an alternate method for communicating directly with the tracker follows our popular and often-used **Ascension RS232 Interface** protocol

### Chapter 7: Troubleshooting

- Lists common setup problems and solutions.

### Chapter 8: Maintenance, Repair and Disposal

- Offers user maintenance prior to each use and other period maintenance.
- Addresses cleaning and disinfecting methods.
- Lists replacement part numbers.
- Provides details of the warranty.

- Identifies disposal guidance.

## Chapter 9: Regulatory Information and Product Specifications

- Lists applicable standards, symbols, specifications, and certifications for the medSAFE.

## Guide Conventions

This Guide uses a number of conventions to explain procedures and present information clearly.

**Notes:** Notes describe important hardware or software features.



**Note:** This call-out explains important information about the features of your tracker.

**Tips:** Tips will help you get the best performance out of your tracker.



**Tips:** This call-out provides advice for maximizing the performance of your tracker.

**Caution!** These messages alert you to important operating instructions. If unsure about an action you are about to take, contact our Technical Support Group.



**CAUTION!** This call-out points out steps that should be avoided to prevent damage to your tracker.

**Names of files, directories and programs:** These are italicized (for example, *ATC3DGm.lib*)

## Getting Assistance

If you are experiencing a problem with the installation, setup, or operation of your tracker, we suggest your first consult the [Troubleshooting Table](#) in Chapter 7. It describes potential setup problems and how to resolve them. If you continue to experience problems, contact us as follows:

**World Wide Web:** <http://www.ascension-tech.com/technical/>

**E-mail:** <mailto:support@ascension-tech.com>

**Telephone:** (802) 893-6657 (U.S. Eastern Standard Time: 9AM – 5PM)

**Fax:** (802) 893-6659

# Chapter 1: Preparing for Setup and Safe Performance

*This chapter describes everything you will need to setup your 3D Guidance medSAFE tracker*

## System Requirements

### Intended Use Statement

medSAFE is designed to allow real-time tracking or measurement of an object's position and orientation in free space.

### Software Requirements

The following Windows- based utilities are included on the 3DGuidance CD-ROM:

1. *medSAFE Demo* - a demonstration utility that communicates using USB and the 3DGuidance USB API.
2. *winBIRD* - a demonstration utility that communicates via RS232 using the 3D Guidance RS232 interface.
3. *medSAFE Utility* - a utility for changing default configuration parameters of your tracker

These utilities and communication with the tracker via the Windows APIs require:

Windows XP or Windows Vista



**Note:** A Windows OS is required for running the utilities or communicating with the tracker using one of the Windows APIs. Communicate directly with the 3DGuidance using our 3D Guidance RS232 Interface. See Chapter 6

Note: *medSAFEDemo* was formerly known as *PCICubes* and *medSAFEUtility* was formerly known as the *medical tracker utility*.

## Hardware Requirements

**USB port:** medSAFE reports data serially to your host computer through a USB cable. An unused USB 2.0 port is required for the tracker.

**COM port:** medSAFE reports data serially to your host computer through a RS232 cable. An unused COM port is required for the tracker.

**Power:** The trackers' power supply will operate from 100 to 240V AC, at frequencies of 50-60 Hz using up to 60 VA of power.

**CD-ROM drive:** You need this only for accessing utilities and drivers, described above. These utilities may also be conveniently downloaded from the Internet by visiting the FTP site on our website: <ftp://ftp.ascension-tech.com/>

## Unpacking the Tracker

Your medSAFE tracker is packaged in one shipping box. Inside, you will find the following items needed to set-up and operate your tracker:

- 3D Guidance medSAFE Electronics Unit:



- Sensor Options:

Model 800



Model 180, 130, 90 Sensors



- Pre-amplifier Options:

Model 800



Model 180, 130, 90



- Transmitter Options:

Mid-Range Transmitter:



Short-Range Transmitter



4-Axis Flat Transmitter or 9-Axis Flat Transmitter



**Note:**

The transmitters are specifically calibrated for your 3D Guidance medSAFE unit.

- Cables:

1- AC Power cord



RS232 Cable



USB Cable



- 3D Guidance medSAFE CD-ROM:



If you notice any missing components or the shipment is damaged, please contact Ascension Technical Support.



## Safe Performance & Handling Precautions

Ascension sensors and transmitters, along with their attached cables and connectors, are sensitive electronic components. To obtain consistent performance and maintain your warranty, treat them carefully.

- Read this Guide.
- Handle the section of cable near the sensor head or transmitter housing with care. Repeated bending of the cable near the sensor head or transmitter housing is the most common cause of tracker failure.
- When power is applied or the system is running, do not touch exposed electronic components. *Contact with exposed components could cause injury.*
- If you insert the sensor or transmitter in a mounting bracket or holder, be careful when you remove them. **Do not yank or pull on the cable.**
- Sensors and transmitters can be damaged if you carry, throw, or swing them by their cables or if you let them drop against hard surfaces.
- Sensor and transmitter cables have been precisely bundled, shielded, and calibrated to minimize noise and ensure accurate performance. Do not tamper with them. If you attempt to add your own extension cables or connectors, you may well compromise performance and, of course, void both regulatory approvals and your warranty.
- To clean your equipment, use a cloth to wipe components with a general purpose cleaning solution such as soap and water, isopropyl alcohol, etc. Do not immerse the transmitter, sensors, or cables in any liquids.
- Keep the transmitter, sensors, and cables away from sources of heat.
- If mounting a mid-range or short-range transmitter inside an enclosure, be sure to provide adequate ventilation. Mid-range and short-range transmitters should not be mounted beneath mattresses, pillows, or any other object that will curtail air circulation in the immediate vicinity. Flat transmitters are excluded from this restriction.
- Never power up the system or place the transmitter in an explosive atmosphere.
- Tracker components may be subject to interference from or may interfere with other electrical equipment in your environment. Be sure to identify sources of interference in



your particular environment before using this tracker. See [Electromagnetic and Other Interference in Tracking](#).

- Do not hang the mid-range transmitter upside down by its rear mounting holes. They are not designed to hold the full weight of the transmitter. Such a set-up could cause damage to the transmitter, nearby equipment, and even human injury.
- Care should be taken to avoid spillage on the electronics unit and components.
- Do not overly flex or twist the sensor cable.
- Do not allow the sensor or any cables to be crushed or subjected to undue strain and stress. The connectors can become warped if stepped on; the internal wires in the sensor cable can break or become weakened if pinched; and the sensor head may be damaged if trapped under heavy weights.
- Do not drop or smack the sensor head against a hard surface. Such impacts can produce internal damage and adversely affect tracking accuracy.
- Be sure to implement a strain relief if you embed a sensor and its cable in an instrument or tool. The point where the sensor cable exits from your tool needs protection. Your sensor will last a long time if you take steps now to distribute forces over an extended region of the cable.
- To extend tracker life, be sure to shut down the transmitter when not in use. You can do this in several ways:
  - a. Select “No Transmitter” by setting the System parameter: [SELECT\\_TRANSMITTER](#) value to -1 in the 3D Guidance API.
  - b. Recycle the power on the electronic unit (if independent access is available to the unit’s power).
  - c. Disconnect and reconnect the USB cable.
  - d. Hold the system in reset using the [CTS Reset feature](#) or [SLEEP](#) command, if using the RS232 mode.

## Environmental Conditions

The medSAFE must be used and maintained in the following ranges only:

### Temperature

The tracker operates within specification when the ambient air temperature is between 15 degree C and 35 degree C. The medSAFE can be packaged and shipped in environments with an ambient air temperature between -40 degree C and 70 degree C without degradation of its components.

### Humidity

The tracker operates in non-condensing environments with relative humidity between 10% and 90%. It is capable of being packaged and shipped in environments with a relative humidity between 5% and 95%.


# 2

## Chapter 2: Setup and Checkout

*This chapter demonstrates how to install the software and connect your components so that you can quickly checkout your device and begin tracking.*

### Install the Software

The medSAFE CD-ROM has an installer that will copy all required software (utilities, sample code, API, etc) and user documentation onto your host PC. Note that you must have administrator privileges for the installer to run.

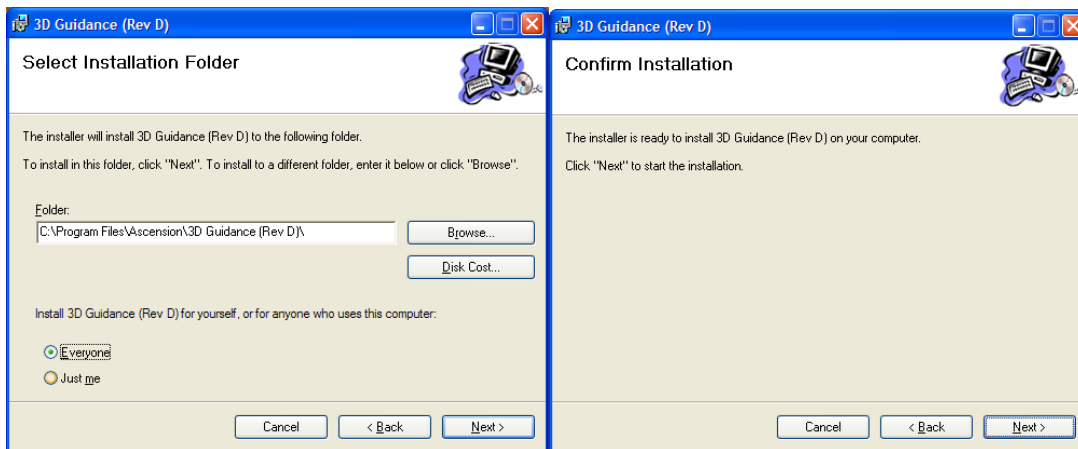
 **Note:**  
Install the software **BEFORE** connecting the tracker.

**\*\*INSTALL THE SOFTWARE BEFORE CONNECTING THE TRACKER\*\***

1. To begin the installer, place the CD-ROM in the tray, and close the drive door. NOTE: If the installer does not start (drive not set to 'Autoplay'), then browse to the CD-ROM folder and run the 'setup.exe' file.



2. Follow the prompts in the Setup Wizard, confirming the target folder and selecting the user access (install for everyone or just current user).



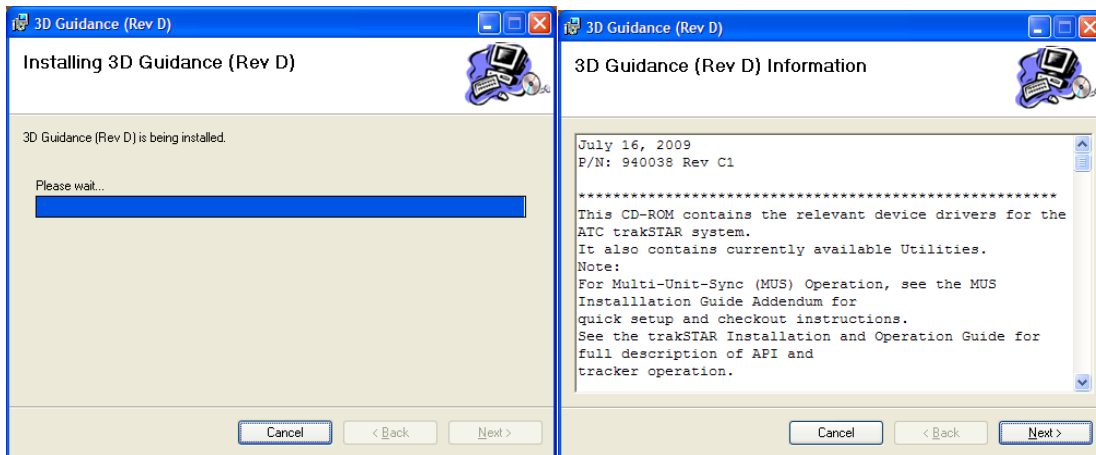
3. After the installer has copied over the software, it will ask you to select a USB driver.



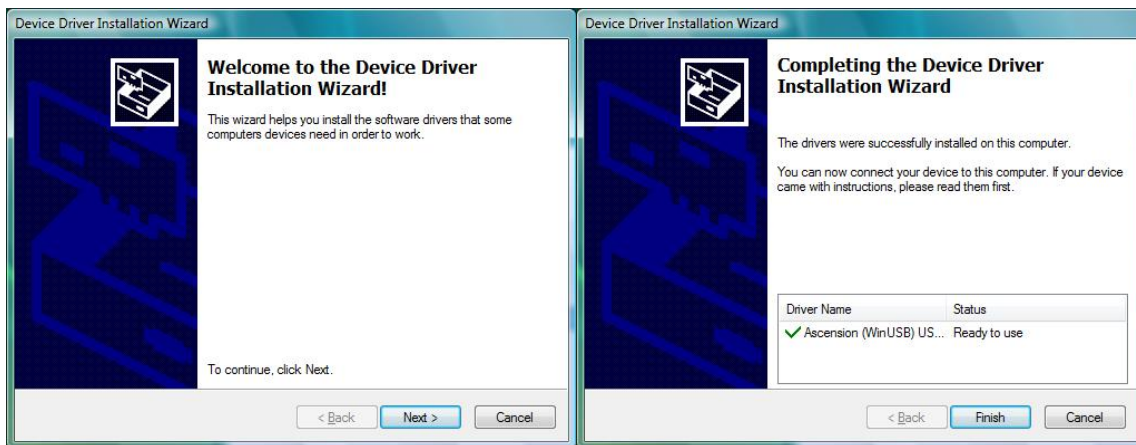
- a. For new installations, and systems running a 64-bit operating system, select the 'Windows USB driver device driver' (winusb.sys).
  - b. For continued use of the legacy USB driver (cyusb.sys), select the 'Cypress USB device driver'.
4. Note that Windows may then indicate that the software has not passed the 'Windows Logo Testing'. Select 'Continue Anyway' to proceed with the installation.



After the necessary installation files have been copied over, the *Readme.txt* file will be displayed and provide important information about the software.



5. Note that Windows XP and Vista32-bit systems may prompt the user to run the Driver Installation Wizard. Follow the prompts in the Wizard to copy over the required driver installation files.



## Cable Connections

medSAFE hardware can be setup and prepared to operate in five easy steps:

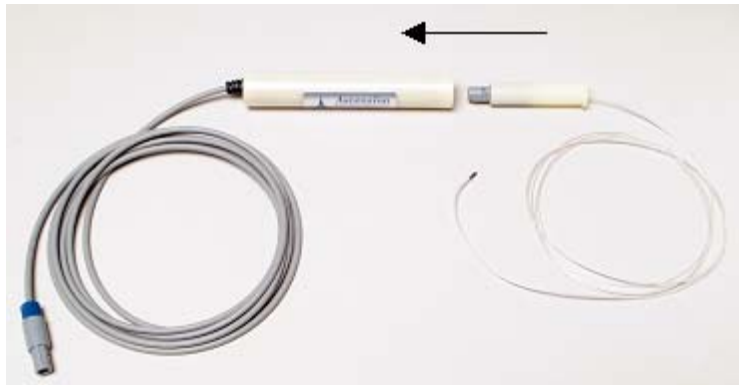
### 1. Connect the transmitter:

Plug the transmitter cable into the 37-pin connector on the electronics unit. Tighten the screws on this connector.



### 2. Connect the sensor(s) to the pre-amplifier(s):

Plug the connector of each sensor into the receptacle in the pre-amplifier module. Be sure to fully insert the connector until an audible 'click' is heard.



#### Note:

The sensor connector and pre-amp receptacle are specifically keyed to mate. You may need to rotate the sensor connector before it will click into place.

### 3. Connect the pre-amplifier(s) to the electronics unit:

Plug the connector of each pre-amplifier into the receptacles on the tracker's electronics unit.

**Note:**

Model 800 (8mm) sensors use a blue pre-amp (not shown) with a different keying from the standard pre-amp.

#### 4. Connect the AC power cord to the electronics unit:

The AC power cord plugs into the rear panel of the electronics unit. Plug the prong end into an AC source (wall outlet). Locate the power switch on the rear panel and turn it ON. The unit will not communicate until the front panel LED turns green, and the unit completes an internal initialization.



**Note:** When you turn the power switch on, the processors begin their initialization routines and read data stored in the transmitter and sensor proms. Commands should not be sent until the front panel LED is green. A blinking orange LED indicates that no valid transmitter was attached.

#### 5. Connect the USB or RS232 interface cable.

- a. If you wish to communicate using the 3DGuidance API (USB interface), connect the USB Cable to the Electronics Unit and to a USB port on the your host PC.



- b. If you will communicate with the tracker using Ascension RS232 protocol, connect the RS232 cable to the electronics unit and a COM port on the Host PC.





## Installing the Driver

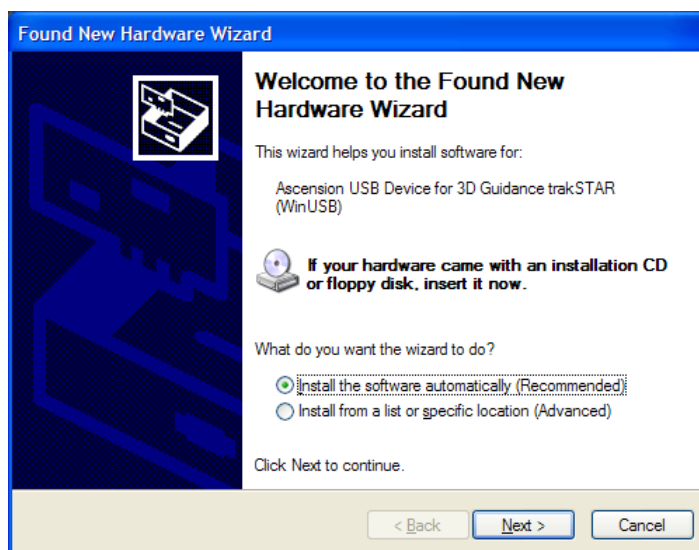
After plugging in the USB cable, your computer's operating system will indicate it has detected new hardware and start the **New Hardware Wizard**. (Note that Vista systems will initiate the driver install automatically)

1. Follow the *New Hardware Wizard* prompts, allowing Windows to Automatically search for a suitable driver. This is the default option. Note that if you inserted the CD-ROM and ran the installer prior to connecting the hardware, the Wizard will find these automatically.



### Note:

If you're going to communicate with the tracker directly using the RS232 interface, you do not need to install these drivers – go to next section.



### Tip:

The latest driver/DLLs can be downloaded from the Ascension website or by contacting technical support.

2. Follow the steps in the *New Hardware Wizard*, allowing Windows to install the USB driver. When notified that the software has not passed 'Windows Logo' testing, select 'Continue Anyway'.
3. When the Wizard completes installing the appropriate files, close the *New Hardware Wizard*.

## System Checkout

### Running the USB Demo Software

With the tracker running and the drivers and utilities installed, you are now ready to run the demo software and checkout your system.

1. Start the demo application by selecting *medSAFE Demo* from the Ascension Technology program group in the Windows Start menu.

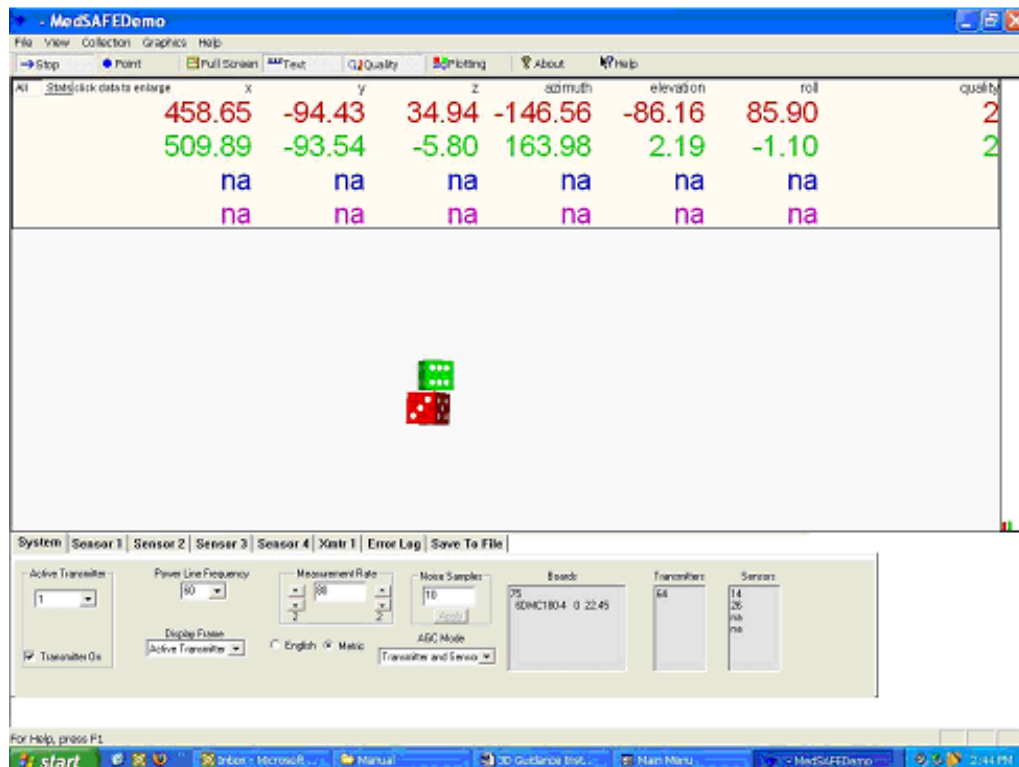
If you have installed the hardware and drivers correctly, there will be a brief pause (~3 seconds) while the application establishes communication with the tracker. Then the main window of the utility will be displayed.



2. To start collecting data from the tracker, press the **Run** button in the toolbar.

If the *medSAFEDemo* application does not run, consult the troubleshooting table for assistance.

The following is what you will see when the demo is running:



The top of the window displays numerically reported position and orientation data for each sensor attached to the tracker electronics unit. Each color-coded row contains tracking data for a single sensor. The first three values in a row represent sensor position in millimeters, relative to the transmitter. The next three values in the row represent sensor orientation in degrees.

The last column in each row presents reported QUALITY number. It gives you an indication of the degree to which position and angle measurements are in error. Errors may often be attributed to metal in the environment. See [Magnetic Distortion](#).

**Tip:** See the API Reference for additional information and assistance on using the [QUALITY](#) number.

The center of the window displays a colored cube for each sensor attached to the electronics unit. Note that colors correspond with the preceding rows of text values.

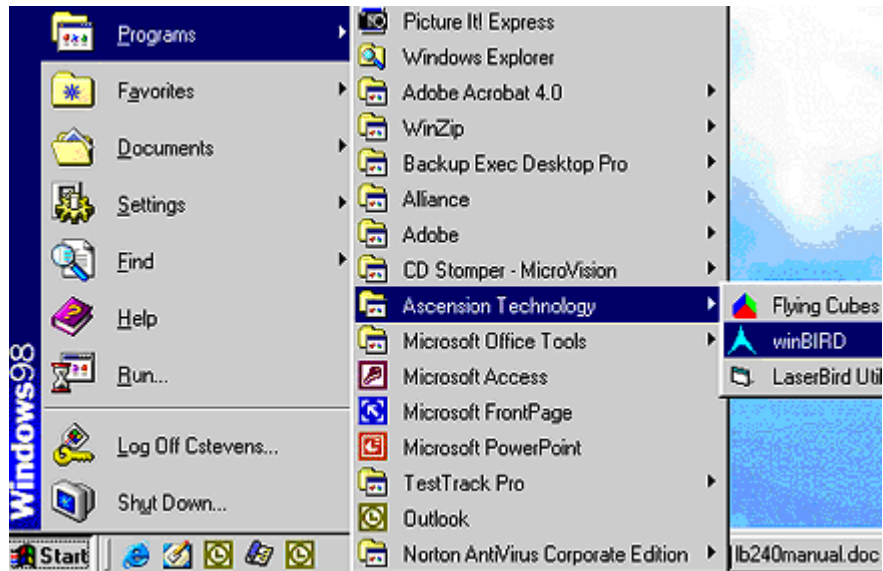
The bottom of the window allows you to configure and adjust some tracker parameters.

**Tip:** For additional information on using *medSAFE Demo*, see the program's Help menu.

Use the demo utility to acquaint yourself with the sensor's motion volume (i.e., motion box) and the tracker's measurement capabilities. If the utility does not run or the tracker does not operate as described, please consult the troubleshooting table for assistance.

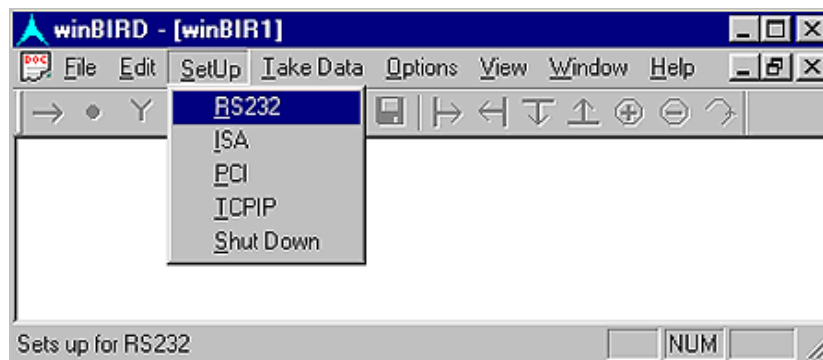
## Running the RS232 Demo Utility

1. Start the demo utility by selecting *winBIRD* from the Ascension Technology program group in the Windows Start menu.

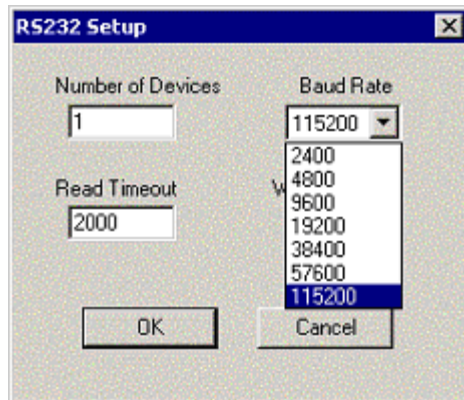


This will open the *winBIRD* window. The top of the screen contains the menu bar and toolbars, providing links to main *winBIRD* functions. At the beginning of a session most of these will be disabled, each becoming available as procedures are fulfilled.

2. Click **S**etup on the menu bar, and select **R**S232.

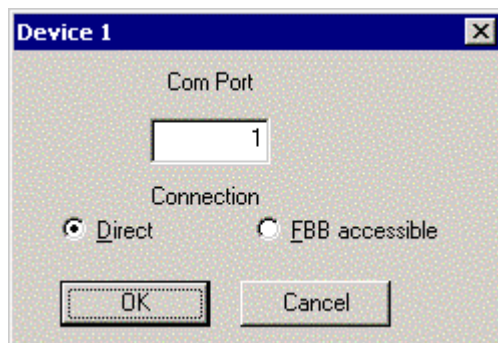


3. Change the '**Baud Rate**' to setup your COM port for the tracker's default setting: **115200**



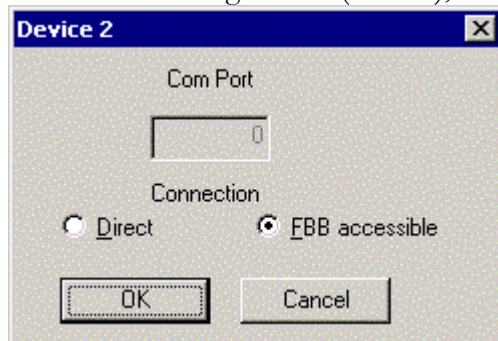
**Note:**  
115200 baud is the only baud rate currently supported by the medSAFE tracker.

4. If you have more than one sensor connected, enter the total number in the "Number of Devices" field. If one sensor, see note at left.
5. In the next pop-up window (Device 1) enter the correct Com Port for the tracker. Leave the default radio button enabled for 'Device 1'.



**Note:** For a single sensor you must still configure the tracker to *Not Standalone* mode. 'Not Standalone' or 'Group' mode' is used for a 'group' of trackers or multi-sensor systems

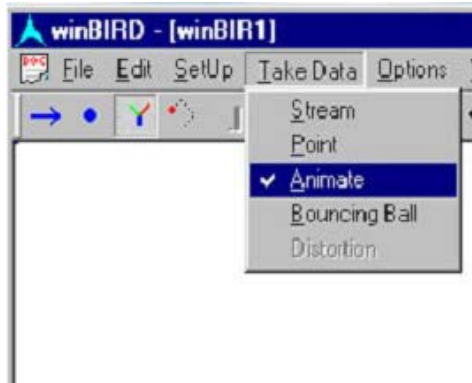
6. For the remaining devices (sensors), leave the 'FBB accessible' radio button enabled.



When you click **OK**, the main screen will indicate that the utility is establishing communication with the system: '**Waking up bird...**'

After a brief pause, the text will change to indicate '**Setup complete**', and several of the icons on the toolbar will be enabled.

7. Select '**Take Data**' from the menu bar and choose the option you would like for displaying the data.



#### What you'll see:

If you have selected '**Animate**', the screen will display a set of tri-color axes. (one axis per sensor) This axis graphically represents the sensor's translations and rotations in the motion region.

If you have selected '**Stream**' mode, the screen will continuously update two rows of data for each sensor. The three values in the first row of each sensor block represent sensor position, in inches, relative to the transmitter. The three values in the second row give sensor orientation in degrees.

'**Point**' mode is a snap-shot form of the 'Stream' mode.

8. Use the demo utility to become familiar with the sensor's motion region and the tracker's capabilities. If the utility does not run or the tracker does not operate as described, please consult the [Troubleshooting Table](#) for assistance.

# 3

## Chapter 3: Configuration and Basic Operation

*You will find a method for configuring the tracker in this chapter. It also contains a description of basic operating principles and factors that affect performance.*

When the tracker is configured at the factory, settings are optimized to meet the needs of most applications. However, you may find that it is helpful to customize the power-up behavior of your tracker to better meet your specific application requirements.

**Follow these steps to customize your system:**

1. Review the list of configurable default settings.
2. Determine which (if any) of the settings you would like to change.
3. Follow the steps in the ['Changing Your Settings'](#) section to change the power up defaults with the *medSAFE Utility*.

### Default Configuration

The following settings are installed as power-up defaults. *MedSAFE Utility* may be used to alter this default power-up configuration for Non-Dipole Transmitters only. Those settings not accessible with the *medSAFE Utility* may be changed during normal operation through the appropriate software command. Dipole transmitters include the short and mid-range transmitters. Non-dipole transmitters include 4-axis and 9-axis flat transmitters.



**Note:** The power-up configuration for dipole transmitters cannot be changed. Settings must be reconfigured at runtime using a software command.



	Dipole (Cubic Transmitter)	Non Dipole (Flat Transmitter)	Utility NonDipole Only	SW call
Baud Rate: (RS232)	115200		✓	
Measurement Rate:	80.0 Hz	40.5 Hz (4 Axis) 22 Hz (9 Axis)	✓	✓
Update Rate (Solutions/Second)	205 Hz (dipole) 162 Hz (4-Axis) 198 Hz (9 axis)			
Scale:	36 inches (91.4 cm)		✓	✓
Sensor Offsets (deg):	x = 0, y = 0, z = 0		✓	✓
Angle Align (deg):	az = 0, el = 0, rl = 0		✓	✓
Reference Frame (deg):	az = 0, el = 0, rl = 0		✓	✓
XYZRef:	False		✓	✓
Hemisphere:	Front	N/A	✓	✓
Sleep on Reset:	Enabled		✓	
Port Type:	N/A	6DOF	✓	
Report Rate:	1		✓	✓
AC Wide Notch:	Enabled		✓	✓
AC Narrow Notch:	Disabled		✓	✓
Adaptive Filter (DC):	Enabled	Disabled	✓	✓
Alpha Min:	0.02		✓	✓
Alpha Max:	0.90		✓	✓
Vm Table:	30,15,1,1,1,1,1,1,1		✓	✓
Data Record Type:	DOUBLE POSITION ANGLE			✓

### Configurable Power-up Settings:


The following may be re-configured using the *medSAFE Utility* for non-dipole (flat) transmitters only.

**IP Address** IP Address for TCP/IP communications. Not supported.

**Mask** Mask for TCP/IP communications. Not supported.

**Port** Port for TCP/IP communications. Not supported.

**Baud Rate** Establishes the default RS232 communication rate for power-up.

 Note:  
Currently the  
only supported  
Baud Rate is  
115200.



***Measurement  
Rate***

Sets the acquisition rate for the tracker. This can be altered to optimize susceptibility to distortion from certain metals. See Environment section below. Note: this is not the same as the tracker's Update Rate. Update Rate is the number of full tracking solutions iteratively computed by the tracker each second and available for your use.

***Report  
Rate***

Sets how many times the tracker should update the tracking solution before reporting a new data record. Thus, with a report rate of 1, every update is reported. With a report rate of 2, every other update is reported, etc. The report rate only applies to streaming data ([GetSynchronousRecord](#)).

***Hemisphere***

Defines the hemisphere (region), centered about the transmitter, in which the sensor makes measurements. There are six hemispheres from which to choose: the FRONT (forward), BACK (rear), TOP (upper), BOTTOM (lower), LEFT, and the RIGHT. If no HEMISPHERE parameter is specified, the FRONT is used by default. The HEMISPHERE parameter has no effect on non-dipole (flat) tracking solutions.

***Scale***

Sets the scale factor used by the tracker to report the position of the sensor with respect to the transmitter. Valid values of 36 and 72 represent the full-scale position output in inches.

***Data Format***

Sets the default data format for returned data records.

***Sensor Offsets***

By default, the tracker outputs the X, Y, Z position of the magnetic center of the sensor coil (approximate center of sensor housing) with respect to the transmitter origin. The Sensor Offsets allow you to configure the position outputs so the tracker is reporting the position of a location that is offset from the center of the sensor. See the Sensor Parameter [SENSOR\\_OFFSET](#) for details.

***Angle Align***

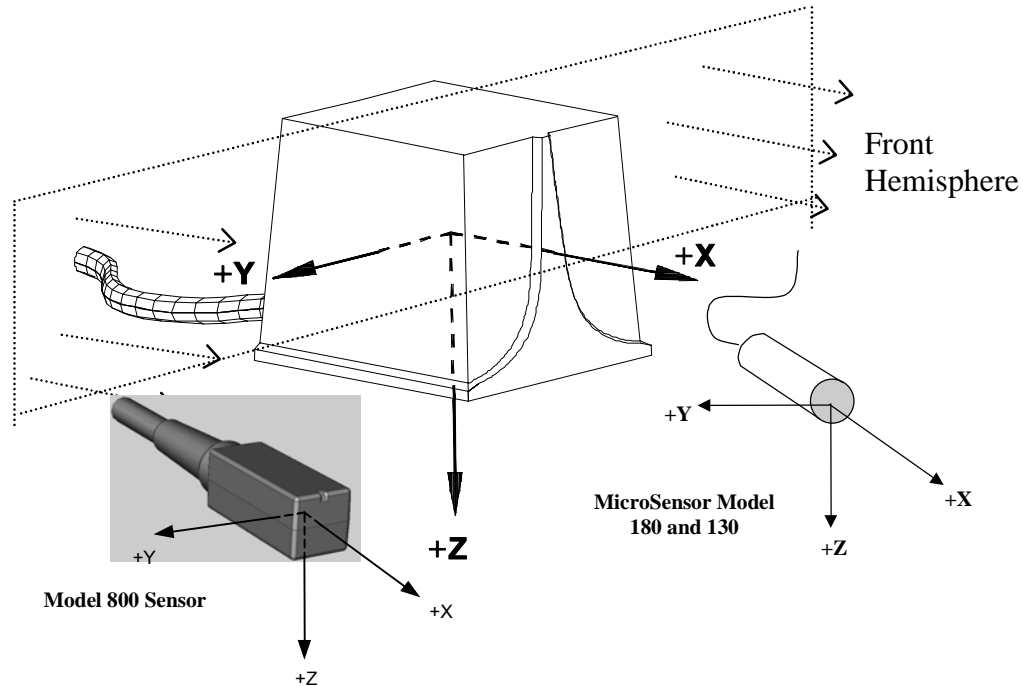
Allows you to mathematically align the sensor(s)' coordinate frame to the coordinate frame of the object being tracked. This is beneficial if you find the angle outputs for the object being tracked are not zero when in the normal 'resting' position.

***Reference Frame*** Defines the reference frame centered at the transmitter's X, Y, and Z-axes (See [Default Reference Frames](#)). Reference Frame settings allow you to enter the angles required to mathematically align the axes of the transmitter with those of a new reference frame.

- Sleep on Reset*** When enabled, this setting will cause the tracker to enter SLEEP mode after completing a reset or following power-up. In the SLEEP mode, the transmitter is turned off, but the unit will continue to respond to commands. Issue the RUN command (or 3DGuidance API equivalent) to resume normal operation. See the [\*SLEEP\*](#) command description in the RS232 Command Reference section for details.
- Port Type*** When using a non-dipole (flat) transmitter, each sensor port can either track a single 6DOF sensor or up to three 5DOF sensors. The tracking mode is chosen at startup based on this setting. There are three options: 6DOF - the port always computes a 6DOF tracking solution; 5DOF – the port always computes a 5DOF tracking solution; Auto – the port will configure itself at startup based on what sensor is plugged into the port. A 6DOF sensor will cause the port to be configured for 6DOF tracking. A 5DOF sensor or no sensor will cause the port to be configured for 5DOF tracking.
- AC Wide Notch*** An eight tap finite impulse response (FIR) notch filter that is applied to the sensor data to eliminate sinusoidal signals with a frequency between 30 and 72 HZ.
- AC Narrow Notch*** A two-tap FIR notch filter that is applied to signals measured by the tracker's sensor. You can use this filter in place of the AC WIDE notch filter when you want to minimize the delay between the measurement and outputs of the sensor data. The transport delay of the AC NARROW notch filter is approximately one third the delay of the AC WIDE notch filter.
- Adaptive Filter*** When ON, it is a low pass filter applied to sensor data that eliminates high frequency noise. Generally, this filter is always ON, unless your application permits noisy outputs. When the filter is ON, you can modify its noise/lag characteristics by changing ALPHA\_MIN and Vm.
- Kalman Filter*** These parameters are used to tune the static and dynamic tracking performance for non-dipole transmitters. These settings are to be altered by the factory only. If you think that tuning these parameters may be appropriate for your application, please contact us first.

## Default Reference Frames

### Mid and Short-Range Transmitter Reference Frame



#### *Default Transmitter/Sensor Coordinate Frames*

The origin of the short and mid-range transmitters' default Reference Frame is an approximate location at the center of the transmitter's coil set.

You can locate the reference frame origins for mid and short-range transmitters at their surface housings as follows:

Mid Range: (cable exits lower rear face)

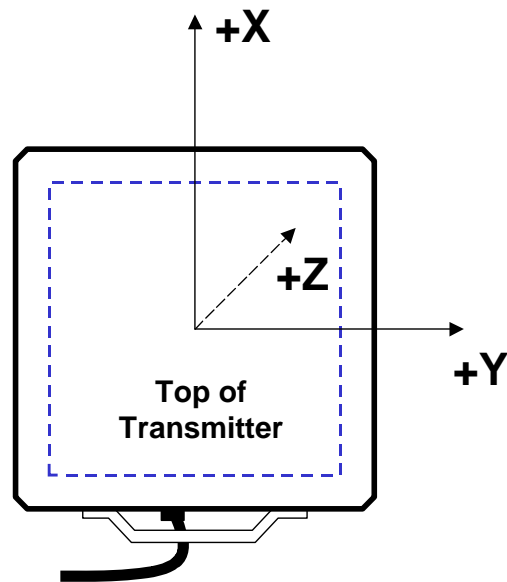
- 1.88 inches from bottom left edge
- 1.60 inches from the top face
- 1.80 inches from bottom front edge

Short Range: (cable exits lower rear face)

- 1.21 inches from front face / 1.51 inches from rear face
- 1.05 inches from the left and top faces
- 1.05 inches from right and bottom face

## Flat Transmitters

Models 600760 and 600760-9C



*Flat Transmitter Coordinate Frame*

4 Axis Flat Transmitter Origin: In the middle of the square that defines the top surface

- 11.0" from left/front edge
- 11.0" from right/rear edge
- 0" from the top face

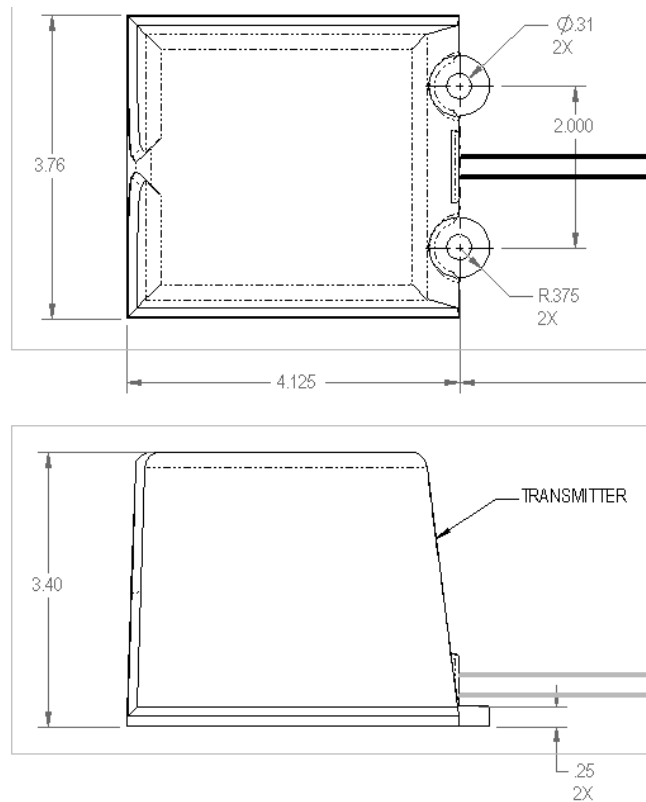
## Changing Your Settings

The *Medical Tracker Utility* may be used to alter power-up defaults for Non-Dipole Transmitters only. See [Appendix I, medSAFE Utility](#).

## Mounting the Hardware

### Mid-Range Transmitter Mounting and Location

Mount the mid-range transmitter on any non-metallic surface, such as wood or plastic. Be sure to use non-metallic bolts or 300-series stainless steel bolts.



**⊗ CAUTION!**

The mounting holes are not strong enough to hold the transmitter upside down

*Mid-range Transmitter Mounting Dimensions (inches)-top and side views*

As mentioned earlier, the mid-range transmitter's mounting holes are not designed to support the transmitter's weight when mounted upside down. If you choose to mount the transmitter upside down, use hardware that firmly holds the flanges along the front and both sides of the transmitter in addition to bolting the two mounting holes.

Never mount the transmitter on the floor (including concrete), ceiling, or walls as they may contain hidden metal objects that may affect accuracy of measurements.

Your transmitter should be located at least 24 inches from the electronics unit. It should not be placed within 10 feet of another operational transmitter since the two may interfere with one another. Contact us if you need to synch transmitters at closer ranges than described above.

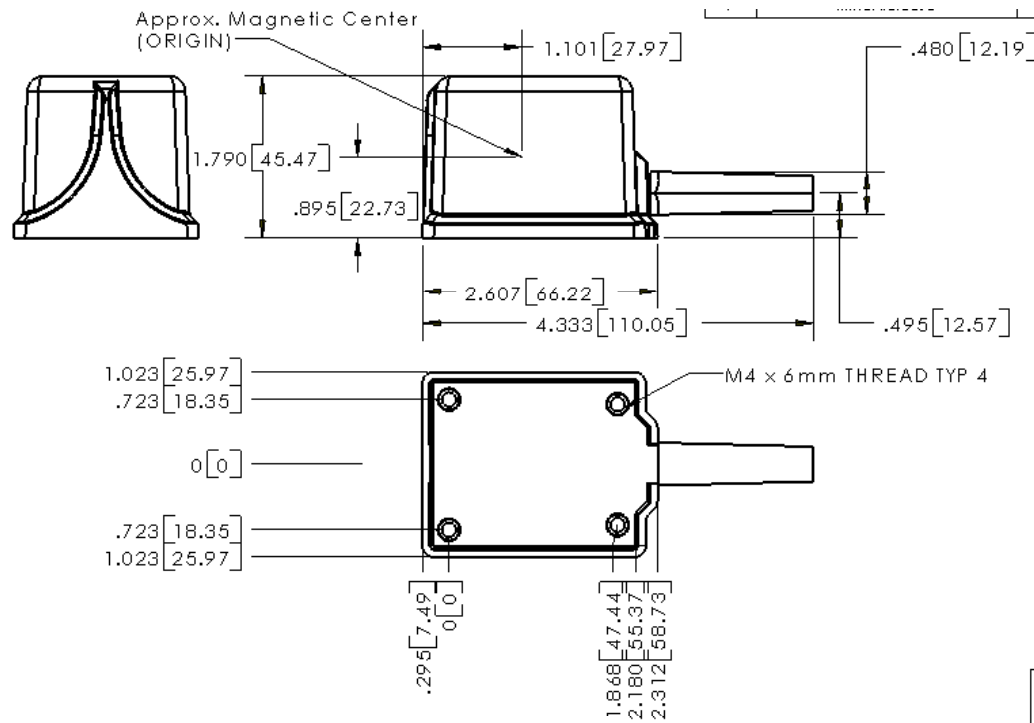
## Short-Range Transmitter Mounting and Location

Mount the short-range transmitter on a non-metallic surface, such as wood or plastic. Again, be sure to use non-metallic bolts or 300-series stainless steel bolts.

The mounting provisions for the short-range transmitter are located in the base of its housing. The four threaded inserts molded into its base accept an M4 threaded screw. Remember to never mount the short-range transmitter on the floor (including concrete), ceiling, or walls to avoid the danger that hidden metals might cause distortions in measurements.

The short-range transmitter should be located at least 24 inches from its electronics unit.

Again, keep it at least 10 feet away from another transmitter to avoid possible interference.



*Compact Transmitter Mounting Dimensions (inches)*

## Sensor Mounting

Mount the sensors on non-metallic surfaces (such as wood or plastic). Do not place sensors near power cords, power supplies, or other low frequency current generating devices (for example, CRT displays).

When tracking with multiple Model 800 sensors try to maintain a center-to-center separation of approximately 1.3 inches. If they get too close, you may notice some distortion in the measurements.

**Note:**  
To ensure proper sensor operation, treat the sensor cable very carefully.

The cables used in the Model 130 and 180 sensors are designed for integration into many instruments, including medical devices. No matter how you use them, always treat them as delicate electronic devices – sensor and the cable too. Repeated bending, pulling, or yanking of the cable will result in damage and failure of the sensor. You can also run the risk that bending, pulling or yanking the sensor - especially near its head - can tear the assembly away from its cable. The junction between the sensor head and the cable is a potential failure point and bending at this junction should be always avoided.

## Locating Your Electronics Unit

For best performance, always make sure that the electronics unit is at least 24 inches from the transmitter.

## Flat Transmitter Mounting

The flat transmitter should lie on an even surface such that it is fully supported. To provide optimal shielding, the flat transmitter should be positioned directly above the distorter.

## Pre-amplifier Mounting

The pre-amplifier is a potential distortion source because of its magnetic shielding. For maximum accuracy, it should be located as far from the tracking volume and the magnetic transmitter as practical.

## Rear Panel Connectors

There are 5 connectors on the rear panel of the electronics unit.




### Power

Standard 3-prong AC power inlet. Accepts power from 100 to 240V, at 50/60Hz.

### RS232

This is one of the two communication interfaces for the tracker. A pinout and signal description of the RS-232C interface is included in Chapter 6. Note that the tracker requires connections only to pins 2, 3 and 5 of the 9-pin interface connector.

 **Note:**  
See the [RS232 Signal Description](#) in Chapter 6.

## USB

The USB 2.0 connector is a standard USB TYPE B -Female for peripheral devices.



PIN: 1: VCC +5VDC

PIN: 2: DATA-

PIN: 3: DATA+

PIN: 4: GROUND

## SYNC

The SYNC connector provides inputs/outputs for synchronizing measurements from an external source. TTL input is accepted.



**Note:**

Synchronization not an active function at time of manual writing. Contact us for more info.

## SWITCH



The SWITCH connector may be used as an input from any analog switch. The contact closure (such as a button or footswitch) should be connected between the center conductor and the barrel of a mating BNC connector, and exhibit less than 50 ohms total when in the closed position. To be correctly detected and reported by the unit, the switch must be closed for not less than 17mS.



**Note:**

See the API Reference for data formats that include the button state.

## Basic Operation

The tracker determines the six degrees-of-freedom (6DOF) position and orientation ( X, Y, Z, Azimuth, Elevation, and Roll) of one or more 6DOF sensors and/or five degrees of freedom (5DOF) position and orientation (minus Roll) with one or more 5DOF sensors referenced to a fixed transmitter. The transmitter sequentially generates magnetic fields and the sensor instantly measures the transmitted field vectors at a point in space. From theoretical knowledge of the transmitted field, the tracker accurately deduces the real-time location of the sensor(s) relative to the transmitter. The medSAFE tracker is designed to work with two basic types of transmitters: dipole and nondipole.

### Dipole Transmitter

Dipole transmitters consist of a high permeability core with three concentric sets of coils, each coil having an axis at right angles to the other two. Magnetic fields along the X, Y, and Z-axes of the



transmitter are created when current flows in their respective windings. The strength of the magnetic field is highest near the transmitter and falls off with the cube of distance from the transmitter.

Only 6DOF sensors can be used with dipole transmitters.

### Non-Dipole Transmitter

Non-dipole transmitters contain 4 to 9 coils arranged in complex geometries within their planar housing. They employ advanced tracking algorithms to track sensors in the volume above their surfaces. These algorithms and proprietary construction techniques are designed to shield the tracking volume from magnetic distorters below the transmitter and prevent performance degradation.

6DOF sensors can be used with all non-dipole transmitters. 5DOF sensors can only be used with 9-coil non-dipole transmitters.

### 6DOF Sensor

A 6DOF sensor uses three receiving coils to provide a tracking solution that includes the position in three dimensions and the orientation of the three sensor axes relative to the tracker reference frame. 6DOF sensors are factory-calibrated and the calibration data is stored on a memory chip in the sensor's connector housing.

### 5DOF Sensor

A 5DOF sensor uses a single receiving coil to measure the position in three dimensions and the pointing direction of the sensor relative to the tracker reference frame. The roll of a 5DOF sensor is not tracked. 5DOF sensors do not require factory calibration. Identifying data such as part number and serial number can be stored in an optional memory chip in the sensor's connector housing. Because of the simplicity of their design, 5DOF sensors can be fabricated in much smaller form factors than 6DOF sensors.

### Pre-Amplifier

Sensor signal strength decreases as coil size decreases. For micro-miniature sensors to track movement with high accuracy, we must amplify their signals. This is accomplished with preamplifier devices that are factory-calibrated for interchangeability. Calibration data is stored in a memory chip in the pre-amplifier connector. Pre-amplifiers are both electrically and magnetically shielded to minimize noise.

A special reduced-gain preamplifier is required for use with the model 800 sensor to prevent saturation. This preamp is keyed specifically to accept only model 800 sensors.

### Electronics

In addition to computing tracking solutions, the electronics unit contains the transmitter drive circuitry, sensor signal processing, data conversion, processing, power conditioning, and host interface functions.

The transmitter drive is a precision current source, with a maximum output of 3.0A. The system detects the absence of a transmitter by monitoring the current. If current is interrupted, the transmit driver will be turned off until a valid transmitter EEPROM is detected. This ensures that the connector is de-energized when open. Also, the transmitter is fault-protected for ground shorts. In the event of a short to ground on any transmitter pin, no damage will result to the tracker nor will the tracker create a hazardous current.

The sensor signal processing circuitry acquires the signal from the sensor for each of the 3 axes and continuously converts it to a digital value. This input digital value is summed in an accumulator (digital integrator) and the final value is output and used by the algorithm to derive a tracking solution. The sensor connector is also fault protected for ground shorts. No damage to the system or excessive current hazards will result from shorting any sensor connector pin to ground.


The electronics unit contains six onboard processors:

- PServer: It handles all communications to and from the host PC as well as computes the tracking solutions.
- Acquisition/MDSP: It performs all acquisition and digital signal processing of the sensor data.
- PODSP: When using non-dipole transmitters, an additional co-processor per sensor port computes the tracking solutions.

## Measurement Cycle

The tracker electronics unit activates transmitter coils sequentially and outputs a data record at the end of each cycle. Once each transmitter coil has been activated, a measurement cycle is complete and a new cycle begins. Thus, the update rate for a dipole (3-coil) transmitter running at measurement rate of 50 Hz will be 150 tracking solutions per second (3 coils x 50 Hz = 150 solutions per second). A non-dipole transmitter with 9 coils running at 20 Hz will output 180 tracking solutions per second.

See the Default Measurement Rate paragraph below for a discussion of measurement rate effects on performance.

 **Note:** The update rate available from the tracker with a dipole transmitter is always 3 times its measurement rate.

## Calibration

Each component is manufactured within tight tolerances, but differences still exist between components. These differences are measured, recorded, and adjusted for in the system through the tracking algorithm. Calibration values represent the measured difference between any particular component and the ideal for that component.

 **Note:** 5DOF sensors do not require factory calibration.

Calibration allows components to be interchanged with minimal effect on the resulting tracking values. The calibration values for each component are stored within the hardware of the individual component. For the Transmitter and Sensor, the values are stored in an EEPROM at the connector.

For the Electronics unit, the calibration values are stored in an EEPROM mounted on the printed circuit board. Non-dipole Transmitters include an additional Flash memory chip in the connector. The calibration data in each component is programmed at the factory and is read-only.

## Performance Factors

### Electromagnetic and Other Interference in Tracking

Other electrical and magnetic devices sharing the immediate volume with the tracker may influence tracking data.

The following factors may affect the stability of the tracking area and thus the tracking accuracy:

- Excessive electrical noise
- Magnetic distortion

#### Excessive Electrical Noise

If the background magnetic field is not constant during the measurement cycle, the tracking data will contain noise. Noise is the seemingly random jumps in position and orientation.

When the sensor is at rest, evaluation of noise in the data will show that the jumps are random and centered on a stable position. Calculation of the mean of the position data will provide the true sensor position.

#### CAUSES OF NOISE

There are two conditions that cause noise in tracking data. These are noise generated from internal sources and noise generated from external sources.

The most prevalent is noise from external sources. External sources of electrical noise include electrical motors, switching power supplies, fluorescent lighting, video CRT monitors, uninterruptible power supplies, and wiring or devices which use or carry large amounts of electrical current that vary over time.

External factors can alter the background magnetic field from one moment to the next. This makes absolutely correct magnetic background subtraction in the tracking device impossible, resulting in slightly unstable results.

Internal sources include factors such as small variations in measurement timing, amplified electronic component thermal activity, algorithm division by very small numbers, and unsuppressed electrical power line noise.

## REDUCING NOISE

Powering off suspect electrical equipment is often the best method of determining sources of noise. Once a source of noise is discovered, removal of the device from the area or turning the power off during tracking is effective in reducing noise. Critical equipment may be shielded as long as the shielding does not result in metal distortion (see “Distortion” section below).

Increasing the distance between the noise source and the sensor or decreasing the sensor distance from the transmitter will reduce the noise. These actions will result in an increase of measured signal from the transmitter relative to the noise level (increased signal-to-noise ratio).

### Magnetic Distortion

Distortion is a constant deviation from the correct value. Unlike noise, distortion is a constant deviation as a function of position. The distorted tracking values are incorrect and averaging the data does not improve the values.

When the sensor takes measurements in the presence of distortion, the tracking device continues to calculate position and orientation based on theoretical knowledge of the undistorted transmitted field. The resulting difference between the calculated location and orientation, and the actual location and orientation, is distortion.

## CAUSES OF DISTORTION

Most often the cause of distortion is magnetic and/or electrically conductive metal near the tracking volume or motion box. The ferrous magnetic property of the metal, the electrical conductivity of the metal, the physical orientation, and other physical features will all alter the level of tracking distortion.

The ferrous magnetic property of the metal will distort the transmitted magnetic field from the tracker.

The electrical conductivity of the metal may distort the transmitted magnetic field. Our pulsed DC- tracking technology has a high immunity to distortion caused by residual eddy currents. Note though that physical factors, such as electrically complete loops, can sustain eddy current loops long enough to interact with the tracking field during sensor measurements.

The metal will interact with the transmitted fields, altering the field relative to the tracking system algorithm expectation.

Another common source of distortion is altered tracking components. For example, sensor and transmitter extension cables can cause a change in the electrical characteristics of the device. If this alteration is performed without the direction of Ascension, the change will not be compensated for in calibration. Any physical change to the core tracking electronic components or in the physical connection of the system has the potential of causing distortion.

## REDUCING DISTORTION

As noted, metal is the primary cause of distortion. Removal and reduction of the amount of metal in and around the tracker is most effective means of controlling distortion in measurements. Sources of metal distortion cannot be shielded, as is often the case with noise sources. You can however choose our flat transmitter as a means of controlling distortions from metal contains beneath its flat surface. Contact us for more information about optimizing performance in metallic environments with this transmitter.

If metallic distortion is an issue, consider replacing nearby metal objects with non-metallic ones. Structural fiberglass, plastics, woods, and ceramics are good replacements. Of the metals available, nonmagnetic and high electrically resistive metals are the next desirable. Some alloys of stainless steel (300-series - medical grade) can be used near the tracker with minimal effect on performance. Brass and aluminum are less desirable and are not recommended, but they may be used in some situations. Care must be observed, as machined metal may become magnetic. All nearby metals should be tested with the tracker before finalizing design or use.

As mentioned, eliminating or reducing metals in the tracking volume is recommended. Since the tracker is not a line-of-sight device, care must be taken that the whole volume around the transmitter is considered with regard to metal. The area behind and under the transmitter should be examined and modified as closely as the area between the sensor and the transmitter.

## METAL DETECTION

Distortion due to metal may be monitored through the use of an extra sensor mounted a fixed distance from the transmitter. Mount the fixed sensor at or near the maximum distance used by the unhindered sensors. Monitor the fixed sensor's position and orientation for significant deviations. Situations that distort the fixed sensor's measurements will distort the rest of the system. The application should flag the user during one of these distortion events.

## QUALITY/ METAL NUMBER

Alternatively, you may wish to experiment with the use of our QUALITY number. Also referred to as the METAL error or Distortion number, its returned value will give you an indication of the degree to which the position and angle measurements are in error. See the Quality Sensor Parameter Type for details.

## Tracker as the Cause of Interference

Just as some tracker components may be subject to interference from electrical equipment in the immediate environment, so too the tracker's magnetic fields may possibly interfere with nearby electrical systems, e.g., an EKG. It is up to you to identify nearby devices and make sure their performance is not degraded when you are simultaneously using the tracker. If you rely on life-sustaining equipment, such as a pacemaker or defibrillator, be sure to consult with your physician prior to powering up this tracking device.

## Factors in Tracker Accuracy

### Warm-up

As with most electrical components, the tracker goes through a period of drift as it reaches a thermal equilibrium. The system shall meet accuracy specifications within two (2) minutes. For effective warm up, the transmitter must be running.

A previously unused sensor may be swapped without worry about sensor warm-up drift.

### Default Measurement Rate

Mid and short-range transmitters: The selected measurement rate will have a small effect on accuracy. The system is calibrated at the default measurement rate of 80.0 Hz measurements per second. At this rate, the tracker will be most accurate.

In some cases, you may not want to use the default measurement rate. Here are a few reasons why:

- The application requires a specific measurement rate (e.g. it must be in synchronization with video update rate or another measurement tool).
- The environment is electrically unstable at the default measurement rate or significantly more stable at another measurement rate.

Measurement rates that significantly differ from the default rate will reduce the system accuracy. Furthermore, the measurement rate for the tracker cannot be set below 20Hz.

### Equipment Alteration

Each tracking component has been calibrated to measure the difference between it and the ideal component of that type. This calibration information is stored on an EEPROM in the respective component.

Any alteration that will affect the electrical properties of a component or change the access to the calibration data should be avoided. Changes in cable length through addition of an extension cable, adding a connector, or cutting and shortening any cable will result in tracking problems. Altering any of the board jumpers or settings will degrade accuracy. Changing any component on the tracking board will degrade accuracy.

### Power Grid Magnetic Interference

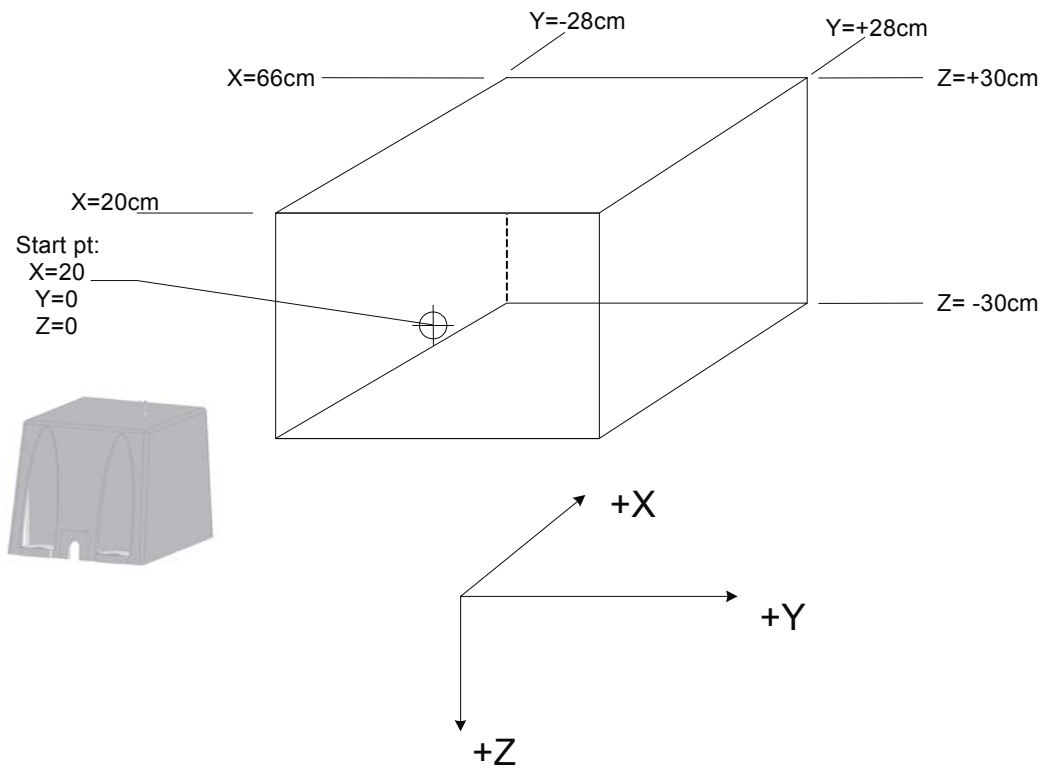
The power grid frequency in North America is 60 Hz. In Europe, it is 50Hz. Magnetic radiation from the power grid can interfere with the tracker measurements.

Advanced software filters are now implemented in medSAFE to reduce this effect without compromising dynamic performance, but it is still a potential source of noise in the system. Highest performance will be achieved by operating the system away from walls, floors, or other structures in which electrical wiring is routed. Also, high current devices such as heaters, motors, and transformers

should be kept as far away from the system as practical. The filter coefficients used to reduce power grid magnetic interference require the correct power line frequency value to operate effectively. The application developer and the user should thus input the correct frequency to the tracker through the API.

### Performance Motion Box

All tracking components are subjected to a calibration procedure that optimizes performance over a given region. This region is referred to as the Performance Motion Box. In these regions, tracking accuracy is the greatest. If you are developing an application that requires high tracking accuracy, be sure to position the transmitter such that critical measurements are taken in these regions. Tracking outside the box may not yield results with equivalent accuracy.



*Performance Motion Box: Mid-Range Transmitter and 8mm Sensor*

Dimensions in each axis for the mid-range transmitter are:

For **Model 800** sensors:

**X = 20 to 66cm** from the transmitter center

**Y =  $\pm 28$  cm** from the transmitter center

**Z =  $\pm 30$  cm** from the transmitter center

For **Model 180** sensors:

**X = 20 to 51cm** from the transmitter center

**Y =  $\pm 23$  cm** from the transmitter center

**Z =  $\pm 15$  cm** from the transmitter center

For **Model 130 and 90\*** sensors:

**X = 20 to 36cm** from the transmitter center

**Y =  $\pm 20$  cm** from the transmitter center

**Z =  $\pm 20$  cm** from the transmitter center

\*Note –Recommended max range for the Model 90 sensor with the mid-range transmitter is 36cm (14")

For the Short-Range Transmitter:

For **Model 800** sensors:

**X = 15 to 41cm** from the transmitter center

**Y =  $\pm 12$  cm** from the transmitter center

**Z =  $\pm 12$  cm** from the transmitter center

### Flat Transmitter Motion Box

When operating the tracker with a Flat Transmitter, it is important to note that the motion box for the tracked sensors is **smaller** than the area covered by the dimensions of the transmitter's housing. It also **does not start immediately above the transmitter's top surface**.

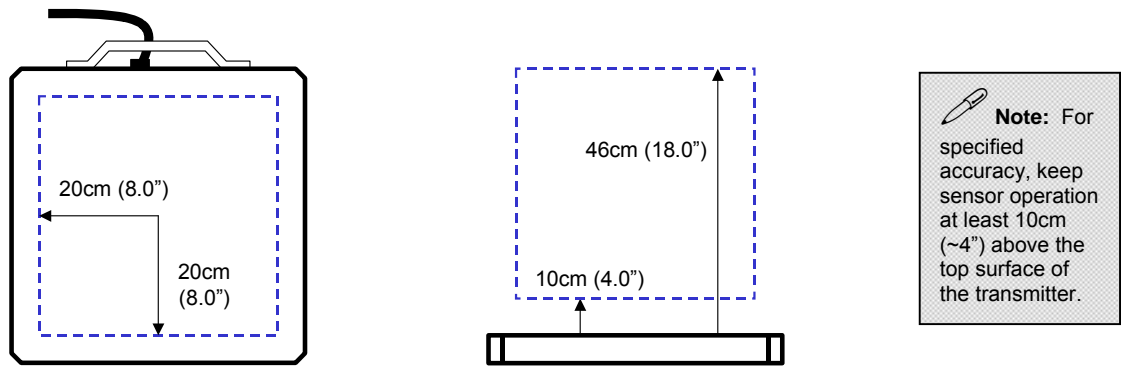
Specified accuracy will only be achieved over the motion box: that meets the following:

- Motion Box **begins 10.2cm (4.0 inches) above** the top surface of the transmitter
- Motion Box is **7.6cm (3.0 inches) in from each side** of the transmitter
- Motion Box **ends 40.6cm (16.0 inches) above** the top surface of the transmitter



**Note:** The motion box for the flat transmitter is smaller than the dimensions of the housing. Specified accuracy will only be achieved over the motion box shown.





*4-Axis Flat Transmitter Motion Box*

In these regions, you will find the best tracking accuracy. If you are developing an application that requires high tracking accuracy, be sure to position the transmitter such that critical measurements are taken in these regions. Tracking outside the box may not yield results with equivalent accuracy.

In the performance motion box, you will find the best tracking accuracy. If you are developing an application that requires high tracking accuracy, be sure to position the transmitter such that critical measurements are taken in this region. Tracking outside the performance motion box may not yield results with equivalent accuracy.

# 4

## Chapter 4: Software Operation: Tools for Successful Tracking

*This chapter outlines the interfaces and methods for communicating with the 3D Guidance medSAFE. It also includes sample programs for the USB and RS232 interface protocols.*

### Software Overview

You can communicate with 3D Guidance medSAFE by using either of two Ascension interface protocols: **3D Guidance Windows API** or the **Ascension RS232 interfaces**.

### 3DGuidance API

You will use our 3D Guidance API to communicate with the tracker via USB.. This interface is standard among older Ascension tracking devices and has been continued for the new generation of 3D Guidance trackers. If you have developed an application using the pciBIRD/3D Guidance driver for PCI bus-based trackers, you will find integration of 3D Guidance medSAFE virtually seamless.

If you are new to the 3DGuidance API, you will find several tools on the CD-ROM to assist in experimenting with the tracker's capabilities and in quickly creating custom code. These tools include two demo applications, medSAFE Demo and medSAFE API Test, and two programs containing sample C++ project files. The tables below describe both the tools available and their location on the CD-ROM. See [3DGuidance API Reference](#) for further details.

<u><b>API Components</b></u>	<u><b>Description</b></u>	<u><b>Location</b></u>
<b>ATC3DGm.h</b>	Header file that contains the definitions of the constants, structures, and functions needed to make calls to the API. Calls defined here can be used in a developer's code by including this file in the project that makes calls to the API	CD-ROM: ATC 3DGuidance API Or Program Files\Ascension\3D GuidanceXXXX\3D Guidance API\
<b>ATC3DGm.lib</b>	Library file required during compiling of any code that makes calls to API	
<b>ATC3DGm.DLL</b>	Dynamic Link Library - This file is needed in the Windows System folder (or DLL search path) to support all the function calls described in the header file.	

<u><b>Applications</b></u>	<u><b>Description</b></u>	<u><b>Location</b></u>
<b>medSAFEDemo.exe</b>	<p>A Demo Windows application that displays tracking data (both test and graphical representation) for up to 12 sensors (see <a href="#">ATC3DGm.ini File</a> for information on configuring for more than 4 sensors.) Supports the following functions:</p> <ol style="list-style-type: none"> <li>1. TX On/off -turn the transmitter on or off</li> <li>2. System RESET - reset/re-initialize the tracker</li> <li>3. System settings - change measurement rate, line frequency, AGC mode, English vs. metric units</li> <li>4. Sensor settings- change quality parameters, angle align, and filter settings</li> <li>5. Transmitter - change reference frame</li> <li>6. Graph Mode - Plot up to 3 parameters</li> <li>7. Noise Statistics - Collects X samples and shows AVG, pk-to-pk and RMS deviation</li> </ol>	<p>CD-ROM: medSAFEDemo Or Program Files\Ascension\3D GuidanceXXXX\ medSAFEDemo</p>
<b>medSAFEAPITest.exe</b>	<p>A Windows application that allows the user to send any command defined in the API to the tracker, and to view the associated response. All function calls and the possible arguments for those calls are selected via pull-down menus, so re-compiling/re-build is not necessary. This allows user to:</p> <ol style="list-style-type: none"> <li>1. Check response to particular command without implementing in your own code</li> <li>2. Check response using a particular non-default setting (filter, etc) without implementing in their code</li> <li>3. Confirm/Debug hardware and their code by reproducing chosen settings and viewing response</li> <li>4. Save system settings that have been tried to an .ini file (using SaveSystemConfig call)</li> </ol> <p>NOTE: It's important to know how commands are defined and sequence of commands to send for this tool to be used effectively.</p>	<p>CD-ROM: medSAFE API Test Or Program Files\Ascension\3D GuidanceXXXX\ medSAFE API Test</p>

### Sample Programs

<u><b>Sample Code</b></u>	<u><b>Description</b></u>	<u><b>Location</b></u>
<b>Sample.exe</b>	<p>This C++ project contains sample code for a very simple console application that demonstrates fundamental communication with the tracker using the 3DGuidance™ API. Specifically, it:</p> <ol style="list-style-type: none"> <li>1. Initializes the Tracker</li> <li>2. Reads in the system configuration (status of board, sensors, TX, etc)</li> <li>3. Turns on the transmitter</li> <li>4. If all above is valid, collects 100 records (POS/ANG format) from each of the sensors and streams them to the screen.</li> <li>5. Error Handler - A simple error handler is included. It just takes any error codes returned from the DLL/Tracker, and sends them to the screen</li> <li>6. TX Off - Before closing the application, shows how to turn off the TX</li> </ol> <p>All necessary source files to re-build and run the application are included, and each of the above steps are accompanied by detailed comment descriptions directly in the code.</p>	<p>CD-ROM: \Samples\Sample\ Sample.dsp or Program Files\Ascension3D GuidanceXXXX\Samples\ Sample\Sample.dsp;</p>

<u><i>Sample Code</i></u>	<u><i>Description</i></u>	<u><i>Location</i></u>
<b>Sample2.exe</b>	<p>This C++ project also contains sample code for a very simple console application using the 3DGuidance™ API. It is more comprehensive than "Sample", in that it shows how to use each of the GETXXX/SETXXX calls appropriately. These calls give access to all configurable tracker parameters. An additional feature: also shows command for saving configuration settings to an .ini file</p> <p>All necessary source files to re-build and run the application are included, and each of the above steps are accompanied by detailed comment descriptions directly in the code.</p>	<p>CD-ROM:  \Samples\Sample\ Sample2.dsp  or  Program Files\Ascension3D  GuidanceXXX\Samples\  Sample2\Sample2.dsp;</p>
<b>GetSynchronousRecordSample.exe</b>	<p>This C++ project also contains sample code illustrating the usage of the new data record streaming functionality provided by the GetSynchronousRecord command. This new command allows the user to easily acquire each data record from the tracker without missed or duplicate records.</p> <p>All necessary source files to re-build and run the application are included, and each of the above steps are accompanied by detailed comment descriptions directly in the code.</p>	<p>CD-ROM:  \Samples\  GetSynchronousRecordSample\  GetSynchronousRecordSample.  dsp  or  Program Files\Ascension3D  GuidanceXXX\Samples\  GetSynchronousRecordSample\  GetSynchronousRecordSample.  dsp</p>

## Ascension RS232 Interface

You can communicate 3D Guidance medSAFE via RS232 by using the Ascension RS232 protocol. It will give you instant access to a wide range of applications.

There are two options to access the 3DGuidance via this RS232 interface:

1. Use an existing driver for the 3DGuidance (i.e. SGI, Win32)
2. Communicate directly via the **Ascension RS232** protocol

### Ascension RS232 Driver

The Ascension RS232 Windows driver provides an interface to trackers for WIN32 applications. With the exception of the initialization routines, this interface is standardized across many platforms and devices. The following trackers support this driver: Flock of Birds, pcBIRD, miniBIRD, MotionStar, MotionStar Wireless, laserBIRD, phasorBIRD, and 3Dguidance medSAFE. To use the driver, simply include the header file Bird.h in your source code, link with the library module Bird.lib, and put the dynamic-link library Bird.dll anywhere on the DLL search path. Definitions for the data structures and API calls documented here are in the file bird.h. The Windows Driver User Manual and driver files are located in the \FOB\_WIN32 directory on your 3DGuidance CD-ROM.

### Direct Communication: Ascension's RS232 Protocol

The Ascension RS232 protocol involves interpreting RS232 commands sent directly to your tracker.

### RS232 Sample Program

An example of the basic structure that applications should follow to communicate with the medSAFE using this interface can be found in the sample program *Terminal*. This program utilizes the basic elements (RS232 commands) required to establish communication, configure the data format, and output data to the HOST screen display. Users will find the source code and the executable file located in the `\FOB_Direct` subdirectory on the 3DGuidance CD-ROM.

NOTE: You should use this program as an example only. Follow the description included with each command in the [RS232 Command Reference](#) section of Chapter 6 for details of Command usage in your specific application.

```
Ascension Technology Corporation - 3D Guidance RS-232 Example 08/20/2009
  X      Y      Z      A      E      R
+11.72  -1.06  -5.50  +25.6  -13.5  +39.5

Press any key to exit
```



## Chapter 5: 3DGuidance API Reference

*This chapter will show you how to write an application that will access the tracker using the 3DGuidance API provided in ATC3DGm.dll .It also describes the setup of the tracker's various parameters.*

### Using 3D Guidance medSAFE

These sections describe how to use the 3DGuidance API to perform the following operations:

[Quick Reference](#)

[System Initialization](#)

[System Setup](#)

[Transmitter Setup](#)

[Sensor Setup](#)

[Acquiring Position and Orientation Data](#)

[Error Handling](#)

## Quick Reference

### SYSTEM

The following system setup operations are available. All these parameters may be setup or the current status may be interrogated by calling [SetSystemParameter](#) and [GetSystemParameter](#). All of these parameters affect the operation of all transmitters and sensors in the system and cannot be modified on a sensor-by-sensor or transmitter-by-transmitter basis. The power up system **defaults** for short and mid-range transmitter configurations are as follows:

- Select Transmitter: No transmitter selected
- Power Line Frequency: 60.0 (Hz)
- AGC Mode: Sensor AGC Only
- Measurement Rate: 80.0 (Hz)
- Report Rate: 1 (Report every update)
- Maximum Range: 36.0 (inches, Maximum range)
- Metric: True (floating point output representation is in inches)

- [\*\*SELECT\\_TRANSMITTER\*\*](#)

This command allows us to turn on next transmitter or turn off the current transmitter in the medSAFE tracker. A full description of the operation is found at [SELECT\\_TRANSMITTER](#).

- [\*\*POWER\\_LINE\\_FREQUENCY\*\*](#)

This parameter represents the frequency of the AC power source used by the system. It is necessary to set this for proper operation of the [AC\\_Wide\\_Notch\\_Filter](#).

- [\*\*AGC\\_MODE\*\*](#)

medSAFE has two modes of operation.

- 1) [\*\*TRANSMITTER\\_AND\\_SENSOR\\_AGC\*\*](#)

(Not currently supported) In this mode, an automatic gain control (AGC) system implemented in the firmware will dynamically adjust the gain of the input variable gain amplifier (VGA)

AND the power level of the transmitter in order to keep the sensor input signal within the dynamic range of the system.

## 2) [SENSOR AGC ONLY](#)

In this mode, the firmware will only adjust the gain of the VGA. The power level of the transmitter is never altered and remains set at full power.

- [MEASUREMENT RATE](#)

The measurement rate is the tracker's sample rate. For the mid-range transmitter system configuration the measurement rate is nominally set at 80.0 measurements/second. You can increase the tracker's measurement rate to a maximum of 125 measurements/sec. The downside of selecting a rate faster than 103 measurements/sec is that you may notice increased noise and distortion errors in measurements. . You can decrease the tracker's measurement rate to no less than 9 measurements/sec. Decreasing the measurement rate is useful to reduce errors resulting from highly conductive metals such as aluminum. If you have low-conductive, highly permeable metals in your environment, such as carbon steel or iron, changing the measurement rate will not change the distortions. For low-conductive, low permeability metals such as 300-series stainless steel or nickel, you do not need to lower the default measurement rate to achieve maximum metal immunity.

- [REPORT RATE](#)

The report rate determines how many times the tracker will update its solution before reporting a new data record. The report rate can be an integer from 1 to 127. E.g., a report rate of 3 will result in the tracker reporting a new data record for every 3rd acquisition/update.

- [MAXIMUM\\_RANGE](#)

This represents the scale factor for position data returned as signed binary integers. The position scaling can be set to either 36 or 72 inches. The value set will be the maximum possible full-scale value returned by the tracker.

- [METRIC](#)

This tracker option allows the data to be formatted in double precision floating point for outputs pre-scaled to either inches (the default) or millimeters. Setting the METRIC flag true will cause output to be in millimeters.

## SENSOR

The following operations and set up can be performed individually for each sensor. The parameters can be set and read by making calls to [SetSensorParameter](#) and [GetSensorParameter](#). Upon power up, each sensor channel is setup with the following **defaults**:



- Data Format: Double precision floating point Position/Angles
- Angle Align: 0, 0, 0
- Filters: All values in **Alpha max** table = **0.9000**, all values in **alpha min** table = **0.0200**, **Vm table** values = **2, 4, 4, 4, 4, 4, 4** (This is for the mid-range transmitter. Vm values depend on the type of transmitter and sensors)
- Hemisphere: Front hemisphere (in front of the ATC logo on the transmitter)
- Metal Distortion: Filter alpha = 12, Slope = 0, Offset = 0 and Sensitivity = 2.

- **DATA FORMAT**

The following data record formats are available in integer and floating point representation. Combinations of these formats are also available in the same data record.

- ANGLES:

Data record contains 3 rotation angles. See [SHORT ANGLES RECORD](#), [DOUBLE ANGLES RECORD](#)

- POSITION:

Data record contains X, Y, Z position of sensor. See [SHORT POSITION RECORD](#), [DOUBLE POSITION RECORD](#)

- MATRIX:

Data record contains 9-element rotation matrix. See [SHORT MATRIX RECORD](#), [DOUBLE MATRIX RECORD](#)

- QUATERNION:

Data record contains quaternion. See [SHORT QUATERNIONS RECORD](#), [DOUBLE QUATERNIONS RECORD](#)

- TIME\_STAMP and METAL DISTORTION status:

Some data formats include a TIME\_STAMP and/or a METAL\_DISTORTION status field. See [DOUBLE POSITION TIME STAMP RECORD](#), [DOUBLE POSITION TIME Q RECORD](#)

- BUTTON status:

Data record contains a Button field. This field gives the open/close state of a contact closure connected to the BNC connector on the rear panel of the tracker labeled SWITCH. See [DOUBLE POSITION ANGLES TIME Q BUTTON RECORD](#), [DOUBLE POSITION MATRIX TIME Q BUTTON RECORD](#), [DOUBLE POSITION QUATERNION TIME Q BUTTON RECORD](#)

Using the [SetSensorParameter](#) command these data formats can be set up for each individual sensor.

See [DATA FORMAT TYPE](#) for all available data format combinations.

- [ANGLE ALIGN](#)

Aligns sensor to reference direction

These parameters can be set up for each individual sensor by using the [SetSensorParameter](#) command. The current setting of [ANGLE ALIGN](#) can be accessed using the [GetSensorParameter](#) command.

- [FILTER ALPHA PARAMETERS](#)

This is an adaptive alpha filter. It is initialized to a default condition but changing the values in 3 tables, which are contained in the [ADAPTIVE PARAMETERS](#), can modify its operation. These parameters are changed through use of the [SetSensorParameter](#) function call and the current state of the alpha filter parameters may be observed by calling the [GetSensorParameter](#) function call.

- [FILTER AC NARROW NOTCH](#)

- This is a 2-tap finite impulse response (FIR) notch filter that is applied to signals measured by the tracker's sensor to eliminate a narrow band of noise with sinusoidal characteristics. This filter can be selected/deselected and interrogated through the [SetSensorParameter](#) and [GetSensorParameter](#) function calls.

- [FILTER AC WIDE NOTCH](#)

This is a 6 tap finite impulse response (FIR) filter that is applied to the sensor data to eliminate signals with a frequency between 30 and 72 Hz. Note: for this filter to work properly the system parameter: [POWER LINE FREQUENCY](#) must be set correctly using the [SetSystemParameter](#) function call.

- [FILTER LARGE CHANGE](#)

If selected this filter will lock the output data to the current position and orientation if a sudden large change in position or orientation is detected.

- [HEMISPHERE](#)

The [HEMISPHERE](#) command lets you establish the hemisphere in which you will be tracking sensors. It determines which of the 6 possible hemispheres of the transmitter in which your sensor(s) are tracking. It can be set up for each sensor by using the [SetSensorParameter](#) command. The current [HEMISPHERE\\_TYPE](#) can be accessed using the [GetSensorParameter](#) command.

- [QUALITY](#)

This command adjusts the behavior of the “Quality” accuracy degradation indicator contained in several of the data record formats. See [DOUBLE POSITION TIME Q RECORD](#), [DOUBLE ANGLES TIME Q RECORD](#), [DOUBLE POSITION ANGLES TIME Q RECORD](#) for examples. See also [DATA FORMAT TYPE](#) for a list of all data formats. Those format types containing “\_Q\_” indicate the presence of the “quality” value.

The user can modify the sensitivity and response of the quality number returned. These parameters can be set up for each individual sensor by using the [SetSensorParameter](#) command. The current setting of The METAL DISTORTION parameters can be accessed using the [GetSensorParameter](#) command. See [QUALITY](#) for a description of the meaning and usage of the [QUALITY PARAMETERS](#).

- [POINT](#)

The API internally calls this low-level command when the [GetAsynchronousRecord](#) function call is issued. It is not directly accessible via the API. In response to the [POINT](#) command, the tracker immediately transmits one data record containing its last known tracking solution.

Note that a record containing data with the same timestamp from all sensors can be obtained by setting the sensor ID to ALL\_SENSORS.

- [STREAM](#)

The API internally calls this low-level command when the [GetSynchronousRecord](#) function call is issued. It is not directly accessible via the API. After the [STREAM](#) command is issued, the tracker begins sending continuous data records to the host PC (API) without waiting for the next data request, thus ensuring that each and every data record computed by the tracker is sent. While this prevents the occurrence of duplicate records (as can occur when calling the [GetAsynchronousRecord](#) faster than the tracker update rate), it does not guarantee that records are not overwritten. The buffer available to the system for each sensor is 8 records long. If the host application does not keep up with the constant stream of data being provided in this mode, this buffer will overflow and records will be lost.

Note that issuing commands (other than [GetSynchronousRecord](#)) that must query the unit for a response, will cause the unit to come out of STREAM mode.

Also note that hot-swapping sensors during STREAM mode operation will introduce delay in data for all sensor channels, as the unit must be taken out of this mode to detect and process info from the inserted sensor, then commanded to resume the STREAM mode operation.

The rate at which records are transmitted when using the `GetSynchronousRecord` can be changed through use of the `REPORT_RATE` system parameter. See the [Configurable Settings](#) section in Chapter 3 for details.

As with the `GetAsynchronous` call, a record containing data with the same timestamp from all sensors can be obtained by setting the sensor ID to `ALL_SENSORS`.

- **[SERIAL\\_NUMBER\\_RX](#)**

The sensor's serial number can be obtained by calling [GetSensorParameter](#).

## BOARD

Some specific information of interest to the user concerning the PCB hardware is available. Apart from this information there are no operations necessary or available for interacting directly with the PCB.

- **[SERIAL\\_NUMBER\\_PCB](#)**

The board's serial number can be obtained by calling [GetBoardParameter](#).

- **[BOARD\\_SOFTWARE\\_REVISIONS](#)**

The tracker's firmware version number is stored as a 2-digit revision number. You can access it by issuing the [GetBoardConfiguration](#) command for the specified board.

## TRANSMITTER

The following operations apply only to transmitters. `REFERENCE_FRAME` and `XYZ_REFERENCE_FRAME` are both used to set up the transmitters reference frame for all sensors using that transmitter. The reference frame must be set up for each transmitter separately and may be set up differently for each one. Upon power up the Reference Frame is initialized to 0,0,0 and the XYZ Reference Frame is disabled. Note: No transmitter is selected at power up.

- **[REFERENCE\\_FRAME](#)**

Defines new measurement reference frame. The new reference frame is provided as 3 angles describing the azimuth, elevation and roll angles. There is no offset component and the reference frame is still centered on the transmitter. This parameter is changed or examined by using the [SetTransmitterParameter](#) and [GetTransmitterParameter](#) function calls. See [REFERENCE\\_FRAME](#) for full description and details.

- **[XYZ\\_REFERENCE\\_FRAME](#)**

When the transmitter REFERENCE\_FRAME is changed, it will cause the azimuth, elevation and roll angles of all the sensors to change to a new reference frame but it will not cause the x, y and z position coordinates to change unless the XYZ Reference Frame flag is set. This flag is changed and examined with the [SetTransmitterParameter](#) and the [GetTransmitterParameter](#) function calls. See [XYZ REFERENCE FRAME](#) for full description and usage.

- **SERIAL NUMBER**

The transmitter's serial number can be obtained by using [GetTransmitterParameter](#)

- **NEXT TRANSMITTER**

This command allows us to turn on next transmitter or turn off the current transmitter in tracker. A full description of the operation is found at [SELECT\\_TRANSMITTER](#).

## System Initialization

The first operation that must be performed before the system can be used is initialization. Calling InitializeBIRDSystem performs this function. The call takes no parameters and returns no information except for a completion code. The only acceptable code is BIRD\_ERROR\_SUCCESS. All other codes are fatal errors that either indicates a condition that has prevented the system from initializing or they indicate a prevailing condition that disallows the system from completing the initialization. For example, the error code: BIRD\_ERROR\_COMMAND\_TIME\_OUT probably indicates a non-responding board. This is a hard failure. The error code BIRD\_ERROR\_INVALID\_DEVICE\_ID indicates that although the board is functional, initialization will not be allowed to proceed because the board is incompatible with the driver and API. The error codes are provided as a diagnostic and indicate a system condition that needs to be rectified before initialization can complete. Without a complete and successful initialization the system cannot be used.

Note: Initialization is an all-inclusive operation. Internally, the first task it performs is to enumerate your tracker's unique signature. Secondly, your electronics unit is queried for status and functionality. An internal database is then constructed of the current state of the system. The synchronization hardware is initialized and enabled.

The initialization may be invoked as follows:

```
#include "ATC3DGm.h"
.
.
.
int errorCode;
.
.
.
errorCode = InitializeBIRDSystem();

if(errorCode!=BIRD_ERROR_SUCCESS)
{
    // place error handler here
}
```

Note: In order to use any 3DGuidance API calls it is necessary to include the header file *ATC3DGm.h*. The returned value `errorCode` must be declared as a variable of type *int*.

Note: The application should terminate, as no further progress is possible without successful initialization. Calling any function except a `GetxxxStatus()` function before initialization has been performed will result in the function returning the error code `BIRD_ERROR_SYSTEM_UNINITIALIZED`. The response to a `GetxxxStatus()` call is for the `UNINITIALIZED` bit field to be set. The [GetErrorText](#) call is the only function that can be called at any time. (It may be used to decode the `BIRD_ERROR_SYSTEM_UNINITIALIZED` response and generate a message string.)

### ATC3DGm.ini File

The 3D Guidance API for medSAFE uses an .ini file, *ATC3DGm.ini*, during calls to [InitializeBIRDSYSTEM](#). The .ini file is automatically created in the same local directory as the API library, *ATC3DGm.dll*. The .ini file currently supports the following keys:

Autoconfig=4

`InitializeBIRDSYSTEM` tries to configure the tracker for the number of tracked objects specified by the `Autoconfig` key. For a Dipole transmitter, the tracker will disregard this request and configure for 4 tracked objects. This key should be changed to 12 for operation of multiheaded 5DOF sensors.

Logging=no

When the logging key is set to yes, the API will create a file, *LogFile.txt*, containing the bytes sent and received between the API and the tracker.

## System Setup

The system setup involves setting the sensor measurement rate, selecting the AGC mode, power line frequency and maximum range, setting the metric/English flag and turning on a transmitter. All of these operations are performed using the [SetSystemParameter](#) call. All parameters have a default value associated with them so unless the default is unsuitable the parameter need not be changed.

The following code fragment shows how all the parameters may be changed to a new value:

```
#include "ATC3DGm.h"          // needed for enumerated types and calls

int errorCode;

double pl = 50.0;              // 50 Hz
AGC_MODE_TYPE agc = SENSOR_AGC_ONLY; // tx power fixed at max
double rate = 86.1;           // 86.1 Hz
double range = 72.0;          // 72 inches
BOOL metric = true;           // metric reporting enabled
short tx = 0;                  // tx index number 0 selected

errorCode = SetSystemParameter(POWER_LINE_FREQUENCY, &pl, sizeof(pl));
if(errorCode != BIRD_ERROR_SUCCESS)
{
    // error handler
}

errorCode = SetSystemParameter(AGC_MODE, &agc, sizeof(agc));
if(errorCode != BIRD_ERROR_SUCCESS)
{
    // error handler
}

errorCode = SetSystemParameter(MEASUREMENT_RATE, &rate, sizeof(rate));
if(errorCode != BIRD_ERROR_SUCCESS)
{
    // error handler
}

errorCode = SetSystemParameter(MAXIMUM_RANGE, &range, sizeof(range));
if(errorCode != BIRD_ERROR_SUCCESS)
{
    // error handler
}

errorCode = SetSystemParameter(METRIC, &metric, sizeof(metric));
if(errorCode != BIRD_ERROR_SUCCESS)
{
    // error handler
}

errorCode = SetSystemParameter(SELECT_TRANSMITTER, &tx, sizeof(tx));
if(errorCode != BIRD_ERROR_SUCCESS)
{
    // error handler
}
```

An alternative approach is to use an exception handler for the error handler.

Another way to setup the system is to use the [RestoreSystemConfiguration](#). This together with the [SaveSystemConfiguration](#) call provide a convenient way for the user to save the current state of

the total system to an information file (.inf) and then use that file at a later time to re-initialize the system to that exact state. These calls allow the user to save or restore all settable parameters used by the system, sensors and transmitter. The following code fragment illustrates the usage of the [RestoreSystemConfiguration](#) call.

```
//
// Initialize system from ini file
//
errorCode = RestoreSystemConfiguration("oldconfig.ini");
if(errorCode!=BIRD_ERROR_SUCCESS) errorHandler(errorCode);
```

The system searches initially for the .ini file in the <Windows Directory>\inf directory. If it doesn't find it, it then looks for it in the <Windows Directory>\system32 directory unless the filename's path was fully specified. In the above example the system will search for "newfile.ini" first in the \inf directory then the \system32 directory. If not found an error will be generated. In the following sample the file will be looked for at the given location only.

```
errorCode = RestoreSystemConfiguration("c:\pcibird\oldconfig.ini");
if(errorCode!=BIRD_ERROR_SUCCESS) errorHandler(errorCode);
```

The simplest way to create a system configuration file is to let the system do it for you by using the [SaveSystemConfiguration](#) call. This call will create a file with the required format and including the current value for every system, sensor and transmitter parameter available. These files are saved as text files and can be edited using a text editor such as notepad.exe. See the section on configuration file format for details. The following code fragment shows how to save the current system configuration.

```
errorCode = SaveSystemConfiguration("c:\pcibird\newconfig.ini");
if(errorCode!=BIRD_ERROR_SUCCESS) errorHandler(errorCode);
```

The [RestoreSystemConfiguration](#) call is capable of setting every system, sensor and transmitter parameter available but it cannot and will not initialize the system. As with all other API, calls the [InitializeBIRDSystem](#) call must be made before [RestoreSystemConfiguration](#) can be used.



## Sensor Setup

The sensor setup involves selecting a data format, setting the filter and quality parameters, determining the sensor angle alignment and hemisphere of operation. All of these parameters have an associated default value. The parameter only needs to be changed if the default is inappropriate.

The default filter and quality parameters will be found to provide adequate performance for most applications. Unless the sensor is going to be attached to something that would cause it to be tilted while in its reference position then the angle align parameters will not need to be changed. The hemisphere will need to be changed if the sensor is going to operate anywhere other than the forward hemisphere that is the default. Typically the user will only have to set up the data format if something other than position/angles in double floating point is required. At a minimum nothing need be changed and the system will still operate successfully.

Note: It is necessary to set or change the parameter for each of the sensors individually as required. This allows each sensor to have its parameters set to different values.

The following code fragment gives an example of how to call the set parameter function in this case to set the data format to a double floating point value of position and matrix:

```
USHORT sensorID = 2;
int errorCode;
DATA_FORMAT_TYPE format = DOUBLE_POSITION_MATRIX;
errorCode = SetSensorParameter(
    sensorID,          // index number of target sensor
    DATA_FORMAT,      // command parameter type
    &format,           // address of data source buffer
    sizeof(format)     // size of source buffer
);
if(errorCode!=BIRD_ERROR_SUCCESS) errorHandler(errorCode);
// user must provide an error handler
```

The following code fragment shows how all the parameters may be changed to a new value. First a macro is defined which handles the different types of parameters which may be passed to the basic [SetSensorParameter](#) call.

```
#include "ATC3DGm.h"

////////////////////////////////////
////////////////////////////////////
//
// SET_SENSOR_PARAMETER macro
//
#define SET_SENSOR_PARAMETER(id, type, value) \
{ \
    type##_TYPE buf = value; \
    errorCode = SetSensorParameter(id, type, &buf, sizeof(buf)); \
    if(errorCode!=BIRD_ERROR_SUCCESS) errorHandler(errorCode); \
}

// In order for the above macro to compile without error it is
// necessary to provide typedefs for all the XXX_TYPES that are
// generated by "type##_TYPE"
```

```

// DATA_FORMAT_TYPE already defined as an enumerated type
typedef DOUBLE_ANGLES_RECORD      ANGLE_ALIGN_TYPE;
typedef DOUBLE_ANGLES_RECORD      REFERENCE_FRAME_TYPE;
typedef bool                      XYZ_REFERENCE_FRAME_TYPE;
// HEMISPHERE_TYPE already defined as an enumerated type
typedef bool                      FILTER_AC_WIDE_NOTCH_TYPE;
typedef bool                      FILTER_AC_NARROW_NOTCH_TYPE;
typedef double                    FILTER_DC_ADAPTIVE_TYPE;
typedef ADAPTIVE_PARAMETERS        FILTER_ALPHA_PARAMETERS_TYPE;
typedef bool                      FILTER_LARGE_CHANGE_TYPE;
typedef QUALITY_PARAMETERS         QUALITY_TYPE;

////////////////////////////////////
////////////////////////////////////
//
// Main program
//
int errorCode;
sensorID = 0;

SET_SENSOR_PARAMETER(sensorID, DATA_FORMAT, DOUBLE_POSITION_ANGLES_TIME_STAMP);

// initialize a structure of angles
DOUBLE_ANGLES_RECORD anglesRecord = {30, 45, 60};
SET_SENSOR_PARAMETER(sensorID, ANGLE_ALIGN, anglesRecord);

// initialize a structure of angles
DOUBLE_ANGLES_RECORD anglesRecord = {60, 45, 30};
SET_SENSOR_PARAMETER(sensorID, REFERENCE_FRAME, anglesRecord);
SET_SENSOR_PARAMETER(sensorID, XYZ_REFERENCE_FRAME, true);
SET_SENSOR_PARAMETER(sensorID, HEMISPHERE, TOP);
SET_SENSOR_PARAMETER(sensorID, FILTER_AC_WIDE_NOTCH, true);
SET_SENSOR_PARAMETER(sensorID, FILTER_AC_NARROW_NOTCH, false);
SET_SENSOR_PARAMETER(sensorID, FILTER_DC_ADAPTIVE, 1.0);

// initialize the alpha parameters
ADAPTIVE_PARAMETERS adaptiveRecord = {
    500, 500, 500, 500, 500, 500, 500,
    20000, 20000, 20000, 20000, 20000, 20000, 20000,
    2, 4, 8, 16, 32, 32,
    true
};
SET_SENSOR_PARAMETER(sensorID, FILTER_ALPHA_PARAMETERS, adaptiveRecord);
SET_SENSOR_PARAMETER(sensorID, FILTER_LARGE_CHANGE, false);

// initialize the quality parameter structure
QUALITY_PARAMETERS qualityParameters = { 15, 20, 16, 5 };
SET_SENSOR_PARAMETER(sensorID, QUALITY, qualityParameters);

```

## Transmitter Setup

The transmitter setup consists solely of setting up the transmitter reference frame. The default reference frame is (0, 0, 0) using Euler angles. The transmitter reference frame can only be changed by rotation there is no position offset available. The parameters only need to be changed if the default is inappropriate

Once set the transmitter reference frame will apply to all sensors. The reference frame setup is usually used to compensate for a transmitter whose installation results in it being tilted relative to the desired angular reference frame.

The following code fragment illustrates how to use the [SetTransmitterParameter](#) call to setup the transmitter reference frame.

```
USHORT transmitterID = 1;
int errorCode;
// e.g. a transmitter tilted at 45 degrees in elevation
DOUBLE ANGLES_RECORD frame = {0, 45, 0};
errorCode = SetTransmitterParameter(
    transmitterID,          // index number of target transmitter
    REFERENCE_FRAME,        // command parameter type
    &frame,                 // address of data source buffer
    sizeof(frame)           // size of source buffer
);
if(errorCode!=BIRD_ERROR_SUCCESS) errorHandler(errorCode);
// user must provide an error handler

// In this example we also want the sensor position to be
// corrected to compensate for the tilt in the transmitter
// So we set the XYZ_REFERENCE_FRAME parameter to "true"
// (Its default is "false")
BOOL xyz = true;
errorCode = SetTransmitterParameter(
    transmitterID,          // index number of target transmitter
    XYZ_REFERENCE_FRAME,    // command parameter type
    &xyz,                   // address of data source buffer
    sizeof(xyz)             // size of source buffer
);
if(errorCode!=BIRD_ERROR_SUCCESS) errorHandler(errorCode);
// user must provide an error handler
```

## Acquiring Tracking Data

Data is acquired by making calls to [GetAsynchronousRecord\(\)](#) or [GetSynchronousRecord\(\)](#) for each sensor that data is required from. Before calling either function it is necessary to initialize the system, transmitters and sensors to their desired settings. It is possible to acquire data with every setting left in its default state with the exception of [SELECT\\_TRANSMITTER](#). The `SYSTEM_PARAMETER_TYPE`, [SELECT\\_TRANSMITTER](#) is set to (-1) on initialization. This means that no transmitter has been selected. The minimum system setup required before data can be selected is to call [SetSystemParameter](#) with the [SELECT\\_TRANSMITTER](#) parameter and pass the id of the transmitter that is required to be turned on. The following code fragment illustrates a minimum requirement for acquiring data. It assumes that there is a transmitter attached to `id = 0` and that there is a sensor attached to `id = 0`.

```

////////////////////////////////////////
//
// First initialize the system
//
int errorCode = InitializeBIRDSystem();
if(errorCode!=BIRD_ERROR_SUCCESS)
{
    errorHandler(errorCode); // user supplied error handler
}

////////////////////////////////////////
//
// Turn on the transmitter.
// We turn on the transmitter by selecting the
// transmitter using its ID
//
USHORT id = 0;
errorCode = SetSystemParameter(SELECT_TRANSMITTER, &id, sizeof(id));
if(errorCode!=BIRD_ERROR_SUCCESS)
{
    errorHandler(errorCode);
}

////////////////////////////////////////
//
// Get a record from sensor #0.
// The default record type is DOUBLE_POSITION_ANGLES
//
USHORT sensorID = 0;
DOUBLE_POSITION_ANGLES_RECORD record;
errorCode = GetAsynchronousRecord(sensorID, &record, sizeof(record));
if(errorCode!=BIRD_ERROR_SUCCESS)
{
    errorHandler(errorCode);
}

```



**Tip:**  
Setting the  
SensorID to  
`ALL_SENSORS`  
will return data  
records from all  
sensors.

## Error Handling

Each call to the API will return either an error code or a status code depending on the command issued. Most commands will respond with an error code. The only commands that return a status code are the [GetSystemStatus](#), [GetBoardStatus](#), [GetSensorStatus](#) and [GetTransmitterStatus](#) commands.

It can be assumed that all error codes are fatal. In other words, the only acceptable response to a command is BIRD\_ERROR\_SUCCESS. If any other response is received then the command failed to complete and the error code will inform the user of the reason why it failed. The function [GetErrorText](#) can be used to generate a message string for output to a file or screen display describing in English the nature of the error code passed to this command. [GetErrorText](#) is the only command that does not require the Tracker system to be initialized before it can be used.

Even though all error codes indicate a fatal error condition, it is possible for the software to recover from some failures. For example if the system has not been initialized then the error code BIRD\_ERROR\_SYSTEM\_UNINITIALIZED will be returned. The software could recover by calling InitializeBIRDSystem() before doing anything else. But this error is usually an indication of a software “bug”. Other errors like BIRD\_ERROR\_NO\_SENSOR\_ATTACHED can be recovered from by displaying a message to the user suggesting that they attach a sensor to the system.

The status code returned by the GetXXXStatus() commands gives a bit-mapped indication of any error conditions that might exist for the device selected. If the status code returned = 0 then the device is fully operational. Any status other than 0 indicates an error that will prevent successful operation of the device. For example if a call is made to [GetSensorStatus](#) for a sensor channel whose sensor is not attached then the returned status will be 0x00000003, indicating that the NOT\_ATTACHED and the GLOBAL\_ERROR bits are set.

## 3DGuidance API

The following elements define the API used with the your medSAFE tracker.

[3D Guidance API Functions](#)

[3D Guidance API Structures](#)

[3D GuidanceAPI Enumeration Types](#)

[3D GuidanceAPI Status/Error Bit Definitions](#)

[3D GuidanceInitialization Files](#)

## 3DGuidance API Functions

The following functions are used with your medSAFE tracker.

[InitializeBIRDSystem](#)

[GetBIRDSystemConfiguration](#)

[GetTransmitterConfiguration](#)

[GetSensorConfiguration](#)

[GetBoardConfiguration](#)

[GetSystemParameter](#)

[GetSensorParameter](#)

[GetTransmitterParameter](#)

[GetBoardParameter](#)

[SetSystemParameter](#)

[SetSensorParameter](#)

[SetTransmitterParameter](#)

[SetBoardParameter](#)

[GetAsynchronousRecord](#)

[GetSynchronousRecord](#)

[GetBIRDError](#)

[GetErrorText](#)

[GetSensorStatus](#)

[GetTransmitterStatus](#)

[GetBoardStatus](#)

[GetSystemStatus](#)

[SaveSystemConfiguration](#)

[RestoreSystemConfiguration](#)

[CloseBIRDSystem](#)

## InitializeBIRDSystem

The **InitializeBIRDSystem** function resets and initializes the medSAFE hardware and system.

```
int InitializeBIRDSystem();
```

### Parameters

This function takes no parameters

### Return Values

The function returns a value of type *int*. This value takes the form of an [ERRORCODE](#) indicating success or failure for the call. The enumerated error code field contained within the 32-bit [ERRORCODE](#) may have one of the following values for this function call:

Value	Meaning
BIRD_ERROR_SUCCESS	No errors occurred. Initialization completed successfully
BIRD_ERROR_INCORRECT_DRIVER_VERSION	The wrong version of the driver has been installed for this version of the API dll. Install or re-install the correct driver.
BIRD_ERROR_OPENING_DRIVER	Non-specific error opening driver. Make sure that the driver is properly installed.
BIRD_ERROR_NO_DEVICES_FOUND	No Tracker hardware was found by the host system. Verify that hardware is installed and is of the correct type.
BIRD_ERROR_ACCESSING_PCI_CONFIG	The error occurred in the PCIBird PCI interface. There is a problem with the PCI configuration registers. If error is repeatable there is an unrecoverable hardware failure.
BIRD_ERROR_INVALID_DEVICE_ID	A Tracker device has been found that is not supported by this API dll. Verify Tracker model installed.
BIRD_ERROR_FAILED_LOCKING_DEVICE	Driver could not lock Tracker resources. Check that there is not another application using the hardware.
BIRD_ERROR_BOARD_MISSING_ITEMS	The required resources were not found defined in the PCI configuration registers. Possible corrupt configuration. If error is repeatable there is an unrecoverable hardware failure.
BIRD_ERROR_INCORRECT_PLD	The PLD version on the Tracker hardware is incompatible with this version of the API dll. Verify Tracker model installed.
BIRD_ERROR_COMMAND_TIME_OUT	Tracker on-board controller has failed to respond to a command issued to it. If error is repeatable there is an unrecoverable hardware failure.
BIRD_ERROR_WATCHDOG	Tracker internal watchdog timer has elapsed. If this error is repeatable there is an unrecoverable hardware failure.
BIRD_ERROR_INCORRECT_BOARD_DEFAULT	An unexpected response was received from the controller on the Tracker hardware. The board is responding to commands but the data returned is corrupt. If the error is repeatable there is an unrecoverable hardware failure.
BIRD_ERROR_PCB_HARDWARE_FAILURE	The Tracker firmware initialization did not complete within



	10 seconds. It is assumed the board is faulty or the firmware has hung up somewhere. If the error is repeatable there is an unrecoverable hardware failure.
BIRD_ERROR_UNRECOGNIZED_MODEL_STRING	The firmware is reporting a model string that is unrecognized by the API dll. This could be due to a hardware failure causing the model string data to be corrupted or a corrupted board EEPROM may cause it or the board installed is of a type not recognized by the API dll. If the error is repeatable return to vendor.

**Remarks**

When this function is called, it will first reset the tracker. The function will then interrogate the boards and determine their status. Finally, it will build a database of tracker information containing number of sensors, transmitters etc. This function takes several seconds to complete because it has to wait for the boards to reset and initialize internally. This function must be called first, before any other command can be sent

**Requirements**

**Windows NT/2000:** Requires Windows 2000 or later.

**Header:** Declared in ATC3DGm.h

**Library:** Use ATC3DGm.lib

**See Also**

## GetBIRDSystemConfiguration

The **GetBIRDSystemConfiguration** will return a structure containing the [SYSTEM\\_CONFIGURATION](#).

```
int GetBIRDSystemConfiguration(
    SYSTEM_CONFIGURATION* pSystemConfiguration
);
```

### Parameters

*pSystemConfiguration*

[out] Pointer to a [SYSTEM\\_CONFIGURATION](#) structure that receives the information about the system.

### Return Values

The function returns a value of type *int*. This value takes the form of an [ERRORCODE](#) indicating success or failure for the call. The enumerated error code field contained within the 32-bit [ERRORCODE](#) may have one of the following values for this function call:

Value	Meaning
BIRD_ERROR_SUCCESS	No errors occurred. Call completed successfully
BIRD_ERROR_SYSTEM_UNINITIALIZED	The Tracker hardware and system has not been initialized yet. The <a href="#">InitializeBIRDSystem</a> function must be called first.

### Remarks

This function passes a single parameter that is a pointer to a structure, which will hold the system configuration on return from the call. The structure contains variables that give the number of sensors, transmitters and boards in the system. These numbers can then be used to allocate storage for arrays of structures to store the sensor and transmitter configurations. The board configurations may be used to monitor the hardware configuration of the system.

The structure also contains the current measurement rate, line frequency, maximum range and AGC mode of the system when the configuration was returned. These parameters effect operation in a system-wide manner.

### Requirements

**Windows NT/2000:** Requires Windows 2000 or later.

**Header:** Declared in ATC3DGm.h

**Library:** Use ATC3DGm.lib

### See Also

## GetTransmitterConfiguration

The **GetTransmitterConfiguration** will return a structure containing a [TRANSMITTER\\_CONFIGURATION](#).

```
int GetTransmitterConfiguration(
    USHORT transmitterID,
    TRANSMITTER_CONFIGURATION* pTransmitterConfiguration
);
```

### Parameters

*transmitterID*

[in] The transmitterID is in the range 0..(n-1) where n is the number of possible transmitters in the system.

*pTransmitterConfiguration*

[out] Pointer to a [TRANSMITTER\\_CONFIGURATION](#) structure that receives the information about the transmitter.

### Return Values

The function returns a value of type *int*. This value takes the form of an [ERRORCODE](#) indicating success or failure for the call. The enumerated error code field contained within the 32-bit [ERRORCODE](#) may have one of the following values for this function call:

Value	Meaning
BIRD_ERROR_SUCCESS	No errors occurred. Call completed successfully
BIRD_ERROR_SYSTEM_UNINITIALIZED	The Tracker hardware and system has not been initialized yet. The <a href="#">InitializeBIRDSystem</a> function must be called first.
BIRD_ERROR_INVALID_DEVICE_ID	The transmitterID passed was out of range for the system.

### Remarks

This function takes as its parameters an index to a specific transmitter and a pointer to a structure that is used to return the transmitter configuration information.

The index number is in the range 0..(n-1) where n is the number of possible transmitters in the system.

The transmitter configuration returned contains most importantly the serial number of any transmitter attached at the specified ID. This is the most reliable way to correlate an actual physical transmitter and its index number. The other information provided is the index number of the board where the transmitter is found, and the channel number within that board. The transmitter type is also provided.

### Requirements

**Windows NT/2000:** Requires Windows 2000 or later.

**Header:** Declared in ATC3DGm.h

**Library:** Use ATC3DGm.lib

### See Also

## GetSensorConfiguration

The **GetSensorConfiguration** will return a structure containing a [SENSOR\\_CONFIGURATION](#).

```
int GetSensorConfiguration(
    USHORT sensorID,
    SENSOR_CONFIGURATION* pTransmitterConfiguration
);
```

### Parameters

*sensorID*

[in] The sensorID is in the range 0..(n-1) where n is the number of possible sensors in the system.

*pSensorConfiguration*

[out] Pointer to a [SENSOR\\_CONFIGURATION](#) structure that receives the information about the sensor.

### Return Values

The function returns a value of type *int*. This value takes the form of an [ERRORCODE](#) indicating success or failure for the call. The enumerated error code field contained within the 32-bit [ERRORCODE](#) may have one of the following values for this function call:

Value	Meaning
BIRD_ERROR_SUCCESS	No errors occurred. Call completed successfully
BIRD_ERROR_SYSTEM_UNINITIALIZED	The Tracker hardware and system has not been initialized yet. The <a href="#">InitializeBIRDSystem</a> function must be called first.
BIRD_ERROR_INVALID_DEVICE_ID	The sensorID passed was out of range for the system.

### Remarks

This function takes as its parameters an index to a specific sensor and a pointer to a structure that is used to return the sensor configuration information.

The index number is in the range 0..(n-1) where n is the number of possible sensors in the system.

The sensor configuration returned contains most importantly the serial number of any sensor attached at the specified ID. This is the most reliable way to correlate an actual physical sensor and its index number. The other information provided is the index number of the board where the sensor is found, and the channel number within that board. The sensor type is also provided.

### Requirements

**Windows NT/2000:** Requires Windows 2000 or later.

**Header:** Declared in ATC3DGm.h

**Library:** Use ATC3DGm.lib

### See Also

## GetBoardConfiguration

The **GetBoardConfiguration** will return a structure containing a [BOARD\\_CONFIGURATION](#).

```
int GetBoardConfiguration(
    USHORT boardID,
    BOARD_CONFIGURATION* pBoardConfiguration
);
```

### Parameters

*boardID*

[in] The boardID is in the range 0..(n-1) where n is the number of possible boards in the system.

*pBoardConfiguration*

[out] Pointer to a [BOARD\\_CONFIGURATION](#) structure that receives the information about the board.

### Return Values

The function returns a value of type *int*. This value takes the form of an [ERRORCODE](#) indicating success or failure for the call. The enumerated error code field contained within the 32-bit [ERRORCODE](#) may have one of the following values for this function call:

Value	Meaning
BIRD_ERROR_SUCCESS	No errors occurred. Call completed successfully
BIRD_ERROR_SYSTEM_UNINITIALIZED	The Tracker hardware and system has not been initialized yet. The <a href="#">InitializeBIRDSystem</a> function must be called first.
BIRD_ERROR_INVALID_DEVICE_ID	The boardID passed was out of range for the system.

### Remarks

This function takes as its parameters an index to a specific board and a pointer to a structure that is used to return the board configuration information.

The index number is in the range 0..(n-1) where n is the number of possible boards in the system.

This function returns a structure slightly different from the [SENSOR\\_CONFIGURATION](#) and [TRANSMITTER\\_CONFIGURATION](#) structures. The [BOARD\\_CONFIGURATION](#) returned with this call provides the number of sensor and transmitter connectors available on this board. It also provides the revision number of the firmware running on the board.

### Requirements

**Windows NT/2000:** Requires Windows 2000 or later.

**Header:** Declared in ATC3DGm.h

**Library:** Use ATC3DGm.lib

### See Also

## GetSystemParameter

The **GetSystemParameter** will return a buffer containing the selected parameter values(s).

```
int GetSystemParameter(
    SYSTEM_PARAMETER_TYPE parameterType,
    Void* pBuffer,
    Int bufferSize
);
```

### Parameters

#### *parameterType*

[in] Contains the parameter type to be returned in the buffer. Must be a valid enumerated constant of the type [SYSTEM\\_PARAMETER\\_TYPE](#).

#### *pBuffer*

[out] Points to a buffer to be used for returning the information about the [SYSTEM\\_PARAMETER\\_TYPE](#) being queried. WARNING: The size of the buffer must be equal to or greater than the size of the parameter being returned or the function may overwrite user memory.

#### *bufferSize*

[in] Contains the size of the buffer whose address is passed in *pBuffer*. Note the *bufferSize* value must match the size of the returned parameter exactly.

### Return Values

The function returns a value of type *int*. This value takes the form of an [ERRORCODE](#) indicating success or failure for the call. The enumerated error code field contained within the 32-bit [ERRORCODE](#) may have one of the following values for this function call:

Value	Meaning
BIRD_ERROR_SUCCESS	No errors occurred. Call completed successfully
BIRD_ERROR_SYSTEM_UNINITIALIZED	The Tracker hardware and system has not been initialized yet. The <a href="#">InitializeBIRDSystem</a> function must be called first.
BIRD_ERROR_INCORRECT_PARAMETER_SIZE	The value of the <i>bufferSize</i> parameter passed did not match the size of the parameter being returned.
BIRD_ERROR_ILLEGAL_COMMAND_PARAMETER	Invalid enumerated constant of type <a href="#">SYSTEM_PARAMETER_TYPE</a> used.

### Remarks

The GetSystemParameter and SetSystemParameter commands are designed to allow access to and manipulation of parameters that effect the computation cycle and algorithm. These include measurement rate, AGC mode, Power line frequency etc. Note that some of the parameters take as values other enumerated types.

### Requirements

**Windows NT/2000:** Requires Windows 2000 or later.

**Header:** Declared in ATC3DGm.h

**Library:** Use ATC3DGm.lib

### See Also

[SetSystemParameter](#)

## GetSensorParameter

The **GetSensorParameter** function will return a buffer containing the selected parameter values(s).

```
int GetSensorParameter(
    USHORT sensorID
    SENSOR_PARAMETER_TYPE parameterType,
    Void* pBuffer,
    Int bufferSize
);
```

### Parameters

*sensorID*

[in] Valid SensorIDs are in the range 0..(n-1) where n is the number of sensors in the system.

*parameterType*

[in] Contains the parameter type to be returned in the buffer. Must be a valid enumerated constant of the type SENSOR\_PARAMETER\_TYPE.

*pBuffer*

[out] Points to a buffer to be used for returning the information about the [SENSOR\\_PARAMETER\\_TYPE](#) being queried. WARNING: The size of the buffer must be equal to or greater then the size of the parameter being returned or the function may overwrite user memory.

*bufferSize*

[in] Contains the size of the buffer whose address is passed in *pBuffer*. Note the *bufferSize* value must match the size of the returned parameter exactly.

### Return Values

The function returns a value of type *int*. This value takes the form of an [ERRORCODE](#) indicating success or failure for the call. The enumerated error code field contained within the 32-bit [ERRORCODE](#) may have one of the following values for this function call:

Value	Meaning
BIRD_ERROR_SUCCESS	No errors occurred. Call completed successfully
BIRD_ERROR_SYSTEM_UNINITIALIZED	The Tracker hardware and system has not been initialized yet. The <a href="#">InitializeBIRDSystem</a> function must be called first.
BIRD_ERROR_INCORRECT_PARAMETER_SIZE	The value of the <i>bufferSize</i> parameter passed did not match the size of the parameter being returned.
BIRD_ERROR_ILLEGAL_COMMAND_PARAMETER	Invalid enumerated constant of type SENSOR_PARAMETER_TYPE used.
BIRD_ERROR_INVALID_DEVICE_ID	The sensorID passed was out of range for the system.

### Remarks

The GetSensorParameter command is designed to allow the viewing of parameters that effect the computation cycle and algorithm for a single sensor. The command differs from the system command in that it requires a device ID to

indicate which sensor is being referred to. See [SENSOR\\_PARAMETER\\_TYPE](#) for a description of the individual parameters.

**Requirements**

**Windows NT/2000:** Requires Windows 2000 or later.

**Header:** Declared in ATC3DGm.h

**Library:** Use ATC3DGm.lib

**See Also**

[SetSensorParameter](#)



## GetTransmitterParameter

The **GetTransmitterParameter** function will return a buffer containing the selected parameter values(s).

```
int GetTransmitterParameter(
    USHORT transmitterID
    TRANSMITTER_PARAMETER_TYPE parameterType,
    void* pBuffer,
    int bufferSize
);
```

### Parameters

#### *transmitterID*

[in] Valid transmitterIDs are in the range 0..(n-1) where n is the number of transmitters in the system.

#### *parameterType*

[in] Contains the parameter type to be returned in the buffer. Must be a valid enumerated constant of the type TRANSMITTER\_PARAMETER\_TYPE.

#### *pBuffer*

[out] Points to a buffer to be used for returning the information about the [TRANSMITTER\\_PARAMETER\\_TYPE](#) being queried. WARNING: The size of the buffer must be equal to or greater then the size of the parameter being returned or the function may overwrite user memory.

#### *bufferSize*

[in] Contains the size of the buffer whose address is passed in *pBuffer*. Note the *bufferSize* value must match the size of the returned parameter exactly.

### Return Values

The function returns a value of type *int*. This value takes the form of an [ERRORCODE](#) indicating success or failure for the call. The enumerated error code field contained within the 32-bit [ERRORCODE](#) may have one of the following values for this function call:

Value	Meaning
BIRD_ERROR_SUCCESS	No errors occurred. Call completed successfully
BIRD_ERROR_SYSTEM_UNINITIALIZED	The Tracker hardware and system has not been initialized yet. The <a href="#">InitializeBIRDSystem</a> function must be called first.
BIRD_ERROR_INCORRECT_PARAMETER_SIZE	The value of the <i>bufferSize</i> parameter passed did not match the size of the parameter being returned.
BIRD_ERROR_ILLEGAL_COMMAND_PARAMETER	Invalid enumerated constant of type TRANSMITTER_PARAMETER_TYPE used.
BIRD_ERROR_INVALID_DEVICE_ID	The transmitterID passed was out of range for the system.

### Remarks

The GetTransmitterParameter command is designed to allow the viewing of parameters that effect the operation of a single transmitter. The command differs from the system command in that it requires a device ID to indicate which transmitter is being referred to. See [TRANSMITTER\\_PARAMETER\\_TYPE](#) for a description of the individual parameters.

**Requirements**

**Windows NT/2000:** Requires Windows 2000 or later.

**Header:** Declared in ATC3DGm.h

**Library:** Use ATC3DGm.lib

**See Also**

[SetTransmitterParameter](#)

## GetBoardParameter

The **GetBoardParameter** function will return a buffer containing the selected parameter values(s).

```
int GetBoardParameter(
    USHORT boardID
    BOARD_PARAMETER_TYPE parameterType,
    void* pBuffer,
    int bufferSize
);
```

### Parameters

*boardID*

[in] Valid boardIDs are in the range 0..(n-1) where n is the number of boards in the system.

*parameterType*

[in] Contains the parameter type to be returned in the buffer. Must be a valid enumerated constant of the type BOARD\_PARAMETER\_TYPE.

*pBuffer*

[out] Points to a buffer to be used for returning the information about the BOARD\_PARAMETER\_TYPE being queried. WARNING: The size of the buffer must be equal to or greater than the size of the parameter being returned or the function may overwrite user memory.

*bufferSize*

[in] Contains the size of the buffer whose address is passed in *pBuffer*. Note the *bufferSize* value must match the size of the returned parameter exactly.

### Return Values

The function returns a value of type *int*. This value takes the form of an [ERRORCODE](#) indicating success or failure for the call. The enumerated error code field contained within the 32-bit [ERRORCODE](#) may have one of the following values for this function call:

Value	Meaning
BIRD_ERROR_SUCCESS	No errors occurred. Call completed successfully
BIRD_ERROR_SYSTEM_UNINITIALIZED	The Tracker hardware and system has not been initialized yet. The <a href="#">InitializeBIRDSystem</a> function must be called first.
BIRD_ERROR_INCORRECT_PARAMETER_SIZE	The value of the <i>bufferSize</i> parameter passed did not match the size of the parameter being returned.
BIRD_ERROR_ILLEGAL_COMMAND_PARAMETER	Invalid enumerated constant of type BOARD_PARAMETER_TYPE used.
BIRD_ERROR_INVALID_DEVICE_ID	The boardID passed was out of range for the system.

### Remarks

The GetBoardParameter command is designed to allow the viewing of parameters that effect the operation of a single board. The command differs from the system command in that it requires a board ID to indicate which board is being referred to. See BOARD\_PARAMETER\_TYPE for a description of the individual parameters.

**Requirements**

**Windows NT/2000:** Requires Windows 2000 or later.

**Header:** Declared in ATC3DGm.h

**Library:** Use ATC3DGm.lib

**See Also**

[SetBoardParameter](#)

## SetSystemParameter

The **SetSystemParameter** function allows an application to change basic Tracker system parameters.

```
int SetSystemParameter(
    SYSTEM_PARAMETER_TYPE parameterType,
    void* pBuffer,
    int bufferSize
);
```

### Parameters

#### *parameterType*

[in] Contains the parameter type to be passed in the buffer. Must be a valid enumerated constant of the type `SYSTEM_PARAMETER_TYPE`.

#### *pBuffer*

[in] Points to a buffer to be used for passing the information about the [SYSTEM\\_PARAMETER\\_TYPE](#) being changed. WARNING: The size of the buffer must be equal to or greater than the size of the parameter being passed or the function will read beyond the end of the buffer into user memory with indeterminate results.

#### *bufferSize*

[in] Contains the size of the buffer whose address is passed in *pBuffer*. Note the *bufferSize* value must match the size exactly of the parameter being passed in the buffer.

### Return Values

The function returns a value of type *int*. This value takes the form of an [ERRORCODE](#) indicating success or failure for the call. The enumerated error code field contained within the 32-bit [ERRORCODE](#) may have one of the following values for this function call:

Value	Meaning
BIRD_ERROR_SUCCESS	No errors occurred. Call completed successfully
BIRD_ERROR_SYSTEM_UNINITIALIZED	The Tracker hardware and system has not been initialized yet. The <a href="#">InitializeBIRDSysyem</a> function must be called first.
BIRD_ERROR_INCORRECT_PARAMETER_SIZE	The value of the <i>bufferSize</i> parameter passed did not match the size of the parameter being returned.
BIRD_ERROR_ILLEGAL_COMMAND_PARAMETER	Invalid enumerated constant of type <code>SYSTEM_PARAMETER_TYPE</code> used.
BIRD_ERROR_NO_TRANSMITTER_RUNNING	A request was made to turn off the current transmitter by passing the value -1 with the parameter <code>SELECT_TRANSMITTER</code> selected and there was no transmitter currently running.
BIRD_ERROR_NO_TRANSMITTER_ATTACHED	A request was made to do one of the following: 1) Turn off the currently running transmitter and there is no transmitter attached to the system 2) Turn on the transmitter with the selected ID and there is no transmitter attached at that ID.
BIRD_ERROR_COMMAND_TIME_OUT	Tracker on-board controller has failed to respond to a

	command issued to it. If error is repeatable there is an unrecoverable hardware failure.
BIRD_ERROR_WATCHDOG	Tracker internal watchdog timer has elapsed. If this error is repeatable there is an unrecoverable hardware failure.

### Remarks

The [GetSystemParameter](#) and [SetSystemParameter](#) commands are designed to allow access to and manipulation of parameters that effect the computation cycle and algorithm. These include measurement rate, AGC mode, Power line frequency etc. Note that some of the parameters take as values other enumerated types.

### Requirements

**Windows NT/2000:** Requires Windows 2000 or later.

**Header:** Declared in ATC3DGm.h

**Library:** Use ATC3DGm.lib

### See Also

[GetSystemParameter](#)

## SetSensorParameter

The **SetSensorParameter** function allows an application to select and change specific characteristics of individual sensors in a tracking system.

```
int SetSensorParameter(
    USHORT sensorID
    SENSOR_PARAMETER_TYPE parameterType,
    void* pBuffer,
    int bufferSize
);
```

### Parameters

#### *sensorID*

[in] Valid sensorIDs are in the range 0..(n-1) where n is the number of sensors in the system.

#### *parameterType*

[in] Contains the parameter type whose new value is being passed in the buffer. It must be a valid enumerated constant of the type `SENSOR_PARAMETER_TYPE`.

#### *pBuffer*

[in] Points to a buffer to be used for passing the new parameter information of the [SENSOR\\_PARAMETER\\_TYPE](#) being changed. WARNING: The size of the buffer must be equal to or greater then the size of the parameter being passed or the function will read beyond the end of the buffer into user memory with indeterminate results.

#### *bufferSize*

[in] Contains the size of the buffer whose address is passed in *pBuffer*. Note the *bufferSize* value must match the size of the passed parameter exactly.

### Return Values

The function returns a value of type *int*. This value takes the form of an [ERRORCODE](#) indicating success or failure for the call. The enumerated error code field contained within the 32-bit [ERRORCODE](#) may have one of the following values for this function call:

Value	Meaning
BIRD_ERROR_SUCCESS	No errors occurred. Call completed successfully
BIRD_ERROR_SYSTEM_UNINITIALIZED	The Tracker hardware and system has not been initialized yet. The <a href="#">InitializeBIRDSystem</a> function must be called first.
BIRD_ERROR_INCORRECT_PARAMETER_SIZE	The value of the <i>bufferSize</i> parameter passed did not match the size of the parameter being returned.
BIRD_ERROR_ILLEGAL_COMMAND_PARAMETER	Invalid enumerated constant of type <code>SENSOR_PARAMETER_TYPE</code> used.
BIRD_ERROR_INVALID_DEVICE_ID	The sensorID passed was out of range for the system.
BIRD_ERROR_COMMAND_TIME_OUT	Tracker on-board controller has failed to respond to a command issued to it. If error is repeatable there is an unrecoverable hardware failure.

BIRD_ERROR_WATCHDOG	Tracker internal watchdog timer has elapsed. If this error is repeatable there is an unrecoverable hardware failure.
---------------------	--

**Remarks**

The SetSensorParameter command is designed to allow the manipulation of parameters that effect the computation cycle and algorithm for a single sensor. The command differs from the system command in that it requires a device ID to indicate which sensor is being referred to. See [SENSOR\\_PARAMETER\\_TYPE](#) for a description of the individual parameters.

**Requirements**

**Windows NT/2000:** Requires Windows 2000 or later.

**Header:** Declared in ATC3DGm.h

**Library:** Use ATC3DGm.lib

**See Also**

[GetSensorParameter](#)



## SetTransmitterParameter

The **SetTransmitterParameter** function allows an application to change specific operational characteristics of individual transmitters.

```
int SetTransmitterParameter(
    USHORT transmitterID
    TRANSMITTER_PARAMETER_TYPE parameterType,
    void* pBuffer,
    int bufferSize
);
```

### Parameters

*transmitterID*

[in] Valid transmitterIDs are in the range 0..(n-1) where n is the number of transmitters in the system.

*parameterType*

[in] Contains the parameter type to be passed in the buffer. Must be a valid enumerated constant of the type TRANSMITTER\_PARAMETER\_TYPE.

*pBuffer*

[in] Points to a buffer to be used for passing the information about the [TRANSMITTER\\_PARAMETER\\_TYPE](#) being changed. WARNING: The size of the buffer must be equal to or greater then the size of the parameter being passed or the function may read beyond the end of the buffer into user memory with indeterminate results.

*bufferSize*

[in] Contains the size of the buffer whose address is passed in *pBuffer*. Note the *bufferSize* value must match the size of the passed parameter exactly.

### Return Values

The function returns a value of type *int*. This value takes the form of an [ERRORCODE](#) indicating success or failure for the call. The enumerated error code field contained within the 32-bit [ERRORCODE](#) may have one of the following values for this function call:

Value	Meaning
BIRD_ERROR_SUCCESS	No errors occurred. Call completed successfully
BIRD_ERROR_SYSTEM_UNINITIALIZED	The Tracker hardware and system has not been initialized yet. The <a href="#">InitializeBIRDSysyem</a> function must be called first.
BIRD_ERROR_INCORRECT_PARAMETER_SIZE	The value of the <i>bufferSize</i> parameter passed did not match the size of the parameter being returned.
BIRD_ERROR_ILLEGAL_COMMAND_PARAMETER	Invalid enumerated constant of type TRANSMITTER_PARAMETER_TYPE used.
BIRD_ERROR_INVALID_DEVICE_ID	The transmitterID passed was out of range for the system.
BIRD_ERROR_COMMAND_TIME_OUT	Tracker on-board controller has failed to respond to a

	command issued to it. If error is repeatable there is an unrecoverable hardware failure.
BIRD_ERROR_WATCHDOG	Tracker internal watchdog timer has elapsed. If this error is repeatable there is an unrecoverable hardware failure.

### Remarks

The SetTransmitterParameter command is designed to allow the manipulation of parameters that effect the operation of a single transmitter. The command differs from the system command in that it requires a device ID to indicate which transmitter is being referred to. See [TRANSMITTER\\_PARAMETER\\_TYPE](#) for a description of the individual parameters.

### Requirements

**Windows NT/2000:** Requires Windows 2000 or later.

**Header:** Declared in ATC3DGm.h

**Library:** Use ATC3DGm.lib

### See Also

[GetTransmitterParameter](#)

## SetBoardParameter

The **SetBoardParameter** function takes as a parameter a pointer to a buffer containing the selected parameter values(s) to be changed.

```
int SetBoardParameter(
    USHORT boardID
    BOARD_PARAMETER_TYPE parameterType,
    void* pBuffer,
    int bufferSize
);
```

### Parameters

*boardID*

[in] Valid boardIDs are in the range 0..(n-1) where n is the number of boards in the system.

*parameterType*

[in] Contains the parameter type to be passed in the buffer. Must be a valid enumerated constant of the type BOARD\_PARAMETER\_TYPE.

*pBuffer*

[out] Points to a buffer containing the information about the [BOARD\\_PARAMETER\\_TYPE](#) being changed.

WARNING: The size of the buffer must be equal to or greater then the size of the parameter being modified or the function may attempt to read from user memory.

*bufferSize*

[in] Contains the size of the buffer whose address is passed in *pBuffer*. Note the *bufferSize* value must match the size of the returned parameter exactly.

### Return Values

The function returns a value of type *int*. This value takes the form of an [ERRORCODE](#) indicating success or failure for the call. The enumerated error code field contained within the 32-bit [ERRORCODE](#) may have one of the following values for this function call:

Value	Meaning
BIRD_ERROR_SUCCESS	No errors occurred. Call completed successfully
BIRD_ERROR_SYSTEM_UNINITIALIZED	The Tracker hardware and system has not been initialized yet. The <a href="#">InitializeBIRDSystem</a> function must be called first.
BIRD_ERROR_INCORRECT_PARAMETER_SIZE	The value of the <i>bufferSize</i> parameter passed did not match the size of the parameter being returned.
BIRD_ERROR_ILLEGAL_COMMAND_PARAMETER	Invalid enumerated constant of type BOARD_PARAMETER_TYPE used.
BIRD_ERROR_INVALID_DEVICE_ID	The boardID passed was out of range for the system.

### Remarks

The GetBoardParameter command is designed to allow the changing of parameters that effect the operation of a single board. The command differs from the system command in that it requires a board ID to indicate which board is being referred to. See [BOARD\\_PARAMETER\\_TYPE](#) for a description of the individual parameters.

**Requirements**

**Windows NT/2000:** Requires Windows 2000 or later.

**Header:** Declared in ATC3DGm.h

**Library:** Use ATC3DGm.lib

**See Also**

[GetBoardParameter](#)

## GetAsynchronousRecord

The **GetAsynchronousRecord** function allows an application to acquire a position and orientation data record from an individual sensor.

```
int GetAsynchronousRecord(
    USHORT sensorID
    void* pRecord,
    int recordSize
);
```

### Parameters

#### *sensorID*

[in] Valid sensorIDs for an individual sensor are in the range 0..(n-1) where n is the number of sensors in the system. A sensorID value of ALL\_SENSORS (-1), is used to request records from all possible sensors. Note that using the ALL\_SENSORS sensorID will result in data records in the specified buffer for both attached and not attached sensors (with IDs= 0 ..(n-1)).

#### *pRecord*

[out] Points to a buffer to be used for returning the data record. WARNING: The size of the buffer must be equal to or greater than the size of the data record requested or the function may overwrite user memory with indeterminate results. Note also that when requesting data from all sensors (ID=ALL\_SENSORS), the buffer size must account for size of the data record \* N, where N is the max number of sensors supported by the system.

#### *recordSize*

[in] Contains the size of the buffer whose address is passed in *pRecord*. Note the *recordSize* value must match the size of the currently selected [DATA\\_FORMAT\\_TYPE](#) exactly.

### Return Values

The function returns a value of type *int*. This value takes the form of an [ERRORCODE](#) indicating success or failure for the call. The enumerated error code field contained within the 32-bit [ERRORCODE](#) may have one of the following values for this function call:

Value	Meaning
BIRD_ERROR_SUCCESS	No errors occurred. Call completed successfully
BIRD_ERROR_SYSTEM_UNINITIALIZED	The Tracker hardware and system has not been initialized yet. The <a href="#">InitializeBIRDSystem</a> function must be called first.
BIRD_ERROR_NO_SENSOR_ATTACHED	Request for data record from a sensor channel where no sensor is attached or the sensor has been removed.
BIRD_ERROR_NO_TRANSMITTER_RUNNING	Request for data record but there is no transmitter running. Either the application failed to turn a transmitter on or the currently running transmitter has a problem or has been removed. If a transmitter problem is suspected use the GetTransmitterStatus function to determine the precise problem.
BIRD_ERROR_INCORRECT_RECORD_SIZE	The <i>recordSize</i> of the buffer passed to the function does not match the size of the data format currently

	selected.
BIRD_ERROR_SENSOR_SATURATED	The attached sensor that is otherwise OK has gone into saturation. This may occur if the sensor is too close to the transmitter or if the sensor is too close to metal or an external magnetic field.
BIRD_ERROR_CPU_TIMEOUT	Tracker on-board controller had insufficient time to execute the position and orientation algorithm. This frequently occurs because the Tracker controller is being overwhelmed with user interface commands. Reduce the rate at which GetAsynchronousRecord is being called.
BIRD_ERROR_SENSOR_BAD	The attached sensor is not saturated but is exhibiting another unspecified problem that prevents it from operating normally. Use the GetSensorStatus function to determine the precise problem.
BIRD_ERROR_INVALID_DEVICE_ID	The transmitterID passed was out of range for the system.
BIRD_ERROR_COMMAND_TIME_OUT	Tracker on-board controller has failed to respond to a command issued to it. If error is repeatable there is an unrecoverable hardware failure.
BIRD_ERROR_WATCHDOG	Tracker internal watchdog timer has elapsed. It is necessary to re-initialize the system to recover from this error. If this error is repeatable there is an unrecoverable hardware failure.

## Remarks

The GetAsynchronousRecord function is designed to immediately return the data record from the last computation cycle. If this function is called repeatedly and at a greater rate than the measurement cycle, there will be duplication of data records.

In order to call this function, it is necessary to have already set the data format of the sensor that the record is being obtained from using the [SetSensorParameter\(\)](#) function. Once that is done, it is necessary to pass the ID of the sensor and a pointer to a buffer where the data record will be returned. It is also necessary to pass a parameter with the size of the buffer being passed. If there is a mismatch in the buffer sizes, the command is aborted and an error returned.

The enumerated data formats of type [DATA\\_FORMAT\\_TYPE](#) come in a number of general forms: integer, floating point, floating point with timestamp, floating point with timestamp and quality number, and all. For each form the user can select to have position only, angles only, attitude matrix only or quaternion only, or any of the previous combined with position returned.

## Requirements

**Windows NT/2000:** Requires Windows 2000 or later.

**Header:** Declared in ATC3DGm.h

**Library:** Use ATC3DGm.lib

## See Also

[GetSynchronousRecord](#)

## GetSynchronousRecord

The **GetSynchronousRecord** function allows an application to acquire unique position and orientation data records for a given sensor (or from all possible sensors), only as they are computed by the tracker and become available – once per data acquisition cycle.

```
int GetSynchronousRecord(
    USHORT sensorID
    void* pRecord,
    int recordSize
);
```

### Parameters

#### *sensorID*

[in] Valid sensorIDs for an individual sensor are in the range 0..(n-1) where n is the number of sensors in the system. A sensorID value of ALL\_SENSORS (-1), is used to request records from all possible sensors. Note that using the ALL\_SENSORS sensorID will result in data records in the specified buffer for both attached and not attached sensors (with IDs= 0 ..(n-1)).

#### *pRecord*

[out] Points to a buffer to be used for returning the data record. WARNING: The size of the buffer must be equal to or greater then the size of the data record requested or the function may overwrite user memory with indeterminate results. Note also that when requesting data from all sensors (ID=ALL\_SENSORS), the buffer size must account for size of the data record \* N, where N is the max number of sensors supported by the system.

#### *recordSize*

[in] Contains the size of the buffer whose address is passed in *pRecord*. Note the *recordSize* value must match the size of the currently selected [DATA\\_FORMAT\\_TYPE](#) exactly.

### Return Values

The function returns a value of type *int*. This value takes the form of an [ERRORCODE](#) indicating success or failure for the call. The enumerated error code field contained within the 32-bit [ERRORCODE](#) may have one of the following values for this function call:

Value	Meaning
BIRD_ERROR_SUCCESS	No errors occurred. Call completed successfully
BIRD_ERROR_SYSTEM_UNINITIALIZED	The 3DGuidance hardware and system has not been initialized yet. The <a href="#">InitializeBIRDSystem</a> function must be called first.
BIRD_ERROR_NO_SENSOR_ATTACHED	Request for data record from a sensor channel where no sensor is attached or the sensor has been removed.
BIRD_ERROR_NO_TRANSMITTER_RUNNING	Request for data record but there is no transmitter running. Either the application failed to turn a transmitter on or the currently running transmitter has a problem or has been removed. If a transmitter problem is suspected use the <a href="#">GetTransmitterStatus</a> function to determine the precise problem.
BIRD_ERROR_INCORRECT_RECORD_SIZE	The <i>recordSize</i> of the buffer passed to the function does not match the size of the data format currently

	selected.
BIRD_ERROR_SENSOR_SATURATED	The attached sensor which is otherwise OK has gone into saturation. This may occur if the sensor is too close to the transmitter or if the sensor is too close to metal or an external magnetic field.
BIRD_ERROR_CPU_TIMEOUT	3DGuidance on-board controller had insufficient time to execute the position and orientation algorithm. This frequently occurs because the 3DGuidance controller is being overwhelmed with user interface commands. Reduce the rate at which function is being called.
BIRD_ERROR_SENSOR_BAD	The attached sensor is not saturated but is exhibiting another unspecified problem which prevents it from operating normally. Use the <a href="#">GetSensorStatus</a> function to determine the precise problem.
BIRD_ERROR_INVALID_DEVICE_ID	The transmitterID passed was out of range for the system.
BIRD_ERROR_COMMAND_TIME_OUT	3DGuidance on-board controller has failed to respond to a command issued to it. If error is repeatable there is an unrecoverable hardware failure.
BIRD_ERROR_WATCHDOG	3DGuidance internal watchdog timer has elapsed. It is necessary to re-initialize the system to recover from this error. If this error is repeatable there is an unrecoverable hardware failure.

## Remarks

The GetSynchronousRecord function is designed to place the tracker in a data-reporting mode in which each and every computed data record is sent to the host. The result is a constant [STREAM](#) of data with timing that is independent of the arrival of the host data request during the measurement cycle. While this prevents the occurrence of duplicate records (as can occur when calling the GetAsynchronousRecord faster than the tracker update rate), it does not guarantee that records will not be overwritten. The buffer available to the system for each sensor is 8 records long. If the host application does not keep up with the constant stream of data being provided in this mode, this buffer will overflow and records will be lost.

Note that the rate at which records are transmitted when using the GetSynchronousRecord can be changed through use of the [SetSystemParameter](#) function with the [REPORT\\_RATE](#) parameter. This divisor reduces the number of records output during STREAM mode, to that determined by the setting. For example, at a system measurement rate of 80Hz and a REPORT\_RATE of 1, the tracker will transmit  $80 \times 3 = 240$  Updates/sec (1 record every 4mS) for each sensor. Changing the REPORT\_RATE setting to 4 will reduce the number of records to  $240/4 = 60$  Updates/sec (1 record every 17mS) for each sensor. The default REPORT\_RATE setting of 1 makes all outputs computed by the tracker available. See the [Configurable Settings](#) section in Chapter 3 for details on changing the default setting.

Issuing commands (other than GetSynchronousRecord) that must query the unit for a response, will cause the unit to come out of STREAM mode (i.e GetXXXX). Also note that hot-swapping sensors during STREAM mode operation will introduce delay in data for all sensor channels, as the unit must be taken out of this mode to detect and process info from the inserted sensor, then commanded to resume the STREAM mode operation.

As with the GetAsynchronous call, a record containing data from all sensors can be obtained by setting the sensor ID to ALL\_SENSORS. For legacy tracker users, this is the equivalent of Group Mode. Note also that when requesting data from all sensors (ID=ALL\_SENSORS), the buffer size must account for size of the data record \* N, where N is the max number of sensors supported by the system.

In order to call this function, it is necessary to have already set the data format of the sensor(s) that the record is being obtained from using the [SetSensorParameter\(\)](#) function. Once that is done, it is necessary to pass the ID of the



sensor and a pointer to a buffer where the data record(s) will be returned. It is also necessary to pass a parameter with the size of the buffer being passed. If there is a mismatch in the buffer sizes, the command is aborted and an error returned.

The enumerated data formats of type [DATA\\_FORMAT\\_TYPE](#) come in a number of general forms: integer, floating point, floating point with timestamp, floating point with timestamp and quality number, and all. For each form the user can select to have position only, angles only, attitude matrix only or quaternion only, or any of the previous combined with position returned.

### Requirements

**Windows NT/2000:** Requires Windows 2000 or later.

**Header:** Declared in ATC3DGm.h

**Library:** Use ATC3DG.lib

### See Also

[GET ASYNCHRONOUS RECORD](#)

## GetBIRDError

The **GetBIRDError** function forces the system to interrogate the hardware and update all the device status records. These status records may then be inspected using the [GetSensorStatus](#), [GetTransmitterStatus](#) and [GetBoardStatus](#) function calls.

```
int GetBIRDError();
```

### Parameters

This function takes no parameters

### Return Values

The function returns a value of type *int*. This value takes the form of an [ERRORCODE](#) indicating success or failure for the call. The enumerated error code field contained within the 32-bit [ERRORCODE](#) may have one of the following values for this function call:

Value	Meaning
BIRD_ERROR_SUCCESS	No errors occurred. Call completed successfully
BIRD_ERROR_SYSTEM_UNINITIALIZED	The Tracker hardware and system has not been initialized yet. The <a href="#">InitializeBIRDSystem</a> function must be called first.
BIRD_ERROR_COMMAND_TIME_OUT	Tracker on-board controller has failed to respond to a command issued to it. If error is repeatable there is an unrecoverable hardware failure.
BIRD_ERROR_WATCHDOG	Tracker internal watchdog timer has elapsed. It is necessary to re-initialize the system to recover from this error. If this error is repeatable there is an unrecoverable hardware failure.

### Remarks

The GetBIRDError function will cause the system to interrogate all boards and all sensor and transmitter channels on each board to determine the status of all attached and unattached devices. Consequently the execution of this command may take quite a long time and it is not recommended that it be called regularly. It should only be called after a period of inactivity to refresh the system's internal record of device status. Note: once the global status has been updated, specific device status will be regularly updated during calls to [GetAsynchronousRecord](#) or [GetSynchronousRecord](#). For example: In a system with two boards there will exist four sensor channels. Calling GetBIRDError will acquire the current status for all four channels. If the application then starts to make calls to GetAsynchronousRecord for sensor number 2 then the status for that sensor will be updated as necessary during the data acquisition.

### Requirements

**Windows NT/2000:** Requires Windows 2000 or later.

**Header:** Declared in ATC3DGm.h

**Library:** Use ATC3DGm.lib

### See Also

## GetErrorText

The **GetErrorText** function returns a message string describing the nature of the error code passed to it.

```
int GetErrorText(
    int errorCode,
    char* pBuffer,
    int bufferSize,
    int type
);
```

### Parameters

#### *errorCode*

[in] Contains an *int* value representing the error code parameter whose message string will be returned from the call. Must be a valid enumerated constant of the type [BIRD\\_ERROR\\_CODES](#).

#### *pBuffer*

[out] Points to a buffer that will be used to hold the message string returned from the call. WARNING: The actual buffer size must be equal to or greater than the *bufferSize* value passed or the function may overwrite beyond the end of the buffer into user memory with indeterminate results. Note: If the buffer provided is shorter than the string returned, the string will be truncated to fit into the buffer.

#### *bufferSize*

[in] Contains the size of the buffer whose address is passed in *pBuffer*.

#### *type*

[in] Contains an *int* value of enumerated type [MESSAGE\\_TYPE](#) which may have one of the following values:

Value	Meaning
SIMPLE_MESSAGE	A single line text string will be returned with a terse description of the error.
VERBOSE_MESSAGE	A more complete description of the error will be returned. The description may include possible causes of the error where appropriate and a description of the steps required to ameliorate the error condition.

### Return Values

The function returns a value of type *int*. This value takes the form of an [ERRORCODE](#) indicating success or failure for the call. The enumerated error code field contained within the 32-bit [ERRORCODE](#) may have one of the following values for this function call:

Value	Meaning
BIRD_ERROR_SUCCESS	No errors occurred. Call completed successfully
BIRD_ERROR_SYSTEM_UNINITIALIZED	The Tracker hardware and system has not been initialized yet. The <a href="#">InitializeBIRDSystem</a> function must be called first.
BIRD_ERROR_ILLEGAL_COMMAND_PARAMETER	Invalid enumerated constant of type <a href="#">BIRD_ERROR_CODES</a> was passed in parameter

	<i>errorCode.</i>
--	-------------------

**Remarks**

This is a helper function provided to simplify the error reporting process.

**Requirements**

**Windows NT/2000:** Requires Windows 2000 or later.

**Header:** Declared in ATC3DGm.h

**Library:** Use ATC3DGm.lib

**See Also**

## GetSensorStatus

The **GetSensorStatus** will return the status of the selected sensor channel.

```
DEVICE_STATUS GetSensorStatus(
    USHORT sensorID,
);
```

### Parameters

*sensorID*

[in] The sensorID is in the range 0..(n-1) where n is the number of possible sensors in the system.

### Return Values

The function returns a value of type [DEVICE\\_STATUS](#). The returned value contains the status of the selected sensor channel. The bits in the status word have the following meanings:

Bit	Name	Meaning
0	GLOBAL_ERROR	If the Error bit is cleared then the sensor is attached and fully operational. This bit is set if the following is true:  Error = Not Attached OR Saturated OR Bad EEPROM OR Hardware problem OR Non-Existent OR UnInitialized OR No Transmitter
1	NOT_ATTACHED	No sensor is attached to this sensor channel
2	SATURATED	The sensor is currently saturated.
3	BAD_EEPROM	The sensor is attached but the on-board EEPROM has a problem that renders the sensor unusable.
4	HARDWARE	The sensor is attached, the EEPROM checks out OK but there is an unspecified hardware failure that prevents the sensor from operating properly.
5	NON_EXISTENT	When Non-Existent is set it denotes that this is NOT a valid sensor channel. Note: No error codes are returned with GetSensorStatus calls. If the <i>sensorID</i> used is invalid then this bit will be set.
6	UNINITIALIZED	When UnInitialized is set it denotes that the system initialization function InitializeBIRDSsystem has NOT been called successfully at least once and the sensor status is invalid.
7	NO_TRANSMITTER	This bit will be set if one of the following is true: a) There is no transmitter attached to the system or b) There is a transmitter but it is not turned on.
8	BAD_12V	Always returns zero
9	CPU_TIMEOUT	CPU ran out of time while executing the position and orientation algorithm.
10	INVALID_DEVICE	The attached sensor is an invalid type for this board type.
11 - 31	Reserved (Unused)	Always returns zero

### Remarks

This function takes as its only parameter an index to a selected sensor. The function call returns a

32-bit status word.

No error codes are returned so it is not possible to determine if the call was successful through the standard process of inspecting the returned error code. But there are 2 possible runtime error conditions:

- 1) Calling the function before InitializeBIRDSystem has been called
- 2) Calling the function with an invalid sensor ID.

Both these conditions have been taken care of by the addition of bits 5 and 6 in the status word.

- 1) If the function InitializeBIRDSystem has not been called then the "UnInitialized" bit will be set.
- 2) If this function call was made with an invalid (out of range) sensor ID then the "Non-Existent" bit will be set.

In all cases the setting of any single status bit will cause the "Error" bit to be set. Determining if the sensor is operational can be done by simply testing the "Error" bit or by testing the whole status word. The sensor is only operational when the status word = 0.

Any call made to GetAsynchronousRecord with the "Error" bit set will result in a zero data record being returned. Conversely, any time that a zero data record is received the application should call GetSensorStatus to determine the cause of the problem.

## Requirements

**Windows NT/2000:** Requires Windows 2000 or later.

**Header:** Declared in ATC3DGm.h

**Library:** Use ATC3DGm.lib

## See Also

## GetTransmitterStatus

The **GetTransmitterStatus** will return the status of the selected transmitter channel.

```
DEVICE_STATUS GetTransmitterStatus(
    USHORT transmitterID,
);
```

### Parameters

*transmitterID*

[in] The transmitterID is in the range 0..(n-1) where n is the number of possible transmitters in the system.

### Return Values

The function returns a value of type [DEVICE\\_STATUS](#). The returned value contains the status of the selected transmitter channel. The bits in the status word have the following meanings:

Bit	Name	Meaning
0	GLOBAL_ERROR	If the Error bit is cleared then the transmitter is attached and fully operational. This bit is set if the following is true:  Error = Not Attached OR Bad EEPROM OR Hardware problem OR Non-Existent OR UnInitialized
1	NOT_ATTACHED	No transmitter is attached to this transmitter channel
2	SATURATED	Always returns zero.
3	BAD_EEPROM	The transmitter is attached but the on-board EEPROM has a problem that renders the transmitter unusable.
4	HARDWARE	The transmitter is attached, the EEPROM checks out OK but there is an unspecified hardware failure that prevents the transmitter from operating properly. Either an open circuit coil or an overcurrent condition can cause hardware problems.
5	NON_EXISTENT	When Non-Existent is set it denotes that this is NOT a valid transmitter channel. Note: No error codes are returned with GetTransmitterStatus calls. If the <i>transmitterID</i> used is invalid then this bit will be set.
6	UNINITIALIZED	When UnInitialized is set it denotes that the system initialization function InitializeBIRDSysyem has NOT been called successfully at least once and the transmitter status is invalid.
7	NO_TRANSMITTER	Always returns zero
8	BAD_12V	The +12V power supply has not been attached to the card that this transmitter is located on. This transmitter channel is therefore unusable.
10	INVALID_DEVICE	The attached transmitter is an invalid type for this board type.
11 - 31	Reserved (Unused)	Always returns zero

## Remarks

This function takes as its only parameter an index to a selected transmitter. The function call returns a 32-bit status word.

No error codes are returned so it is not possible to determine if the call was successful through the standard process of inspecting the returned error code. But there are 2 possible runtime error conditions:

- 1) Calling the function before InitializeBIRDSsystem has been called
- 2) Calling the function with an invalid transmitter ID.

Both these conditions have been taken care of by the addition of bits 5 and 6 in the status word.

- 1) If the function InitializeBIRDSsystem has not been called then the "UnInitialized" bit will be set.
- 2) If this function call was made with an invalid (out of range) transmitter ID then the "Non-Existent" bit will be set.

In all cases the setting of any single status bit will cause the "Error" bit to be set. Determining if the transmitter is operational can be done by simply testing the "Error" bit or by testing the whole status word. The transmitter is only operational when the status word = 0.

## Requirements

**Windows NT/2000:** Requires Windows 2000 or later.

**Header:** Declared in ATC3DGm.h

**Library:** Use ATC3DGm.lib

## See Also



## GetBoardStatus

The **GetBoardStatus** will return the status of the selected tracker unit.

```
DEVICE_STATUS GetBoardStatus(
    USHORT boardID,
);
```

### Parameters

*boardID*

[in] The boardID is in the range 0..(n-1) where n is the number of cards installed in the system.

### Return Values

The function returns a value of type [DEVICE\\_STATUS](#). The returned value contains the status of the selected tracker unit. The bits in the status word have the following meanings:

Bit	Name	Meaning
0	GLOBAL_ERROR	If the Error bit is cleared then the board is installed and fully operational. This bit is set if the following is true:  Error = Bad EEPROM OR Hardware problem OR Non-Existent OR UnInitialized  NOTE: This bit will NOT be set if the "+12V missing" bit is set.
1	NOT_ATTACHED	Always returns zero
2	SATURATED	Always returns zero
3	BAD_EEPROM	The board is installed but the on-board EEPROM has a problem that renders the board unusable.
4	HARDWARE	The board is installed, but there is an unspecified hardware failure that prevents the board from operating properly.
5	NON_EXISTENT	When Non-Existent is set it denotes that this is NOT a valid board ID. Note: No error codes are returned with GetBoardStatus calls. If the <i>boardID</i> used is invalid then this bit will be set.
6	UNINITIALIZED	When UnInitialized is set it denotes that the system initialization function InitializeBIRDSsystem has NOT been called successfully at least once and the board status is invalid.
7	NO_TRANSMITTER	Always returns zero
8	BAD_12V	The +12V power supply has not been attached to this card. The transmitter channel on this card is unusable. NOTE: This is not a fatal error and does not render the board totally unusable. Setting this bit does not set the Error bit.
9	CPU_TIMEOUT	CPU ran out of time while executing the position and orientation algorithm.
10 - 31	Reserved (Unused)	Always returns zero

## Remarks

This function takes as its only parameter an index to a selected tracker unit. The function call returns a 32-bit status word.

No error codes are returned so it is not possible to determine if the call was successful through the standard process of inspecting the returned error code. But there are 2 possible runtime error conditions:

- 1) Calling the function before InitializeBIRDSsystem has been called
- 2) Calling the function with an invalid board ID.

Both these conditions have been taken care of by the addition of bits 5 and 6 in the status word.

- 1) If the function InitializeBIRDSsystem has not been called then the "UnInitialized" bit will be set.
- 2) If this function call was made with an invalid (out of range) board ID then the "Non-Existent" bit will be set.

In all cases the setting of any single status bit will cause the "Error" bit to be set. (Except for the +12V Missing status bit) Determining if the board is operational should be done by simply testing the "Error" bit. Testing the entire status word for a value of 0 may or may be successful depending on whether the +12V is installed or not.

## Requirements

**Windows NT/2000:** Requires Windows 2000 or later.

**Header:** Declared in ATC3DGm.h

**Library:** Use ATC3DGm.lib

## See Also

## GetSystemStatus

The **GetSystemStatus** will return the status of the Tracker system.

```
DEVICE_STATUS GetSystemStatus();
```

### Parameters

This function takes no parameters

### Return Values

The function returns a value of type [DEVICE\\_STATUS](#). The returned value contains the status of the tracking system. The bits in the status word have the following meanings:

Bit	Name	Meaning
0	GLOBAL_ERROR	If the Error bit is cleared then the system is fully operational. This bit is set if the following is true:  Error = Hardware problem OR Non-Existent OR UnInitialized OR +12V Missing
1	NOT_ATTACHED	Always returns zero
2	SATURATED	Always returns zero
3	BAD_EEPROM	Always returns zero
4	HARDWARE	There is a fatal hardware failure somewhere that prevents the system from operating.
5	NON_EXISTENT	No Tracker cards have been found in the host system. It is necessary to install at least one card before attempting to initialize the system.
6	UNINITIALIZED	When UnInitialized is set it denotes that the system initialization function InitializeBIRDSsystem has NOT been called successfully at least once and the system status is invalid.
7	NO_TRANSMITTER	Always returns zero
8	BAD_12V	The +12V power supply has not been attached to any card in the system. At least one board will need to have the +12V attached in order to drive a transmitter.
9 - 31	Reserved (Unused)	Always returns zero

### Remarks

No error codes are returned so it is not possible to determine if the call was successful through the standard process of inspecting the returned error code. But there is one possible runtime error condition, namely, calling the function before InitializeBIRDSsystem has been called. If the function InitializeBIRDSsystem has not been called then the "UnInitialized" bit will be set.

In all cases, the setting of any single status bit will cause the “Error” bit to be set. Determining if the system is operational can be done by simply testing the “Error” bit or by testing the whole status word. The system is only operational when the status word = 0.

## Requirements

**Windows NT/2000:** Requires Windows 2000 or later.

**Header:** Declared in ATC3DGm.h

**Library:** Use ATC3DGm.lib

## See Also

## SaveSystemConfiguration

The **SaveSystemConfiguration** will save the current setup of the system to a file.

```
int SaveSystemConfiguration(
    LPCTSTR lpFileName
);
```

### Parameters

*lpFileName*

[in] Pointer to a null-terminated string that specifies the name of the file to create.

### Return Values

The function returns a value of type *int*. This value takes the form of an [ERRORCODE](#) indicating success or failure for the call. The enumerated error code field contained within the 32-bit [ERRORCODE](#) may have one of the following values for this function call:

Value	Meaning
BIRD_ERROR_SUCCESS	No errors occurred. Call completed successfully
BIRD_ERROR_SYSTEM_UNINITIALIZED	The Tracker hardware and system has not been initialized yet. The <a href="#">InitializeBIRDSystem</a> function must be called first.
BIRD_ERROR_UNABLE_TO_CREATE_FILE	The call was unable to complete for some unspecified reason. Check the format of the file name string.
BIRD_ERROR_CONFIG_INTERNAL	Internal error in configuration file handler. Report to vendor.

### Remarks

The only parameter to this call is the null-terminated string containing the file name. Note: In order to include a backslash (\) as a separator in the file name string it is necessary to precede it with a second backslash. See the example below.

```
int error = SaveSystemConfiguration("C:\\Configurations\\MyConfiguration.ini");
```

NOTE: If the file name is given without a full pathname specification then the file will be saved into the current directory. For example in the following example if the application is executing from <C:\MyPrograms> then the following call

```
int error = SaveSystemConfiguration("MyConfiguration.ini");
```

will save the configuration file to <C:\MyPrograms\MyConfiguration.ini>. This default mode of operation differs from the `RestoreSystemConfiguration()` call that uses the %windir%\inf directory as the default directory.

The configuration that is saved contains all of the parameters initialized using the SetSystemParameter, SetSensorParameter and SetTransmitterParameter function calls. The parameters that can be initialized with each of these calls are listed in the enumerated types SYSTEM\_PARAMETER\_TYPE, SENSOR\_PARAMETER\_TYPE and TRANSMITTER\_PARAMETER\_TYPE. Any parameters that are uninitialized will be saved with their default values.

The file format is described in [PCIBird Initialization File Format](#) section.

## Requirements

**Windows NT/2000:** Requires Windows 2000 or later.

**Header:** Declared in ATC3DGm.h

**Library:** Use ATC3DGm.lib

## See Also

[RestoreSystemConfiguration](#)

## RestoreSystemConfiguration

The **RestoreSystemConfiguration** will restore the system configuration to a previous state that has been saved in a file.

```
int RestoreSystemConfiguration(
    LPCTSTR lpFileName
);
```

### Parameters

*lpFileName*

[in] Pointer to a null-terminated string that specifies the name of the file to open.

### Return Values

The function returns a value of type *int*. This value takes the form of an [ERRORCODE](#) indicating success or failure for the call. The enumerated error code field contained within the 32-bit [ERRORCODE](#) may have one of the following values for this function call:

Value	Meaning
BIRD_ERROR_SUCCESS	No errors occurred. Call completed successfully
BIRD_ERROR_SYSTEM_UNINITIALIZED	The Tracker hardware and system has not been initialized yet. The <a href="#">InitializeBIRDSystem</a> function must be called first.
BIRD_ERROR_UNABLE_TO_OPEN_FILE	The call was unable to complete for some unspecified reason. Check the format of the file name string.
BIRD_ERROR_MISSING_CONFIGURATION_ITEM	A mandatory configuration item was missing from the initialization file. Review contents of initialization file or use SaveSystemConfiguration() to automatically save a correctly formatted initialization file.
BIRD_ERROR_MISMATCHED_DATA	Data item in the initialization file does not match a system parameter. For example the initialization file states the system has 3 boards (NumberOfBoards=3) but the system initialization routine – InitializeBIRDSystem() only detected two.
BIRD_ERROR_CONFIG_INTERNAL	Internal error in configuration file handler. Report to vendor.

### Remarks

The only parameter to this call is the null-terminated string containing the file name. Note: In order to include a backslash (\) as a separator in the file name string it is necessary to precede it with a second backslash. See the example below.

```
int error = RestoreSystemConfiguration("C:\\Configurations\\MyConfiguration.ini");
```

NOTE: if the full pathname specification is not provided then the default search path is in the %windir%\inf directory. If the file is not found there then a BIRD\_ERROR\_UNABLE\_TO\_OPEN\_FILE error is generated.

The configuration that is restored contains all of the parameters that can be alternatively initialized using the SetSystemParameter, SetSensorParameter and SetTransmitterParameter function calls. The parameters that can be initialized with each of these calls are listed in the enumerated types SYSTEM\_PARAMETER\_TYPE, SENSOR\_PARAMETER\_TYPE and TRANSMITTER\_PARAMETER\_TYPE.

The file format is described in tracker Initialization File Format

## Requirements

**Windows NT/2000:** Requires Windows 2000 or later.

**Header:** Declared in ATC3DGm.h

**Library:** Use ATC3DGm.lib

## See Also

[SaveSystemConfiguration](#)



## CloseBIRDSystem

The **CloseBIRDSystem** function **shuts down the tracker**. `int CloseBIRDSystem();`

### Parameters

This function takes no parameters

### Return Values

The function returns a value of type *int*. This value takes the form of an [ERRORCODE](#) indicating success or failure for the call. The enumerated error code field contained within the 32-bit [ERRORCODE](#) may have one of the following values for this function call:

Value	Meaning
BIRD_ERROR_SUCCESS	No errors occurred. Call completed successfully

### Remarks

The CloseBIRDSystem function will return the tracker to an uninitialized state and release all resources and handles that were being used. It is recommended that this be called prior to terminating an application that has been using the tracker in order to prevent memory and/or resource leaks.

### Requirements

**Windows NT/2000:** Requires Windows 2000 or later.

**Header:** Declared in ATC3DGm.h

**Library:** Use ATC3DGm.lib

### See Also

## 3D Guidance API Structures

The following structures are used with your tracker:

[SYSTEM CONFIGURATION](#)  
[SENSOR CONFIGURATION](#)  
[TRANSMITTER CONFIGURATION](#)  
[BOARD CONFIGURATION](#)  
[ADAPTIVE PARAMETERS](#)  
[QUALITY PARAMETERS](#)

Data record structures:

[SHORT POSITION RECORD](#)  
[SHORT ANGLES RECORD](#)  
[SHORT MATRIX RECORD](#)  
[SHORT QUATERNIONS RECORD](#)  
[SHORT POSITION ANGLES RECORD](#)  
[SHORT POSITION MATRIX RECORD](#)  
[SHORT POSITION QUATERNION RECORD](#)  
[DOUBLE POSITION RECORD](#)  
[DOUBLE ANGLES RECORD](#)  
[DOUBLE MATRIX RECORD](#)  
[DOUBLE QUATERNIONS RECORD](#)  
[DOUBLE POSITION ANGLES RECORD](#)  
[DOUBLE POSITION MATRIX RECORD](#)  
[DOUBLE POSITION QUATERNION RECORD](#)  
[DOUBLE POSITION TIME STAMP RECORD](#)  
[DOUBLE ANGLES TIME STAMP RECORD](#)  
[DOUBLE MATRIX TIME STAMP RECORD](#)  
[DOUBLE QUATERNIONS TIME STAMP RECORD](#)  
[DOUBLE POSITION ANGLES TIME STAMP RECORD](#)  
[DOUBLE POSITION MATRIX TIME STAMP RECORD](#)  
[DOUBLE POSITION QUATERNION TIME STAMP RECORD](#)  
[DOUBLE POSITION TIME Q RECORD](#)  
[DOUBLE ANGLES TIME Q RECORD](#)  
[DOUBLE MATRIX TIME Q RECORD](#)  
[DOUBLE QUATERNIONS TIME Q RECORD](#)  
[DOUBLE POSITION ANGLES TIME Q RECORD](#)  
[DOUBLE POSITION MATRIX TIME Q RECORD](#)  
[DOUBLE POSITION QUATERNION TIME Q RECORD](#)  
[SHORT ALL RECORD](#)  
[DOUBLE ALL RECORD](#)  
[DOUBLE ALL TIME STAMP RECORD](#)  
[DOUBLE ALL TIME STAMP Q RECORD](#)  
[DOUBLE ALL TIME STAMP Q RAW RECORD](#)  
[DOUBLE POSITION ANGLES TIME Q BUTTON](#)  
[DOUBLE POSITION MATRIX TIME Q BUTTON](#)  
[DOUBLE POSITION QUATERNION TIME Q BUTTON](#)

## SYSTEM\_CONFIGURATION

The **SYSTEM\_CONFIGURATION** structure contains the system information.

```
typedef struct tagSYSTEM_CONFIGURATION{
    double      measurementRate;
    double      powerLineFrequency;
    double      maximumRange;
    AGC_MODE_TYPE agcMode;
    int          numberBoards;
    int          numberSensors;
    int          numberTransmitters;
    int          transmitterIDRunning;
    bool         metric;
} SYSTEM_CONFIGURATION, *PSYSTEM_CONFIGURATION;
```

### Members

#### measurementRate

Indicates the current measurement rate of the tracking system.

#### powerLineFrequency

Indicates current power line frequency being used to set filter coefficients; Default line frequency is 60 Hz.

#### maximumRange

Indicates scale factor used by the tracker to report position of sensor with respect to the transmitter. Valid value of 36, represents full-scale position output in inches.

#### agcMode

Enumerated constant of the type: [AGC\\_MODE\\_TYPE](#). Setting the mode to SENSOR\_AGC\_ONLY disables the normal transmitter power level switching.

#### numberBoards

Indicates the number of tracker cards installed.

#### numberSensors

Indicates the number of ports available to plug in sensors.

#### numberTransmitters

Indicates the number of ports available to plug in transmitters.

#### transmitterIDRunning

Indicates ID of the transmitter that is ON.  
Default is -1 (Transmitter OFF).

#### metric

TRUE = data output in millimeters  
FALSE = output in inches (default)

**Requirements**

**Windows NT/2000:** Requires Windows 2000 or later.

**Header:** Declared in ATC3DGm.h

**Library:** Use ATC3DGm.lib

**See Also**

## TRANSMITTER\_CONFIGURATION

The **TRANSMITTER\_CONFIGURATION** structure contains an individual transmitter's information.

```
typedef struct tagTRANSMITTER_CONFIGURATION{
    ULONG        serialNumber;
    USHORT       boardNumber;
    USHORT       channelNumber;
    DEVICE_TYPE type;
    bool         attached;
} TRANSMITTER_CONFIGURATION, *PTRANSMITTER_CONFIGURATION;
```

### Members

#### **serialNumber**

The serial number of the attached transmitter. If no transmitter is attached this value is zero

#### **boardNumber**

The id number of the board for this transmitter channel.

#### **channelNumber**

The number of the channel on the board where this transmitter is located. Note: Currently boards only support single transmitters so this value will always be 0.

#### **type**

Contains a value of enumerated type `DEVICE_TYPES`.

#### **attached**

Contains a value of type *bool* whose value will be *true* if there is a transmitter attached otherwise it will be *false* if there is no transmitter attached. This value may be *true* even if there is a problem with the transmitter. A call should be made to [GetTransmitterStatus](#) to determine the operational state of the transmitter.

### Requirements

**Windows NT/2000:** Requires Windows 2000 or later.

**Header:** Declared in ATC3DGm.h

**Library:** Use ATC3DGm.lib

### See Also

## SENSOR\_CONFIGURATION

The **SENSOR\_CONFIGURATION** structure contains an individual sensor's information.

```
typedef struct tagSENSOR_CONFIGURATION{
    ULONG        serialNumber;
    USHORT       boardNumber;
    USHORT       channelNumber;
    DEVICE_TYPE type;
    bool         attached;
} SENSOR_CONFIGURATION, *PSENSOR_CONFIGURATION;
```

### Members

#### **serialNumber**

The serial number of the attached sensor. If no sensor is attached this value is zero

#### **boardNumber**

The id number of the board for this sensor channel.

#### **channelNumber**

The number of the channel on the board where this sensor is located.

#### **type**

Contains a value of enumerated type [DEVICE\\_TYPES](#).

#### **attached**

Contains a value of type *bool*/whose value will be *true* if there is a sensor attached otherwise it will be *false* if there is no sensor attached. This value may be *true* even if there is a problem with the sensor. A call should be made to [GetSensorStatus](#) to determine the operational state of the sensor.

### Requirements

**Windows NT/2000:** Requires Windows 2000 or later.

**Header:** Declared in ATC3DGm.h

**Library:** Use ATC3DGm.lib

### See Also

## BOARD\_CONFIGURATION

The **BOARD\_CONFIGURATION** structure contains an individual board's information.

```
typedef struct tagBOARD_CONFIGURATION{
    ULONG        serialNumber;
    BOARD_TYPES   type;
    USHORT        revision;
    USHORT        numberTransmitters;
    USHORT        numberSensors;
    USHORT        firmwareNumber;
    USHORT        firmwareRevision;
    Char          modelString[10];
} BOARD_CONFIGURATION, *PBOARD_CONFIGURATION;
```

### Members

#### **serialNumber**

The serial number of the board.

#### **type**

The board type. The type is of the enumeration type [BOARD\\_TYPES](#).

#### **revision**

The board ECO revision number.

#### **numberTransmitters**

This value denotes the number of available transmitter channels supported by this board.

#### **numberSensors**

This value denotes the number of available sensor channels supported by this board.

#### **firmwareNumber**

The firmware version of the on-board firmware is a two-part number usually denoted as a number and a fraction, e.g. 3.85. The integer number part is contained in the firmwareNumber.

#### **firmwareRevision**

The firmwareRevision contains the fractional part of the firmware version number.

#### **modelString[10]**

Each board has a configuration EEPROM. Contained in the EEPROM are the calibration values belonging to the board. Also contained in the EEPROM is a "model string" which is used to identify the board type. The modelString is a 10-character array, which contains the "model string". The string is not null-terminated. For example the dual 8mm sensor PCIBird card will have the string "6DPCI8MM ". The string is padded with space characters to the end of the buffer.

### Requirements

**Windows NT/2000:** Requires Windows 2000 or later.

**Header:** Declared in ATC3DGm.h

**Library:** Use ATC3DGm.lib

#### See Also



## ADAPTIVE\_PARAMETERS

The **ADAPTIVE\_PARAMETERS** structure contains the adaptive DC filter parameters for an individual sensor channel.

```
typedef struct tagADAPTIVE_PARAMETERS{
    USHORT      alphaMin[7];
    USHORT      alphaMax[7];
    USHORT      vm[7];
    bool         alphaOn;
} ADAPTIVE_PARAMETERS, *PADAPTIVE_PARAMETERS;
```

### Members

#### **alphaMin[7]**

The **alphaMin** values define the lower ends of the adaptive range that the filter constant alpha can assume in the DC filter, as a function of sensor to transmitter. NOTE: Each of the 7 array positions corresponds to a sensor gain setting with position 0 corresponding to the lowest gain setting when the sensor is closest to the transmitter.

#### **alphaMax[7]**

The **alphaMax** values define the upper ends of the adaptive range that the filter constant alpha can assume in the DC filter, as a function of sensor to transmitter. NOTE: Each of the 7 array positions corresponds to a sensor gain setting with position 0 corresponding to the lowest gain setting when the sensor is closest to the transmitter.

#### **vm[7]**

The 7 words that make up the **vm** array represent the expected noise that the DC filter will measure. By changing the table values, you can increase or decrease the DC filter's lag as a function of sensor range from the transmitter. NOTE: Each of the 7 array positions corresponds to a sensor gain setting with position 0 corresponding to the lowest gain setting when the sensor is closest to the transmitter.

#### **alphaOn**

This Boolean value is used to enable or disable the adaptive DC filter.

### Requirements

**Windows NT/2000:** Requires Windows 2000 or later.

**Header:** Declared in ATC3DGm.h

**Library:** Use ATC3DGm.lib

### See Also

## QUALITY\_PARAMETERS

The **QUALITY\_PARAMETERS** structure contains the parameters used to setup the distortion detection algorithm for an individual sensor channel.

```
typedef struct tagQUALITY_PARAMETERS{
    USHORT          error_slope;
    USHORT          error_offset;
    USHORT          error_sensitivity;
    USHORT          filter_alpha;
} QUALITY_PARAMETERS, *PQUALITY_PARAMETERS;
```

### Members

#### **error\_slope**

This value is the slope of the inherent system error. It will need to be adjusted depending on the type of hardware used. The final distortion error delivered to the application is the total system error – inherent system error.

#### **error\_offset**

This value is the offset of the inherent system error.

#### **error\_sensitivity**

This value is used to increase or decrease the sensitivity of the algorithm to distortion error. The distortion error is equal to the total system error – inherent system error. This value is then multiplied by the error\_sensitivity.

#### **filter\_alpha**

The output error value has considerable noise in it. An alpha filter is used to filter the output value. The amount of filtering applied can be adjusted by setting the filter\_alpha value.

### Requirements

**Windows NT/2000:** Requires Windows 2000 or later.

**Header:** Declared in ATC3DGm.h

**Library:** Use ATC3DGm.lib

### See Also

## VPD\_COMMAND\_PARAMETER

The **VPD\_COMMAND\_PARAMETER** structure contains the parameters used to read and write to the Vital Product Data (VPD) storage areas. 128-byte VPD storage areas are provided for the user in the EEPROMs of the electronics unit, transmitter, sensors and preamps. Using the corresponding SetXXXXParameter and GetXXXXParameter commands it is possible to read and write individual bytes within the VPD storage area.

```
typedef struct tagVPD_COMMAND_PARAMETER{
    USHORT      address;
    UCHAR       value;
} VPD_COMMAND_PARAMETER;
```

### Members

#### address

This value is the 0-based address of a byte within the VPD that is the target for either a read or a write operation.

#### value

This parameter contains the actual byte value to be written to the VPD location specified by address during a write operation (SetXXXXParameter) or it is a location where the value read from the VPD will be placed during a read operation (GetXXXXParameter).

### Requirements

**Windows NT/2000:** Requires Windows 2000 or later.

**Header:** Declared in ATC3DGm.h

**Library:** Use ATC3DGm.lib

### See Also

## BOARD\_REVISIONS

The **BOARD\_REVISIONS** structure contains the parameters used to return the revisions of the firmware in the 3DGuidance board.

```
typedef struct tagBOARD_REVISIONS{
    USHORT    boot_loader_sw_number;
    USHORT    boot_loader_sw_revision;
    USHORT    mdsp_sw_number;
    USHORT    mdsp_sw_revision;
    USHORT    nondipole_sw_number;
    USHORT    nondipole_sw_revision;
    USHORT    fivedof_sw_number;
    USHORT    fivedof_sw_revision;
    USHORT    sixdof_sw_number;
    USHORT    sixdof_sw_revision;
    USHORT    dipole_sw_number;
    USHORT    dipole_sw_revision;
} BOARD_REVISIONS;
```

### Members

#### **boot\_loader\_sw\_number**

Major revision number for the boot loader.

#### **boot\_loader\_sw\_revision**

Minor revision number for the boot loader.

#### **mdsp\_sw\_number**

Major revision number for the acquisition DSP.

#### **mdsp\_sw\_revision**

Minor revision number for the acquisition DSP.

#### **nondipole\_sw\_number**

Major revision number for the non-dipole DSP.

#### **nondipole\_sw\_revision**

Minor revision number for the non-dipole DSP.

#### **fivedof\_sw\_number**

Major revision number for the 5DOF DSP.

#### **fivedof\_sw\_revision**

Minor revision number for the 5DOF DSP.

#### **sixdof\_sw\_number**

Major revision number for the 6DOF DSP.

#### **sixdof\_sw\_revision**

Minor revision number for the 6DOF DSP.

#### **dipole\_sw\_number**

Major revision number for the dipole DSP.

#### **dipole\_sw\_revision**

Minor revision number for the dipole DSP.

### Requirements

**Windows NT/2000:** Requires Windows 2000 or later.

**Header:** Declared in ATC3DGm.h

**Library:** Use ATC3DGm.lib

#### See Also

## SHORT\_POSITION\_RECORD

The **SHORT\_POSITION\_RECORD** structure contains position information only in 16-bit signed integer format.

```
typedef struct tagSHORT_POSITION{
    short      x;
    short      y;
    short      z;
} SHORT_POSITION_RECORD, *PSHORT_POSITION_RECORD;
```

### Members

#### x

This is the x position value of a 3-axis coordinate position system. The value is a short integer. In order to determine the true position it is necessary to divide by 32768 (8000 hex) and multiply by the maximum range.

#### y

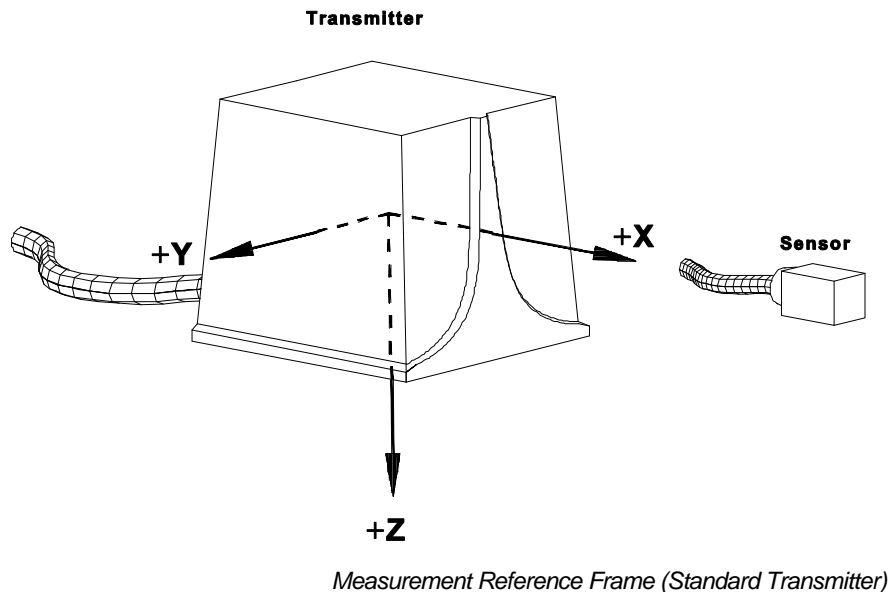
This is the y position value of a 3-axis coordinate position system. The value is a short integer. In order to determine the true position it is necessary to divide by 32768 (8000 hex) and multiply by the maximum range.

#### z

This is the z position value of a 3-axis coordinate position system. The value is a short integer. In order to determine the true position it is necessary to divide by 32768 (8000 hex) and multiply by the maximum range.

### Remarks

The X, Y and Z values vary between the binary equivalent of +/- maximum range. The positive X, Y and Z directions are shown below.



**Requirements**

**Windows NT/2000:** Requires Windows 2000 or later.

**Header:** Declared in ATC3DGm.h

**Library:** Use ATC3DGm.lib

**See Also**

## SHORT\_ANGLES\_RECORD

The **SHORT\_ANGLES\_RECORD** structure contains Euler angle information only in 16-bit signed integer format.

```
typedef struct tagSHORT_ANGLES{
    short      a;
    short      e;
    short      r;
} SHORT_ANGLES_RECORD, *PSHORT_ANGLES_RECORD;
```

### Members

**a**

This value is the azimuth angle of the sensor. The value is a short integer. In order to determine the true angle it is necessary to divide by 32768 (8000 hex) and multiply by 180 degrees.

**e**

This value is the elevation angle of the sensor. The value is a short integer. In order to determine the true angle it is necessary to divide by 32768 (8000 hex) and multiply by 180 degrees.

**r**

This value is the roll angle of the sensor. The value is a short integer. In order to determine the true angle it is necessary to divide by 32768 (8000 hex) and multiply by 180 degrees.

### Remarks

In the ANGLES mode, the tracker outputs the orientation angles of the sensor with respect to the transmitter. The orientation angles are defined as rotations about the Z, Y, and X axes of the sensor. These angles are called Zang, Yang, and Xang or, in Euler angle nomenclature, Azimuth, Elevation, and Roll.

Zang (Azimuth) takes on values between the binary equivalent of +/- 180 degrees. Yang (Elevation) takes on values between +/- 90 degrees, and Xang (Roll) takes on values between +/- 180 degrees. As Yang (Elevation) approaches +/- 90 degrees, the Zang (Azimuth) and Xang (Roll) become very noisy and exhibit large errors. At 90 degrees the Zang (Azimuth) and Xang (Roll) become undefined. This behavior is not a limitation of the tracker - it is an inherent characteristic of these Euler angles. If you need a stable representation of the sensor orientation at high Elevation angles, use the MATRIX output mode.

The scaling of all angles is full scale = 180 degrees. That is, +179.99 deg = 7FFF Hex, 0 deg = 0 Hex, -180.00 deg = 8000 Hex.

Angle information is 0 when sensor saturation occurs.

To convert the numbers received into angles in degrees, first multiply by 180 and finally divide the number by 32768 to get the angle. The equation should look something like:

$$\text{Angle} = (\text{signed int} * 180) / 32768;$$

### Requirements

**Windows NT/2000:** Requires Windows 2000 or later.

**Header:** Declared in ATC3DGm.h

**Library:** Use ATC3DGm.lib



## See Also

## SHORT\_MATRIX\_RECORD

The **SHORT\_MATRIX\_RECORD** structure contains only the 3x3 rotation matrix 'S' in 16-bit signed integer format.

```
typedef struct tagSHORT_MATRIX{
    short      s[3][3];
} SHORT_MATRIX_RECORD, *PSHORT_MATRIX_RECORD;
```

### Members

#### s[3][3]

This is a 3x3 array of values. Each value is delivered as a 16-bit signed integer. In order to convert each value to a number in the range +1 -> -1 it is necessary to divide each value by 32768 (8000 hex). The signed integer has a range of +32767 -> -32768 which when divided by 32768 will give a fractional number in the range 0.99997 -> -1.00000.

### Remarks

The MATRIX mode outputs the 9 elements of the rotation matrix that define the orientation of the sensor's X, Y, and Z axes with respect to the transmitter's X, Y, and Z axes. If you want a three-dimensional image to follow the rotation of the sensor, you must multiply your image coordinates by this output matrix.

The nine elements of the output matrix are defined generically by:

**M(1,1)    M(1,2)    M(1,3)**

**M(2,1)    M(2,2)    M(2,3)**

**M(3,1)    M(3,2)    M(3,3)**

Or in terms of the rotation angles about each axis where **Z = Zang**, **Y = Yang** and **X = Xang**:

<b>COS(Y)*COS(Z)</b>	<b>COS(Y)*SIN(Z)</b>	<b>-SIN(Y)</b>
<b>-(COS(X)*SIN(Z))</b>	<b>(COS(X)*COS(Z))</b>	
<b>+ (SIN(X)*SIN(Y)*COS(Z))</b>	<b>+ (SIN(X)*SIN(Y)*SIN(Z))</b>	<b>SIN(X)*COS(Y)</b>
<b>(SIN(X)*SIN(Z))</b>	<b>-(SIN(X)*COS(Z))</b>	
<b>+ (COS(X)*SIN(Y)*COS(Z))</b>	<b>+ (COS(X)*SIN(Y)*SIN(Z))</b>	<b>COS(X)*COS(Y)</b>

Or in Euler angle notation, where R = Roll, E = Elevation, A = Azimuth:

$\cos(E) * \cos(A)$	$\cos(E) * \sin(A)$	$-\sin(E)$
$-(\cos(R) * \sin(A))$ $+(\sin(R) * \sin(E) * \cos(A))$	$(\cos(R) * \cos(A))$ $+(\sin(R) * \sin(E) * \sin(A))$	$\sin(R) * \cos(E)$
$(\sin(R) * \sin(A))$ $+(\cos(R) * \sin(E) * \cos(A))$	$-(\sin(R) * \cos(A))$ $+(\cos(R) * \sin(E) * \sin(A))$	$\cos(R) * \cos(E)$

The matrix elements take values between the binary equivalents of  $+.99996$  and  $-1.0$ .

Element scaling is  $+.99996 = 7FFF$  Hex,  $0 = 0$  Hex, and  $-1.0 = 8000$  Hex.

Matrix information is 0 when sensor saturation occurs.

### Requirements

**Windows NT/2000:** Requires Windows 2000 or later.

**Header:** Declared in ATC3DGm.h

**Library:** Use ATC3DGm.lib

### See Also

## SHORT\_QUATERNIONS\_RECORD

The **SHORT\_QUATERNIONS\_RECORD** structure contains only the 4 quaternion values in 16-bit signed integer format.

```
typedef struct tagSHORT_QUATERNIONS{
    short      q[4];
} SHORT_QUATERNIONS_RECORD, *PSHORT_QUATERNIONS_RECORD;
```

### Members

#### q[4]

This is an array of 4 quaternion values. Each value is delivered as a 16-bit signed integer. In order to convert each value to a number in the range +1 -> -1 it is necessary to divide each value by 32768 (8000 hex). The signed integer has a range of +32767 -> -32768 which when divided by 32768 will give a fractional number in the range 0.99997 -> -1.00000.

### Remarks

In the QUATERNION mode, the Tracker outputs the four quaternion parameters that describe the orientation of the sensor with respect to the transmitter. The quaternion,  $q_0$ ,  $q_1$ ,  $q_2$ , and  $q_3$  where  $q_0$  is the scalar component, have been extracted from the MATRIX output using the algorithm described in "Quaternion from Rotation Matrix" by Stanley W. Shepperd, Journal of Guidance and Control, Vol. 1, May-June 1978, pp. 223-4.

### Requirements

**Windows NT/2000:** Requires Windows 2000 or later.

**Header:** Declared in ATC3DGm.h

**Library:** Use ATC3DGm.lib

### See Also

## SHORT\_POSITION\_ANGLES\_RECORD

The **SHORT\_POSITION\_ANGLES\_RECORD** structure contains position and angles information in 16-bit signed integer format.

```
typedef struct tagSHORT_POSITION_ANGLES{
    short      x;
    short      y;
    short      z;
    short      a;
    short      e;
    short      r;
} SHORT_POSITION_ANGLES_RECORD, *PSHORT_POSITION_ANGLES_RECORD;
```

### Members

#### x

This is the x position value of a 3-axis coordinate position system. The value is a short integer. In order to determine the true position it is necessary to divide by 32768 (8000 hex) and multiply by the maximum range.

#### y

This is the y position value of a 3-axis coordinate position system. The value is a short integer. In order to determine the true position it is necessary to divide by 32768 (8000 hex) and multiply by the maximum range.

#### z

This is the z position value of a 3-axis coordinate position system. The value is a short integer. In order to determine the true position it is necessary to divide by 32768 (8000 hex) and multiply by the maximum range.

#### a

This value is the azimuth angle of the sensor. The value is a short integer. In order to determine the true angle it is necessary to divide by 32768 (8000 hex) and multiply by 180 degrees.

#### e

This value is the elevation angle of the sensor. The value is a short integer. In order to determine the true angle it is necessary to divide by 32768 (8000 hex) and multiply by 180 degrees.

#### r

This value is the roll angle of the sensor. The value is a short integer. In order to determine the true angle it is necessary to divide by 32768 (8000 hex) and multiply by 180 degrees.

### Requirements

**Windows NT/2000:** Requires Windows 2000 or later.

**Header:** Declared in ATC3DGm.h

**Library:** Use ATC3DGm.lib

### See Also

SHORT\_POSITION\_RECORD, SHORT\_POSITION\_ANGLES\_RECORD

## SHORT\_POSITION\_MATRIX\_RECORD

The **SHORT\_POSITION\_MATRIX\_RECORD** structure contains position and matrix information in 16-bit signed integer format.

```
typedef struct tagSHORT_POSITION_MATRIX{
    short      x;
    short      y;
    short      z;
    short      s[3][3];
} SHORT_POSITION_MATRIX_RECORD, *PSHORT_POSITION_MATRIX_RECORD;
```

### Members

#### **x**

This is the x position value of a 3-axis coordinate position system. The value is a short integer. In order to determine the true position it is necessary to divide by 32768 (8000 hex) and multiply by the maximum range.

#### **y**

This is the y position value of a 3-axis coordinate position system. The value is a short integer. In order to determine the true position it is necessary to divide by 32768 (8000 hex) and multiply by the maximum range.

#### **z**

This is the z position value of a 3-axis coordinate position system. The value is a short integer. In order to determine the true position it is necessary to divide by 32768 (8000 hex) and multiply by the maximum range.

#### **s[3][3]**

This is a 3x3 array of values. Each value is delivered as a 16-bit signed integer. In order to convert each value to a number in the range +1 -> -1 it is necessary to divide each value by 32768 (8000 hex). The signed integer has a range of +32767 -> -32768 which when divided by 32768 will give a fractional number in the range 0.99997 -> -1.00000.

### Requirements

**Windows NT/2000:** Requires Windows 2000 or later.

**Header:** Declared in ATC3DGm.h

**Library:** Use ATC3DGm.lib

### See Also

SHORT\_POSITION\_RECORD, SHORT\_MATRIX\_RECORD

## SHORT\_POSITION\_QUATERNION\_RECORD

The **SHORT\_POSITION\_QUATERNION\_RECORD** structure contains position and quaternion information in 16-bit signed integer format.

```
typedef struct tagSHORT_POSITION_QUATERNION{
    short      x;
    short      y;
    short      z;
    short      s[3][3];
} SHORT_POSITION_QUATERNION_RECORD, *PSHORT_POSITION_QUATERNION_RECORD;
```

### Members

#### **x**

This is the x position value of a 3-axis coordinate position system. The value is a short integer. In order to determine the true position it is necessary to divide by 32768 (8000 hex) and multiply by the maximum range.

#### **y**

This is the y position value of a 3-axis coordinate position system. The value is a short integer. In order to determine the true position it is necessary to divide by 32768 (8000 hex) and multiply by the maximum range.

#### **z**

This is the z position value of a 3-axis coordinate position system. The value is a short integer. In order to determine the true position it is necessary to divide by 32768 (8000 hex) and multiply by the maximum range.

#### **q[4]**

This is an array of 4 quaternion values. Each value is delivered as a 16-bit signed integer. In order to convert each value to a number in the range +1 -> -1 it is necessary to divide each value by 32768 (8000 hex). The signed integer has a range of +32767 -> -32768 which when divided by 32768 will give a fractional number in the range 0.99997 -> -1.00000.

### Requirements

**Windows NT/2000:** Requires Windows 2000 or later.

**Header:** Declared in ATC3DGm.h

**Library:** Use ATC3DGm.lib

### See Also

SHORT\_POSITION\_RECORD, SHORT\_QUATERNIONS\_RECORD

## DOUBLE\_POSITION\_RECORD

The **DOUBLE\_POSITION\_RECORD** structure contains position information only in double floating point format.

```
typedef struct tagDOUBLE_POSITION{  
    double          x;  
    double          y;  
    double          z;  
} DOUBLE_POSITION_RECORD, *PDOUBLE_POSITION_RECORD;
```

### Members

#### **x**

This is the x position value of a 3-axis coordinate position system. The position will be reported in inches unless the SYSTEM\_PARAMETER\_TYPE, METRIC has been set to true in which case the position will be reported in millimeters.

#### **y**

This is the y position value of a 3-axis coordinate position system. The position will be reported in inches unless the SYSTEM\_PARAMETER\_TYPE, METRIC has been set to true in which case the position will be reported in millimeters.

#### **z**

This is the z position value of a 3-axis coordinate position system. The position will be reported in inches unless the SYSTEM\_PARAMETER\_TYPE, METRIC has been set to true in which case the position will be reported in millimeters.

### Remarks

*The X, Y and Z values vary between the double precision floating point equivalent of +/- maximum range.*



## Requirements

**Windows NT/2000:** Requires Windows 2000 or later.

**Header:** Declared in ATC3DGm.h

**Library:** Use ATC3DGm.lib

## See Also

## DOUBLE\_ANGLES\_RECORD

The **DOUBLE\_ANGLES\_RECORD** structure contains Euler angles information only in double floating point format.

```
typedef struct tagDOUBLE_ANGLES{
    double          a;
    double          e;
    double          r;
} DOUBLE_ANGLES_RECORD, *PDOUBLE_ANGLES_RECORD;
```

### Members

#### **a**

This value is the azimuth angle of the sensor. The value is reported in degrees with a range of +179.995 to -180.000 degrees.

#### **e**

This value is the elevation angle of the sensor. The value is reported in degrees with a range of +89.995 to -90.000 degrees.

#### **r**

This value is the roll angle of the sensor. The value is reported in degrees with a range of +179.995 to -180.000 degrees.

### Remarks

In the DOUBLE ANGLES mode, the Tracker outputs the orientation angles of the sensor with respect to the transmitter using double precision floating point format. The orientation angles are defined as rotations about the Z, Y, and X axes of the sensor. These angles are called Zang, Yang, and Xang or, in Euler angle nomenclature, Azimuth, Elevation, and Roll.

Zang (Azimuth) takes on values between the binary equivalent of +/- 180 degrees. Yang (Elevation) takes on values between +/- 90 degrees, and Xang (Roll) takes on values between +/- 180 degrees. As Yang (Elevation) approaches +/- 90 degrees, the Zang (Azimuth) and Xang (Roll) become very noisy and exhibit large errors. At 90 degrees the Zang (Azimuth) and Xang (Roll) become undefined. This behavior is not a limitation of the Tracker - it is an inherent characteristic of these Euler angles. If you need a stable representation of the sensor orientation at high Elevation angles, use the MATRIX output mode.

The scaling of all angles is full scale = 180 degrees. That is, +179.99 deg = 7FFF Hex, 0 deg = 0 Hex, -180.00 deg = 8000 Hex.

Angle information is 0 when sensor saturation occurs.

### Requirements

**Windows NT/2000:** Requires Windows 2000 or later.

**Header:** Declared in ATC3DGm.h

**Library:** Use ATC3DGm.lib

### See Also

## DOUBLE\_MATRIX\_RECORD

The **DOUBLE\_MATRIX\_RECORD** structure contains only a 3x3 rotation matrix in double floating point format.

```
typedef struct tagDOUBLE_MATRIX{
    double          s[3][3];
} DOUBLE_MATRIX_RECORD, *PDOUBLE_MATRIX_RECORD;
```

### Members

#### **s[3][3]**

This is a 3x3 array of values. Each value is in the range +0.99997 to -1.00000

### Remarks

The MATRIX mode outputs the 9 elements of the rotation matrix that define the orientation of the sensor's X, Y, and Z axes with respect to the transmitter's X, Y, and Z axes using double precision floating point format. If you want a three-dimensional image to follow the rotation of the sensor, you must multiply your image coordinates by this output matrix.

The nine elements of the output matrix are defined generically by:

**M(1,1)    M(1,2)    M(1,3)**

**M(2,1)    M(2,2)    M(2,3)**

**M(3,1)    M(3,2)    M(3,3)**

Or in terms of the rotation angles about each axis where **Z = Zang**, **Y = Yang** and **X = Xang**:

Or in Euler angle notation, where **R = Roll**, **E = Elevation**, **A = Azimuth**:

<b>COS(E)*COS(A)</b>	<b>COS(E)*SIN(A)</b>	<b>-SIN(E)</b>
<b>-(COS(R)*SIN(A))</b>	<b>(COS(R)*COS(A))</b>	
<b>+ (SIN(R)*SIN(E)*COS(A))</b>	<b>+ (SIN(R)*SIN(E)*SIN(A))</b>	<b>SIN(R)*COS(E)</b>
<b>(SIN(R)*SIN(A))</b>	<b>-(SIN(R)*COS(A))</b>	
<b>+ (COS(R)*SIN(E)*COS(A))</b>	<b>+ (COS(R)*SIN(E)*SIN(A))</b>	<b>COS(R)*COS(E)</b>

The matrix elements take values between the binary equivalents of +.99996 and -1.0.  
Element scaling is +.99996 = 7FFF Hex, 0 = 0 Hex, and -1.0 = 8000 Hex.

Matrix information is 0 when sensor saturation occurs.

**Requirements**

**Windows NT/2000:** Requires Windows 2000 or later.

**Header:** Declared in ATC3DGm.h

**Library:** Use ATC3DGm.lib

**See Also**

## DOUBLE\_QUATERNIONS\_RECORD

The **DOUBLE\_QUATERNIONS\_RECORD** structure contains only an array of 4 quaternion values in double floating point format.

```
typedef struct tagDOUBLE_QUATERNIONS{
    double      q[4];
} DOUBLE_QUATERNIONS_RECORD, *PDOUBLE_QUATERNIONS_RECORD;
```

### Members

#### q[4]

This is an array of 4 quaternion values. Each value is in the range +0.99997 to -1.00000

### Remarks

In the QUATERNION mode, the Tracker outputs the four quaternion parameters that describe the orientation of the sensor with respect to the transmitter using double precision floating point format. The quaternions,  $q_0$ ,  $q_1$ ,  $q_2$ , and  $q_3$  where  $q_0$  is the scalar component, have been extracted from the MATRIX output using the algorithm described in "Quaternion from Rotation Matrix" by Stanley W. Shepperd, *Journal of Guidance and Control*, Vol. 1, May-June 1978, pp. 223-4. Shepperd, when defining his quaternion conversion, uses the convention that  $q = \cos(\alpha/2) - \sin(\alpha/2)*u$ , where  $\alpha$  is the angle of rotation and  $u$  is the axis of rotation, i.e., the vector part of the quaternion. Some sources use the convention that  $q = \cos(\alpha/2) + \sin(\alpha/2)*u$ . To convert from one convention to the other, invert the sign on the vector portion of the quaternion:  $q_1$ ,  $q_2$ ,  $q_3$ .

### Requirements

**Windows NT/2000:** Requires Windows 2000 or later.

**Header:** Declared in ATC3DGm.h

**Library:** Use ATC3DGm.lib

### See Also

## DOUBLE\_POSITION\_ANGLES\_RECORD

The **DOUBLE\_POSITION\_ANGLES\_RECORD** structure contains position and Euler angle information in double floating point format.

```
typedef struct tagDOUBLE_POSITION_ANGLES{
    double          x;
    double          y;
    double          z;
    double          a;
    double          e;
    double          r;
} DOUBLE_POSITION_ANGLES_RECORD, *PDOUBLE_POSITION_ANGLES_RECORD;
```

### Members

#### **x**

This is the x position value of a 3-axis coordinate position system. The position will be reported in inches unless the SYSTEM\_PARAMETER\_TYPE, METRIC has been set to true in which case the position will be reported in millimeters.

#### **y**

This is the y position value of a 3-axis coordinate position system. The position will be reported in inches unless the SYSTEM\_PARAMETER\_TYPE, METRIC has been set to true in which case the position will be reported in millimeters.

#### **z**

This is the z position value of a 3-axis coordinate position system. The position will be reported in inches unless the SYSTEM\_PARAMETER\_TYPE, METRIC has been set to true in which case the position will be reported in millimeters.

#### **a**

This value is the azimuth angle of the sensor. The value is reported in degrees with a range of +179.995 to -180.000 degrees.

#### **e**

This value is the elevation angle of the sensor. The value is reported in degrees with a range of +89.995 to -90.000 degrees.

#### **r**

This value is the roll angle of the sensor. The value is reported in degrees with a range of +179.995 to -180.000 degrees.

### Requirements

**Windows NT/2000:** Requires Windows 2000 or later.

**Header:** Declared in ATC3DGm.h

**Library:** Use ATC3DGm.lib

### See Also

DOUBLE\_POSITION\_RECORD, DOUBLE\_ANGLES\_RECORD

## DOUBLE\_POSITION\_MATRIX\_RECORD

The **DOUBLE\_POSITION\_MATRIX\_RECORD** structure contains position and matrix information in double floating point format.

```
typedef struct tagDOUBLE_POSITION_MATRIX{
    double          x;
    double          y;
    double          z;
    double          s[3][3];
} DOUBLE_POSITION_MATRIX_RECORD, *PDOUBLE_POSITION_MATRIX_RECORD;
```

### Members

#### **x**

This is the x position value of a 3-axis coordinate position system. The position will be reported in inches unless the SYSTEM\_PARAMETER\_TYPE, METRIC has been set to true in which case the position will be reported in millimeters.

#### **y**

This is the y position value of a 3-axis coordinate position system. The position will be reported in inches unless the SYSTEM\_PARAMETER\_TYPE, METRIC has been set to true in which case the position will be reported in millimeters.

#### **z**

This is the z position value of a 3-axis coordinate position system. The position will be reported in inches unless the SYSTEM\_PARAMETER\_TYPE, METRIC has been set to true in which case the position will be reported in millimeters.

#### **s[3][3]**

This is a 3x3 array of values. Each value is in the range +0.99997 to -1.00000

### Requirements

**Windows NT/2000:** Requires Windows 2000 or later.

**Header:** Declared in ATC3DGm.h

**Library:** Use ATC3DGm.lib

### See Also

DOUBLE\_POSITION\_RECORD, DOUBLE\_MATRIX\_RECORD

## DOUBLE\_POSITION\_QUATERNION\_RECORD

The **DOUBLE\_POSITION\_QUATERNION\_RECORD** structure contains position and quaternion information in double floating point format.

```
typedef struct tagDOUBLE_POSITION_QUATERNION{
    double          x;
    double          y;
    double          z;
    double          q[4];
} DOUBLE_POSITION_QUATERNION_RECORD, *PDOUBLE_POSITION_QUATERNION_RECORD;
```

### Members

#### **x**

This is the x position value of a 3-axis coordinate position system. The position will be reported in inches unless the SYSTEM\_PARAMETER\_TYPE, METRIC has been set to true in which case the position will be reported in millimeters.

#### **y**

This is the y position value of a 3-axis coordinate position system. The position will be reported in inches unless the SYSTEM\_PARAMETER\_TYPE, METRIC has been set to true in which case the position will be reported in millimeters.

#### **z**

This is the z position value of a 3-axis coordinate position system. The position will be reported in inches unless the SYSTEM\_PARAMETER\_TYPE, METRIC has been set to true in which case the position will be reported in millimeters.

#### **q[4]**

This is an array of 4 quaternion values. Each value is in the range +0.99997 to -1.00000

### Requirements

**Windows NT/2000:** Requires Windows 2000 or later.

**Header:** Declared in ATC3DGm.h

**Library:** Use ATC3DGm.lib

### See Also

DOUBLE\_POSITION\_RECORD, DOUBLE\_QUATERNIONS\_RECORD



## DOUBLE\_POSITION\_TIME\_STAMP\_RECORD

The **DOUBLE\_POSITION\_TIME\_STAMP\_RECORD** structure contains position information only in double floating point format.

```
typedef struct tagDOUBLE_POSITION_TIME_STAMP{
    double          x;
    double          y;
    double          z;
    double          time;
} DOUBLE_POSITION_TIME_STAMP_RECORD, *PDOUBLE_POSITION_TIME_STAMP_RECORD;
```

### Members

#### **x**

This is the x position value of a 3-axis coordinate position system. The position will be reported in inches unless the SYSTEM\_PARAMETER\_TYPE, METRIC has been set to true in which case the position will be reported in millimeters.

#### **y**

This is the y position value of a 3-axis coordinate position system. The position will be reported in inches unless the SYSTEM\_PARAMETER\_TYPE, METRIC has been set to true in which case the position will be reported in millimeters.

#### **z**

This is the z position value of a 3-axis coordinate position system. The position will be reported in inches unless the SYSTEM\_PARAMETER\_TYPE, METRIC has been set to true in which case the position will be reported in millimeters.

#### **time**

The time variable is the time stamp for the data record and is returned as a **double** value. The integer portion of the variable represents the number of elapsed seconds since midnight, Jan 1, 1970, UTC, and is the standard way of representing time and date. If this is cast as a time\_t structure, it can be converted into a date and time string using ctime() for example. The fractional part of the time variable represents fractions of a second.

### Requirements

**Windows NT/2000:** Requires Windows 2000 or later.

**Header:** Declared in ATC3DGm.h

**Library:** Use ATC3DGm.lib

### See Also

DOUBLE\_POSITION\_RECORD

## DOUBLE\_ANGLES\_TIME\_STAMP\_RECORD

The **DOUBLE\_ANGLES\_TIME\_STAMP\_RECORD** structure contains Euler angles information only in double floating point format.

```
typedef struct tagDOUBLE_ANGLES_TIME_STAMP{
    double          a;
    double          e;
    double          r;
    double          time;
} DOUBLE_ANGLES_TIME_STAMP_RECORD, *PDOUBLE_ANGLES_TIME_STAMP_RECORD;
```

### Members

#### **a**

This value is the azimuth angle of the sensor. The value is reported in degrees with a range of +179.995 to -180.000 degrees.

#### **e**

This value is the elevation angle of the sensor. The value is reported in degrees with a range of +89.995 to -90.000 degrees.

#### **r**

This value is the roll angle of the sensor. The value is reported in degrees with a range of +179.995 to -180.000 degrees.

#### **time**

The time variable is the time stamp for the data record and is returned as a **double** value. The integer portion of the variable represents the number of elapsed seconds since midnight, Jan 1, 1970, UTC, and is the standard way of representing time and date. If this is cast as a time\_t structure, it can be converted into a date and time string using ctime() for example. The fractional part of the time variable represents fractions of a second.

### Requirements

**Windows NT/2000:** Requires Windows 2000 or later.

**Header:** Declared in ATC3DGm.h

**Library:** Use ATC3DGm.lib

### See Also

DOUBLE\_ANGLES\_RECORD

## DOUBLE\_MATRIX\_TIME\_STAMP\_RECORD

The **DOUBLE\_MATRIX\_TIME\_STAMP\_RECORD** structure contains only a 3x3 rotation matrix in double floating point format.

```
typedef struct tagDOUBLE_MATRIX_TIME_STAMP{  
    double          s[3][3];  
    double          time;  
} DOUBLE_MATRIX_TIME_STAMP_RECORD, *PDOUBLE_MATRIX_TIME_STAMP_RECORD;
```

### Members

#### **s[3][3]**

This is a 3x3 array of values. Each value is in the range +0.99997 to -1.00000

#### **time**

The time variable is the time stamp for the data record and is returned as a **double** value. The integer portion of the variable represents the number of elapsed seconds since midnight, Jan 1, 1970, UTC, and is the standard way of representing time and date. If this is cast as a `time_t` structure, it can be converted into a date and time string using `ctime()` for example. The fractional part of the time variable represents fractions of a second.

### Requirements

**Windows NT/2000:** Requires Windows 2000 or later.

**Header:** Declared in `ATC3DGm.h`

**Library:** Use `ATC3DGm.lib`

### See Also

`DOUBLE_MATRIX_RECORD`

## DOUBLE\_QUATERNIONS\_TIME\_STAMP\_RECORD

The **DOUBLE\_QUATERNIONS\_TIME\_STAMP\_RECORD** structure contains only an array of 4 quaternion values in double floating point format.

```
typedef struct tagDOUBLE_QUATERNIONS_TIME_STAMP{
    double          q[4];
    double          time;
} DOUBLE_QUATERNIONS_TIME_STAMP_RECORD, *PDOUBLE_QUATERNIONS_TIME_STAMP_RECORD;
```

### Members

#### q[4]

This is an array of 4 quaternion values. Each value is in the range +0.99997 to -1.00000

#### time

The time variable is the time stamp for the data record and is returned as a **double** value. The integer portion of the variable represents the number of elapsed seconds since midnight, Jan 1, 1970, UTC, and is the standard way of representing time and date. If this is cast as a `time_t` structure, it can be converted into a date and time string using `ctime()` for example. The fractional part of the time variable represents fractions of a second.

### Requirements

**Windows NT/2000:** Requires Windows 2000 or later.

**Header:** Declared in ATC3DGm.h

**Library:** Use ATC3DGm.lib

### See Also

DOUBLE\_QUATERNIONS\_RECORD

## DOUBLE\_POSITION\_ANGLES\_TIME\_STAMP\_RECORD

The **DOUBLE\_POSITION\_ANGLES\_TIME\_STAMP\_RECORD** structure contains position and Euler angle information in double floating point format.

```
typedef struct tagDOUBLE_POSITION_ANGLES_TIME_STAMP{
    double          x;
    double          y;
    double          z;
    double          a;
    double          e;
    double          r;
    double          time;
} DOUBLE_POSITION_ANGLES_TIME_STAMP_RECORD, *PDOUBLE_POSITION_ANGLES_TIME_STAMP_RECORD;
```

### Members

#### **x**

This is the x position value of a 3-axis coordinate position system. The position will be reported in inches unless the SYSTEM\_PARAMETER\_TYPE, METRIC has been set to true in which case the position will be reported in millimeters.

#### **y**

This is the y position value of a 3-axis coordinate position system. The position will be reported in inches unless the SYSTEM\_PARAMETER\_TYPE, METRIC has been set to true in which case the position will be reported in millimeters.

#### **z**

This is the z position value of a 3-axis coordinate position system. The position will be reported in inches unless the SYSTEM\_PARAMETER\_TYPE, METRIC has been set to true in which case the position will be reported in millimeters.

#### **a**

This value is the azimuth angle of the sensor. The value is reported in degrees with a range of +179.995 to -180.000 degrees.

#### **e**

This value is the elevation angle of the sensor. The value is reported in degrees with a range of +89.995 to -90.000 degrees.

#### **r**

This value is the roll angle of the sensor. The value is reported in degrees with a range of +179.995 to -180.000 degrees.

#### **time**

The time variable is the time stamp for the data record and is returned as a **double** value. The integer portion of the variable represents the number of elapsed seconds since midnight, Jan 1, 1970, UTC, and is the standard way of representing time and date. If this is cast as a time\_t structure, it can be converted into a date and time string using ctime() for example. The fractional part of the time variable represents fractions of a second.

### Requirements

**Windows NT/2000:** Requires Windows 2000 or later.

**Header:** Declared in ATC3DGm.h

**Library:** Use ATC3DGm.lib

**See Also**

DOUBLE\_POSITION\_RECORD, DOUBLE\_ANGLES\_RECORD

## DOUBLE\_POSITION\_MATRIX\_TIME\_STAMP\_RECORD

The **DOUBLE\_POSITION\_MATRIX\_TIME\_STAMP\_RECORD** structure contains position and matrix information in double floating point format.

```
typedef struct tagDOUBLE_POSITION_MATRIX_TIME_STAMP{
    double          x;
    double          y;
    double          z;
    double          s[3][3];
    double          time;
} DOUBLE_POSITION_MATRIX_TIME_STAMP_RECORD, *PDOUBLE_POSITION_MATRIX_TIME_STAMP_RECORD;
```

### Members

#### **x**

This is the x position value of a 3-axis coordinate position system. The position will be reported in inches unless the SYSTEM\_PARAMETER\_TYPE, METRIC has been set to true in which case the position will be reported in millimeters.

#### **y**

This is the y position value of a 3-axis coordinate position system. The position will be reported in inches unless the SYSTEM\_PARAMETER\_TYPE, METRIC has been set to true in which case the position will be reported in millimeters.

#### **z**

This is the z position value of a 3-axis coordinate position system. The position will be reported in inches unless the SYSTEM\_PARAMETER\_TYPE, METRIC has been set to true in which case the position will be reported in millimeters.

#### **s[3][3]**

This is a 3x3 array of values. Each value is in the range +0.99997 to -1.00000

#### **time**

The time variable is the time stamp for the data record and is returned as a **double** value. The integer portion of the variable represents the number of elapsed seconds since midnight, Jan 1, 1970, UTC, and is the standard way of representing time and date. If this is cast as a time\_t structure, it can be converted into a date and time string using ctime() for example. The fractional part of the time variable represents fractions of a second.

### Requirements

**Windows NT/2000:** Requires Windows 2000 or later.

**Header:** Declared in ATC3DGm.h

**Library:** Use ATC3DGm.lib

### See Also

DOUBLE\_POSITION\_RECORD, DOUBLE\_MATRIX\_RECORD

## DOUBLE\_POSITION\_QUATERNION\_TIME\_STAMP\_RECORD

The **DOUBLE\_POSITION\_QUATERNION\_TIME\_STAMP\_RECORD** structure contains position and quaternion information in double floating point format.

```
typedef struct tagDOUBLE_POSITION_QUATERNION_TIME_STAMP{
    double          x;
    double          y;
    double          z;
    double          q[4];
    double          time;
}                DOUBLE_POSITION_QUATERNION_TIME_STAMP_RECORD,
*PDOUBLE_POSITION_QUATERNION_TIME_STAMP_RECORD;
```

### Members

#### **x**

This is the x position value of a 3-axis coordinate position system. The position will be reported in inches unless the SYSTEM\_PARAMETER\_TYPE, METRIC has been set to true in which case the position will be reported in millimeters.

#### **y**

This is the y position value of a 3-axis coordinate position system. The position will be reported in inches unless the SYSTEM\_PARAMETER\_TYPE, METRIC has been set to true in which case the position will be reported in millimeters.

#### **z**

This is the z position value of a 3-axis coordinate position system. The position will be reported in inches unless the SYSTEM\_PARAMETER\_TYPE, METRIC has been set to true in which case the position will be reported in millimeters.

#### **q[4]**

This is an array of 4 quaternion values. Each value is in the range +0.99997 to -1.00000

#### **time**

The time variable is the time stamp for the data record and is returned as a **double** value. The integer portion of the variable represents the number of elapsed seconds since midnight, Jan 1, 1970, UTC, and is the standard way of representing time and date. If this is cast as a time\_t structure, it can be converted into a date and time string using ctime() for example. The fractional part of the time variable represents fractions of a second.

### Requirements

**Windows NT/2000:** Requires Windows 2000 or later.

**Header:** Declared in ATC3DGm.h

**Library:** Use ATC3DGm.lib

### See Also

DOUBLE\_POSITION\_RECORD, DOUBLE\_QUATERNIONS\_RECORD



## DOUBLE\_POSITION\_TIME\_Q\_RECORD

The **DOUBLE\_POSITION\_TIME\_Q\_RECORD** structure contains position, timestamp and distortion information in double floating point format.

```
typedef struct tagDOUBLE_POSITION_TIME_Q{
    double          x;
    double          y;
    double          z;
    double          time;
    USHORT          quality;
} DOUBLE_POSITION_TIME_Q_RECORD, *PDOUBLE_POSITION_TIME_Q_RECORD;
```

### Members

#### **x**

This is the x position value of a 3-axis coordinate position system. The position will be reported in inches unless the **SYSTEM\_PARAMETER\_TYPE**, **METRIC** has been set to true in which case the position will be reported in millimeters.

#### **y**

This is the y position value of a 3-axis coordinate position system. The position will be reported in inches unless the **SYSTEM\_PARAMETER\_TYPE**, **METRIC** has been set to true in which case the position will be reported in millimeters.

#### **z**

This is the z position value of a 3-axis coordinate position system. The position will be reported in inches unless the **SYSTEM\_PARAMETER\_TYPE**, **METRIC** has been set to true in which case the position will be reported in millimeters.

#### **time**

The time variable is the time stamp for the data record and is returned as a **double** value. The integer portion of the variable represents the number of elapsed seconds since midnight, Jan 1, 1970, UTC, and is the standard way of representing time and date. If this is cast as a **time\_t** structure, it can be converted into a date and time string using **ctime()** for example. The fractional part of the time variable represents fractions of a second.

#### **quality**

The quality value is a 16-bit unsigned integer. A very small quality number indicates no or minimal position and orientation errors due to distortion of the transmitter field depending on how sensitive you have set the error indicator. A large quality number indicates maximum error for the sensitivity level selected.

### Requirements

**Windows NT/2000:** Requires Windows 2000 or later.

**Header:** Declared in ATC3DGm.h

**Library:** Use ATC3DGm.lib

**See Also**

DOUBLE\_POSITION\_RECORD

## DOUBLE\_ANGLES\_TIME\_Q\_RECORD

The **DOUBLE\_ANGLES\_TIME\_Q\_RECORD** structure contains Euler angles, timestamp and distortion information in double floating point format.

```
typedef struct tagDOUBLE_ANGLES_TIME_Q{
    double          a;
    double          e;
    double          r;
    double          time;
    USHORT          quality;
} DOUBLE_ANGLES_TIME_Q_RECORD, *PDOUBLE_ANGLES_TIME_Q_RECORD;
```

### Members

#### **a**

This value is the azimuth angle of the sensor. The value is reported in degrees with a range of +179.995 to -180.000 degrees.

#### **e**

This value is the elevation angle of the sensor. The value is reported in degrees with a range of +89.995 to -90.000 degrees.

#### **r**

This value is the roll angle of the sensor. The value is reported in degrees with a range of +179.995 to -180.000 degrees.

#### **time**

The time variable is the time stamp for the data record and is returned as a **double** value. The integer portion of the variable represents the number of elapsed seconds since midnight, Jan 1, 1970, UTC, and is the standard way of representing time and date. If this is cast as a time\_t structure, it can be converted into a date and time string using ctime() for example. The fractional part of the time variable represents fractions of a second.

#### **quality**

The quality value is a 16-bit unsigned integer. A very small quality number indicates no or minimal position and orientation errors due to distortion of the transmitter field depending on how sensitive you have set the error indicator. A large quality number indicates maximum error for the sensitivity level selected.

### Requirements

**Windows NT/2000:** Requires Windows 2000 or later.

**Header:** Declared in ATC3DGm.h

**Library:** Use ATC3DGm.lib

### See Also

[DOUBLE\\_ANGLES\\_RECORD](#)

## DOUBLE\_MATRIX\_TIME\_Q\_RECORD

The **DOUBLE\_MATRIX\_TIME\_Q\_RECORD** structure contains a 3 x 3 rotation matrix, timestamp and distortion information in double floating point format.

```
typedef struct tagDOUBLE_MATRIX_TIME_Q{
    double          s[3][3];
    double          time;
    USHORT          quality;
} DOUBLE_MATRIX_TIME_Q_RECORD, *PDOUBLE_MATRIX_TIME_Q_RECORD;
```

### Members

#### s[3][3]

This is a 3x3 array of values. Each value is in the range +0.99997 to -1.00000

#### time

The time variable is the time stamp for the data record and is returned as a **double** value. The integer portion of the variable represents the number of elapsed seconds since midnight, Jan 1, 1970, UTC, and is the standard way of representing time and date. If this is cast as a time\_t structure, it can be converted into a date and time string using ctime() for example. The fractional part of the time variable represents fractions of a second.

#### quality

The quality value is a 16-bit unsigned integer. A very small quality number indicates no or minimal position and orientation errors due to distortion of the transmitter field depending on how sensitive you have set the error indicator. A large quality number indicates maximum error for the sensitivity level selected.

### Requirements

**Windows NT/2000:** Requires Windows 2000 or later.

**Header:** Declared in ATC3DGm.h

**Library:** Use ATC3DGm.lib

### See Also

DOUBLE\_MATRIX\_RECORD

## DOUBLE\_QUATERNIONS\_TIME\_Q\_RECORD

The **DOUBLE\_QUATERNIONS\_TIME\_Q\_RECORD** structure contains an array of 4 quaternion values, timestamp and distortion information in double floating point format.

```
typedef struct tagDOUBLE_QUATERNIONS_TIME_Q{
    double          q[4];
    double          time;
    USHORT          quality;
} DOUBLE_QUATERNIONS_TIME_Q_RECORD, *PDOUBLE_QUATERNIONS_TIME_Q_RECORD;
```

### Members

#### q[4]

This is an array of 4 quaternion values. Each value is in the range +0.99997 to -1.00000

#### time

The time variable is the time stamp for the data record and is returned as a **double** value. The integer portion of the variable represents the number of elapsed seconds since midnight, Jan 1, 1970, UTC, and is the standard way of representing time and date. If this is cast as a time\_t structure, it can be converted into a date and time string using ctime() for example. The fractional part of the time variable represents fractions of a second.

#### quality

The quality value is a 16-bit unsigned integer. A very small quality number indicates no or minimal position and orientation errors due to distortion of the transmitter field depending on how sensitive you have set the error indicator. A large quality number indicates maximum error for the sensitivity level selected.

### Requirements

**Windows NT/2000:** Requires Windows 2000 or later.

**Header:** Declared in ATC3DGm.h

**Library:** Use ATC3DGm.lib

### See Also

[DOUBLE\\_QUATERNIONS\\_RECORD](#)

## DOUBLE\_POSITION\_ANGLES\_TIME\_Q\_RECORD

The **DOUBLE\_POSITION\_ANGLES\_TIME\_Q\_RECORD** structure contains position Euler angle, timestamp and distortion information in double floating point format.

```
typedef struct tagDOUBLE_POSITION_ANGLES_TIME_Q{
    double          x;
    double          y;
    double          z;
    double          a;
    double          e;
    double          r;
    double          time;
    USHORT          quality;
} DOUBLE_POSITION_ANGLES_TIME_Q_RECORD, *PDOUBLE_POSITION_ANGLES_TIME_Q_RECORD;
```

### Members

#### x

This is the x position value of a 3-axis coordinate position system. The position will be reported in inches unless the SYSTEM\_PARAMETER\_TYPE, METRIC has been set to true in which case the position will be reported in millimeters.

#### y

This is the y position value of a 3-axis coordinate position system. The position will be reported in inches unless the SYSTEM\_PARAMETER\_TYPE, METRIC has been set to true in which case the position will be reported in millimeters.

#### z

This is the z position value of a 3-axis coordinate position system. The position will be reported in inches unless the SYSTEM\_PARAMETER\_TYPE, METRIC has been set to true in which case the position will be reported in millimeters.

#### a

This value is the azimuth angle of the sensor. The value is reported in degrees with a range of +179.995 to -180.000 degrees.

#### e

This value is the elevation angle of the sensor. The value is reported in degrees with a range of +89.995 to -90.000 degrees.

#### r

This value is the roll angle of the sensor. The value is reported in degrees with a range of +179.995 to -180.000 degrees.

#### time

The time variable is the time stamp for the data record and is returned as a **double** value. The integer portion of the variable represents the number of elapsed seconds since midnight, Jan 1, 1970, UTC, and is the standard way of representing time and date. If this is cast as a time\_t structure, it can be converted into a date and time string using ctime() for example. The fractional part of the time variable represents fractions of a second.

**quality**

The quality value is a 16-bit unsigned integer. A very small quality number indicates no or minimal position and orientation errors due to distortion of the transmitter field depending on how sensitive you have set the error indicator. A large quality number indicates maximum error for the sensitivity level selected.

**Requirements**

**Windows NT/2000:** Requires Windows 2000 or later.

**Header:** Declared in ATC3DGm.h

**Library:** Use ATC3DGm.lib

**See Also**

DOUBLE\_POSITION\_RECORD, DOUBLE\_ANGLES\_RECORD

## DOUBLE\_POSITION\_MATRIX\_TIME\_Q\_RECORD

The **DOUBLE\_POSITION\_MATRIX\_TIME\_Q\_RECORD** structure contains position, matrix, timestamp and distortion information in double floating point format.

```
typedef struct tagDOUBLE_POSITION_MATRIX_TIME_Q{
    double          x;
    double          y;
    double          z;
    double          s[3][3];
    double          time;
    USHORT          quality;
} DOUBLE_POSITION_MATRIX_TIME_Q_RECORD, *PDOUBLE_POSITION_MATRIX_TIME_Q_RECORD;
```

### Members

#### **x**

This is the x position value of a 3-axis coordinate position system. The position will be reported in inches unless the SYSTEM\_PARAMETER\_TYPE, METRIC has been set to true in which case the position will be reported in millimeters.

#### **y**

This is the y position value of a 3-axis coordinate position system. The position will be reported in inches unless the SYSTEM\_PARAMETER\_TYPE, METRIC has been set to true in which case the position will be reported in millimeters.

#### **z**

This is the z position value of a 3-axis coordinate position system. The position will be reported in inches unless the SYSTEM\_PARAMETER\_TYPE, METRIC has been set to true in which case the position will be reported in millimeters.

#### **s[3][3]**

This is a 3x3 array of values. Each value is in the range +0.99997 to -1.00000

#### **time**

The time variable is the time stamp for the data record and is returned as a **double** value. The integer portion of the variable represents the number of elapsed seconds since midnight, Jan 1, 1970, UTC, and is the standard way of representing time and date. If this is cast as a time\_t structure, it can be converted into a date and time string using ctime() for example. The fractional part of the time variable represents fractions of a second.

#### **quality**

The quality value is a 16-bit unsigned integer. A very small quality number indicates no or minimal position and orientation errors due to distortion of the transmitter field depending on how sensitive you have set the error indicator. A large quality number indicates maximum error for the sensitivity level selected.

### Requirements

**Windows NT/2000:** Requires Windows 2000 or later.

**Header:** Declared in ATC3DGm.h

**Library:** Use ATC3DGm.lib



**See Also**

DOUBLE\_POSITION\_RECORD, DOUBLE\_MATRIX\_RECORD

## DOUBLE\_POSITION\_QUATERNION\_TIME\_Q\_RECORD

The **DOUBLE\_POSITION\_QUATERNION\_TIME\_Q\_RECORD** structure contains position, quaternion, timestamp and distortion information in double floating point format.

```
typedef struct tagDOUBLE_POSITION_QUATERNION_TIME_Q{
    double          x;
    double          y;
    double          z;
    double          q[4];
    double          time;
    USHORT          quality;
} DOUBLE_POSITION_QUATERNION_TIME_Q_RECORD, *PDOUBLE_POSITION_QUATERNION_TIME_Q_RECORD;
```

### Members

#### **x**

This is the x position value of a 3-axis coordinate position system. The position will be reported in inches unless the SYSTEM\_PARAMETER\_TYPE, METRIC has been set to true in which case the position will be reported in millimeters.

#### **y**

This is the y position value of a 3-axis coordinate position system. The position will be reported in inches unless the SYSTEM\_PARAMETER\_TYPE, METRIC has been set to true in which case the position will be reported in millimeters.

#### **z**

This is the z position value of a 3-axis coordinate position system. The position will be reported in inches unless the SYSTEM\_PARAMETER\_TYPE, METRIC has been set to true in which case the position will be reported in millimeters.

#### **q[4]**

This is an array of 4 quaternion values. Each value is in the range +0.99997 to -1.00000

#### **time**

The time variable is the time stamp for the data record and is returned as a **double** value. The integer portion of the variable represents the number of elapsed seconds since midnight, Jan 1, 1970, UTC, and is the standard way of representing time and date. If this is cast as a time\_t structure, it can be converted into a date and time string using ctime() for example. The fractional part of the time variable represents fractions of a second.

#### **quality**

The quality value is a 16-bit unsigned integer. A very small quality number indicates no or minimal position and orientation errors due to distortion of the transmitter field depending on how sensitive you have set the error indicator. A large quality number indicates maximum error for the sensitivity level selected.

### Requirements

**Windows NT/2000:** Requires Windows 2000 or later.

**Header:** Declared in ATC3DGm.h

**Library:** Use ATC3DGm.lib

**See Also**

DOUBLE\_POSITION\_RECORD, DOUBLE\_QUATERNIONS\_RECORD

## SHORT\_ALL\_RECORD

The **SHORT\_ALL\_RECORD** structure contains position, Euler angles, rotation matrix and quaternion information in 16-bit signed integer format.

```
typedef struct tagSHORT_ALL{
    short      x;
    short      y;
    short      z;
    short      a;
    short      e;
    short      r;
    short      s[3][3];
    short      q[4];
} SHORT_ALL_RECORD, *PSHORT_ALL_RECORD;
```

### Members

#### **x**

This is the x position value of a 3-axis coordinate position system. The value is a short integer. In order to determine the true position it is necessary to divide by 32768 (8000 hex) and multiply by the maximum range.

#### **y**

This is the y position value of a 3-axis coordinate position system. The value is a short integer. In order to determine the true position it is necessary to divide by 32768 (8000 hex) and multiply by the maximum range.

#### **z**

This is the z position value of a 3-axis coordinate position system. The value is a short integer. In order to determine the true position it is necessary to divide by 32768 (8000 hex) and multiply by the maximum range.

#### **a**

This value is the azimuth angle of the sensor. The value is a short integer. In order to determine the true angle it is necessary to divide by 32768 (8000 hex) and multiply by 180 degrees.

#### **e**

This value is the elevation angle of the sensor. The value is a short integer. In order to determine the true angle it is necessary to divide by 32768 (8000 hex) and multiply by 180 degrees.

#### **r**

This value is the roll angle of the sensor. The value is a short integer. In order to determine the true angle it is necessary to divide by 32768 (8000 hex) and multiply by 180 degrees.

#### **s[3][3]**

This is a 3x3 array of values. Each value is delivered as a 16-bit signed integer. In order to convert each value to a number in the range +1 -> -1 it is necessary to divide each value by 32768 (8000 hex). The signed integer has a range of +32767 -> -32768 which when divided by 32768 will give a fractional number in the range 0.99997 -> -1.00000.

#### **q[4]**

This is an array of 4 quaternion values. Each value is delivered as a 16-bit signed integer. In order to convert each value to a number in the range +1 -> -1 it is necessary to divide each value by 32768 (8000 hex). The signed integer has a range of +32767 -> -32768 which when divided by 32768 will give a fractional number in the range 0.99997 -> -1.00000.

**Requirements**

**Windows NT/2000:** Requires Windows 2000 or later.

**Header:** Declared in ATC3DGm.h

**Library:** Use ATC3DGm.lib

**See Also**

SHORT\_ANGLES\_RECORD, SHORT\_MATRIX\_RECORD, SHORT\_POSITION\_RECORD, SHORT\_QUATERNIONS\_RECORD

## DOUBLE\_ALL\_RECORD

The **DOUBLE\_ALL\_RECORD** structure contains position, Euler angles, rotation matrix and quaternion information in double floating point format.

```
typedef struct tagDOUBLE_ALL{
    double          x;
    double          y;
    double          z;
    double          a;
    double          e;
    double          r;
    double          s[3][3];
    double          q[4];
} DOUBLE_ALL_RECORD, *PDOUBLE_ALL_RECORD;
```

### Members

#### **x**

This is the x position value of a 3-axis coordinate position system. The position will be reported in inches unless the SYSTEM\_PARAMETER\_TYPE, METRIC has been set to true in which case the position will be reported in millimeters.

#### **y**

This is the y position value of a 3-axis coordinate position system. The position will be reported in inches unless the SYSTEM\_PARAMETER\_TYPE, METRIC has been set to true in which case the position will be reported in millimeters.

#### **z**

This is the z position value of a 3-axis coordinate position system. The position will be reported in inches unless the SYSTEM\_PARAMETER\_TYPE, METRIC has been set to true in which case the position will be reported in millimeters.

#### **a**

This value is the azimuth angle of the sensor. The value is reported in degrees with a range of +179.995 to -180.000 degrees.

#### **e**

This value is the elevation angle of the sensor. The value is reported in degrees with a range of +89.995 to -90.000 degrees.

#### **r**

This value is the roll angle of the sensor. The value is reported in degrees with a range of +179.995 to -180.000 degrees.

#### **s[3][3]**

This is a 3x3 array of values. Each value is in the range +0.99997 to -1.00000

#### **q[4]**

This is an array of 4 quaternion values. Each value is in the range +0.99997 to -1.00000

**Requirements**

**Windows NT/2000:** Requires Windows 2000 or later.

**Header:** Declared in ATC3DGm.h

**Library:** Use ATC3DGm.lib

**See Also**

DOUBLE\_ANGLES\_RECORD, DOUBLE\_MATRIX\_RECORD, DOUBLE\_POSITION\_RECORD, DOUBLE\_QUATERNIONS\_RECORD

## DOUBLE\_ALL\_TIME\_STAMP\_RECORD

The **DOUBLE\_ALL\_TIME\_STAMP\_RECORD** structure contains position, Euler angles, rotation matrix, quaternion and timestamp information in double floating point format.

```
typedef struct tagDOUBLE_ALL_TIME_STAMP{
    double          x;
    double          y;
    double          z;
    double          a;
    double          e;
    double          r;
    double          s[3][3];
    double          q[4];
    double          time;
} DOUBLE_ALL_TIME_STAMP_RECORD, *PDOUBLE_ALL_TIME_STAMP_RECORD;
```

### Members

#### **x**

This is the x position value of a 3-axis coordinate position system. The position will be reported in inches unless the SYSTEM\_PARAMETER\_TYPE, METRIC has been set to true in which case the position will be reported in millimeters.

#### **y**

This is the y position value of a 3-axis coordinate position system. The position will be reported in inches unless the SYSTEM\_PARAMETER\_TYPE, METRIC has been set to true in which case the position will be reported in millimeters.

#### **z**

This is the z position value of a 3-axis coordinate position system. The position will be reported in inches unless the SYSTEM\_PARAMETER\_TYPE, METRIC has been set to true in which case the position will be reported in millimeters.

#### **a**

This value is the azimuth angle of the sensor. The value is reported in degrees with a range of +179.995 to -180.000 degrees.

#### **e**

This value is the elevation angle of the sensor. The value is reported in degrees with a range of +89.995 to -90.000 degrees.

#### **r**

This value is the roll angle of the sensor. The value is reported in degrees with a range of +179.995 to -180.000 degrees.

#### **s[3][3]**

This is a 3x3 array of values. Each value is in the range +0.99997 to -1.00000

#### **q[4]**

This is an array of 4 quaternion values. Each value is in the range +0.99997 to -1.00000



**time**

The time variable is the time stamp for the data record and is returned as a **double** value. The integer portion of the variable represents the number of elapsed seconds since midnight, Jan 1, 1970, UTC, and is the standard way of representing time and date. If this is cast as a `time_t` structure, it can be converted into a date and time string using `ctime()` for example. The fractional part of the time variable represents fractions of a second.

**Requirements**

**Windows NT/2000:** Requires Windows 2000 or later.

**Header:** Declared in `ATC3DGm.h`

**Library:** Use `ATC3DGm.lib`

**See Also**

`DOUBLE_ANGLES_RECORD`, `DOUBLE_MATRIX_RECORD`, `DOUBLE_POSITION_RECORD`,  
`DOUBLE_QUATERNIONS_RECORD`

## DOUBLE\_ALL\_TIME\_STAMP\_Q\_RECORD

The **DOUBLE\_ALL\_TIME\_STAMP\_Q\_RECORD** structure contains position, Euler angles, rotation matrix, quaternion, timestamp and quality information in double floating point format.

```
typedef struct tagDOUBLE_ALL_TIME_STAMP_Q{
    double          x;
    double          y;
    double          z;
    double          a;
    double          e;
    double          r;
    double          s[3][3];
    double          q[4];
    double          time;
    USHORT          quality;
} DOUBLE_ALL_TIME_STAMP_Q_RECORD, *PDOUBLE_ALL_TIME_STAMP_Q_RECORD;
```

### Members

#### **x**

This is the x position value of a 3-axis coordinate position system. The position will be reported in inches unless the SYSTEM\_PARAMETER\_TYPE, METRIC has been set to true in which case the position will be reported in millimeters.

#### **y**

This is the y position value of a 3-axis coordinate position system. The position will be reported in inches unless the SYSTEM\_PARAMETER\_TYPE, METRIC has been set to true in which case the position will be reported in millimeters.

#### **z**

This is the z position value of a 3-axis coordinate position system. The position will be reported in inches unless the SYSTEM\_PARAMETER\_TYPE, METRIC has been set to true in which case the position will be reported in millimeters.

#### **a**

This value is the azimuth angle of the sensor. The value is reported in degrees with a range of +179.995 to -180.000 degrees.

#### **e**

This value is the elevation angle of the sensor. The value is reported in degrees with a range of +89.995 to -90.000 degrees.

#### **r**

This value is the roll angle of the sensor. The value is reported in degrees with a range of +179.995 to -180.000 degrees.

#### **s[3][3]**

This is a 3x3 array of values. Each value is in the range +0.99997 to -1.00000

**q[4]**

This is an array of 4 quaternion values. Each value is in the range +0.99997 to -1.00000

**time**

The time variable is the time stamp for the data record and is returned as a **double** value. The integer portion of the variable represents the number of elapsed seconds since midnight, Jan 1, 1970, UTC, and is the standard way of representing time and date. If this is cast as a `time_t` structure, it can be converted into a date and time string using `ctime()` for example. The fractional part of the time variable represents fractions of a second.

**quality**

The quality value is a 16-bit unsigned integer. A very small quality number indicates no or minimal position and orientation errors due to distortion of the transmitter field depending on how sensitive you have set the error indicator. A large quality number indicates maximum error for the sensitivity level selected.

**Requirements**

**Windows NT/2000:** Requires Windows 2000 or later.

**Header:** Declared in `ATC3DGm.h`

**Library:** Use `ATC3DGm.lib`

**See Also**

`DOUBLE_ANGLES_RECORD`, `DOUBLE_MATRIX_RECORD`, `DOUBLE_POSITION_RECORD`,  
`DOUBLE_QUATERNIONS_RECORD`

## DOUBLE\_ALL\_TIME\_STAMP\_Q\_RAW\_RECORD

The **DOUBLE\_ALL\_TIME\_STAMP\_Q\_RAW\_RECORD** structure contains position, Euler angles, rotation matrix, quaternion, timestamp, quality and raw matrix information in double floating point format.

```
typedef struct tagDOUBLE_ALL_TIME_STAMP_Q_RAW{
    double          x;
    double          y;
    double          z;
    double          a;
    double          e;
    double          r;
    double          s[3][3];
    double          q[4];
    double          time;
    USHORT          quality;
    Double          raw[3][3];
} DOUBLE_ALL_TIME_STAMP_Q_RAW_RECORD, *PDOUBLE_ALL_TIME_STAMP_Q_RAW_RECORD;
```

### Members

#### x

This is the x position value of a 3-axis coordinate position system. The position will be reported in inches unless the SYSTEM\_PARAMETER\_TYPE, METRIC has been set to true in which case the position will be reported in millimeters.



#### Note:

This data format is not currently supported.

#### y

This is the y position value of a 3-axis coordinate position system. The position will be reported in inches unless the SYSTEM\_PARAMETER\_TYPE, METRIC has been set to true in which case the position will be reported in millimeters.

#### z

This is the z position value of a 3-axis coordinate position system. The position will be reported in inches unless the SYSTEM\_PARAMETER\_TYPE, METRIC has been set to true in which case the position will be reported in millimeters.

#### a

This value is the azimuth angle of the sensor. The value is reported in degrees with a range of +179.995 to -180.000 degrees.

#### e

This value is the elevation angle of the sensor. The value is reported in degrees with a range of +89.995 to -90.000 degrees.

#### r

This value is the roll angle of the sensor. The value is reported in degrees with a range of +179.995 to -180.000 degrees.

#### s[3][3]

This is a 3x3 array of values. Each value is in the range +0.99997 to -1.00000

**q[4]**

This is an array of 4 quaternion values. Each value is in the range +0.99997 to -1.00000

**time**

The time variable is the time stamp for the data record and is returned as a **double** value. The integer portion of the variable represents the number of elapsed seconds since midnight, Jan 1, 1970, UTC, and is the standard way of representing time and date. If this is cast as a time\_t structure, it can be converted into a date and time string using ctime() for example. The fractional part of the time variable represents fractions of a second.

**quality**

The quality value is a 16-bit unsigned integer. A very small quality number indicates no or minimal position and orientation errors due to distortion of the transmitter field depending on how sensitive you have set the error indicator. A large quality number indicates maximum error for the sensitivity level selected.

**raw[3][3]**

This is a 3x3 array of values. Each value is in the range +0.99997 to -1.00000. These values are the raw sensor values after they have been corrected for all known system error sources. This information is for factory use only.

**Requirements**

**Windows NT/2000:** Requires Windows 2000 or later.

**Header:** Declared in ATC3DGm.h

**Library:** Use ATC3DGm.lib

**See Also**

DOUBLE\_ANGLES\_RECORD, DOUBLE\_MATRIX\_RECORD, DOUBLE\_POSITION\_RECORD,  
DOUBLE\_QUATERNIONS\_RECORD

## DOUBLE\_POSITION\_ANGLES\_TIME\_Q\_BUTTON\_RECORD

The **DOUBLE\_POSITION\_ANGLES\_TIME\_Q\_BUTTON\_RECORD** structure contains position Euler angle, timestamp, distortion and button information in double floating point format.

```
typedef struct tagDOUBLE_POSITION_ANGLES_TIME_Q_BUTTON{
    double          x;
    double          y;
    double          z;
    double          a;
    double          e;
    double          r;
    double          time;
    USHORT          quality;
    USHORT          button;
}DOUBLE_POSITION_ANGLES_TIME_Q_BUTTON_RECORD,*PDOUBLE_POSITION_ANGLES_TIME_Q_BUTTON_RECORD
;
```

### Members

#### **x**

This is the x position value of a 3-axis coordinate position system. The position will be reported in inches unless the SYSTEM\_PARAMETER\_TYPE, METRIC has been set to true in which case the position will be reported in millimeters.

#### **y**

This is the y position value of a 3-axis coordinate position system. The position will be reported in inches unless the SYSTEM\_PARAMETER\_TYPE, METRIC has been set to true in which case the position will be reported in millimeters.

#### **z**

This is the z position value of a 3-axis coordinate position system. The position will be reported in inches unless the SYSTEM\_PARAMETER\_TYPE, METRIC has been set to true in which case the position will be reported in millimeters.

#### **a**

This value is the azimuth angle of the sensor. The value is reported in degrees with a range of +179.995 to -180.000 degrees.

#### **e**

This value is the elevation angle of the sensor. The value is reported in degrees with a range of +89.995 to -90.000 degrees.

#### **r**

This value is the roll angle of the sensor. The value is reported in degrees with a range of +179.995 to -180.000 degrees.

**time**

The time variable is the time stamp for the data record and is returned as a **double** value. The integer portion of the variable represents the number of elapsed seconds since midnight, Jan 1, 1970, UTC, and is the standard way of representing time and date. If this is cast as a `time_t` structure, it can be converted into a date and time string using `ctime()` for example. The fractional part of the time variable represents fractions of a second.

**quality**

The quality value is a 16-bit unsigned integer. A very small quality number indicates no or minimal position and orientation errors due to distortion of the transmitter field depending on how sensitive you have set the error indicator. A large quality number indicates maximum error for the sensitivity level selected.

**button**

The button value is a 16-bit unsigned integer that represents the state of an external contact closure that the user has been connected to the tracker (see [SWITCH](#) for connector description). When a switch is connected, a 1 in the button value indicates the contact or switch is CLOSED, and a 0 indicates the contact is OPEN. The button line is sampled and available in this data record once per transmitter axis cycle - 3 times the system measurement rate for a mid or short-range transmitter.

**Requirements**

**Windows NT/2000:** Requires Windows 2000 or later.

**Header:** Declared in ATC3DG.h

**Library:** Use ATC3DG.lib

**See Also**

DOUBLE\_POSITION\_RECORD, DOUBLE\_ANGLES\_RECORD

## DOUBLE\_POSITION\_MATRIX\_TIME\_Q\_BUTTON\_RECORD

The **DOUBLE\_POSITION\_MATRIX\_TIME\_Q\_BUTTON\_RECORD** structure contains position, matrix, timestamp, distortion and button information in double floating point format.

```
typedef struct tagDOUBLE_POSITION_MATRIX_TIME_Q_BUTTON{
    double          x;
    double          y;
    double          z;
    double          s[3][3];
    double          time;
    USHORT          quality;
    USHORT          button;
}DOUBLE_POSITION_MATRIX_TIME_Q_BUTTON_RECORD,*PDOUBLE_POSITION_MATRIX_TIME_Q_BUTTON_RECORD
;
```

### Members

#### **x**

This is the x position value of a 3-axis coordinate position system. The position will be reported in inches unless the SYSTEM\_PARAMETER\_TYPE, METRIC has been set to true in which case the position will be reported in millimeters.

#### **y**

This is the y position value of a 3-axis coordinate position system. The position will be reported in inches unless the SYSTEM\_PARAMETER\_TYPE, METRIC has been set to true in which case the position will be reported in millimeters.

#### **z**

This is the z position value of a 3-axis coordinate position system. The position will be reported in inches unless the SYSTEM\_PARAMETER\_TYPE, METRIC has been set to true in which case the position will be reported in millimeters.

#### **s[3][3]**

This is a 3x3 array of values. Each value is in the range +0.99997 to -1.00000

#### **time**

The time variable is the time stamp for the data record and is returned as a **double** value. The integer portion of the variable represents the number of elapsed seconds since midnight, Jan 1, 1970, UTC, and is the standard way of representing time and date. If this is cast as a time\_t structure, it can be converted into a date and time string using ctime() for example. The fractional part of the time variable represents fractions of a second.

#### **quality**

The quality value is a 16-bit unsigned integer. A very small quality number indicates no or minimal position and orientation errors due to distortion of the transmitter field depending on how sensitive you have set the error indicator. A large quality number indicates maximum error for the sensitivity level selected.

#### **button**

The button value is a 16-bit unsigned integer that represents the state of an external contact closure that the user has been connected to the tracker (see [SWITCH](#) for connector description). When a switch is connected, a 1 in the button value indicates the contact or switch is CLOSED, and a 0 indicates the contact is OPEN. The button line



is sampled and available in this data record once per transmitter axis cycle - 3 times the system measurement rate for a mid or short-range transmitter.

**Requirements**

**Windows NT/2000:** Requires Windows 2000 or later.

**Header:** Declared in ATC3DG.h

**Library:** Use ATC3DG.lib

**See Also**

DOUBLE\_POSITION\_RECORD, DOUBLE\_MATRIX\_RECORD

## DOUBLE\_POSITION\_QUATERNION\_TIME\_Q\_BUTTON\_RECORD

The **DOUBLE\_POSITION\_QUATERNION\_TIME\_Q\_BUTTON\_RECORD** structure contains position, quaternion, timestamp, distortion and button information in double floating point format.

```
typedef struct tagDOUBLE_POSITION_QUATERNION_TIME_Q_BUTTON{
    double          x;
    double          y;
    double          z;
    double          q[4];
    double          time;
    USHORT          quality;
    USHORT          button;
}DOUBLE_POSITION_QUATERNION_TIME_Q_BUTTON_RECORD,*PDOUBLE_POSITION_QUATERNION_TIME_Q_BUTTON_RECORD;
```

### Members

#### x

This is the x position value of a 3-axis coordinate position system. The position will be reported in inches unless the SYSTEM\_PARAMETER\_TYPE, METRIC has been set to true in which case the position will be reported in millimeters.

#### y

This is the y position value of a 3-axis coordinate position system. The position will be reported in inches unless the SYSTEM\_PARAMETER\_TYPE, METRIC has been set to true in which case the position will be reported in millimeters.

#### z

This is the z position value of a 3-axis coordinate position system. The position will be reported in inches unless the SYSTEM\_PARAMETER\_TYPE, METRIC has been set to true in which case the position will be reported in millimeters.

#### q[4]

This is an array of 4 quaternion values. Each value is in the range +0.99997 to -1.00000

#### time

The time variable is the time stamp for the data record and is returned as a **double** value. The integer portion of the variable represents the number of elapsed seconds since midnight, Jan 1, 1970, UTC, and is the standard way of representing time and date. If this is cast as a time\_t structure, it can be converted into a date and time string using ctime() for example. The fractional part of the time variable represents fractions of a second.

#### quality

The quality value is a 16-bit unsigned integer. A very small quality number indicates no or minimal position and orientation errors due to distortion of the transmitter field depending on how sensitive you have set the error indicator. A large quality number indicates maximum error for the sensitivity level selected.

#### button

The button value is a 16-bit unsigned integer that represents the state of an external contact closure that the user has been connected to the tracker (see [SWITCH](#) for connector description). When a switch is connected, a 1 in

the button value indicates the contact or switch is CLOSED, and a 0 indicates the contact is OPEN. The button line is sampled and available in this data record once per transmitter axis cycle - 3 times the system measurement rate for a mid or short-range transmitter.

**Requirements**

**Windows NT/2000:** Requires Windows 2000 or later.

**Header:** Declared in ATC3DG.h

**Library:** Use ATC3DG.lib

**See Also**

DOUBLE\_POSITION\_RECORD, DOUBLE\_QUATERNIONS\_RECORD

## 3D Guidance API Enumeration Types

The following enumeration types are used with the tracker:

[BIRD\\_ERROR\\_CODES](#)

[MESSAGE\\_TYPE](#)

[TRANSMITTER\\_PARAMETER\\_TYPE](#)

[SENSOR\\_PARAMETER\\_TYPE](#)

[BOARD\\_PARAMETER\\_TYPE](#)

[SYSTEM\\_PARAMETER\\_TYPE](#)

[HEMISPHERE\\_TYPE](#)

[AGC\\_MODE\\_TYPE](#)

[DATA\\_FORMAT\\_TYPE](#)

[BOARD\\_TYPES](#)

[DEVICE\\_TYPES](#)

## BIRD\_ERROR\_CODES

The **BIRD\_ERROR\_CODES** enumeration type defines the values that the enumerated error code field of the **ERRORCODE** can be returned with from a function call.

```
enum BIRD_ERROR_CODES{
    BIRD_ERROR_SUCCESS=0,
    BIRD_ERROR_PCB_HARDWARE_FAILURE,
    BIRD_ERROR_TRANSMITTER_EEPROM_FAILURE,
    BIRD_ERROR_SENSOR_SATURATION_START,
    BIRD_ERROR_ATTACHED_DEVICE_FAILURE,
    BIRD_ERROR_CONFIGURATION_DATA_FAILURE,
    BIRD_ERROR_ILLEGAL_COMMAND_PARAMETER,
    BIRD_ERROR_PARAMETER_OUT_OF_RANGE,
    BIRD_ERROR_NO_RESPONSE,
    BIRD_ERROR_COMMAND_TIME_OUT,
    BIRD_ERROR_INCORRECT_PARAMETER_SIZE,
    BIRD_ERROR_INVALID_VENDOR_ID,
    BIRD_ERROR_OPENING_DRIVER,
    BIRD_ERROR_INCORRECT_DRIVER_VERSION,
    BIRD_ERROR_NO_DEVICES_FOUND,
    BIRD_ERROR_ACCESSING_PCI_CONFIG,
    BIRD_ERROR_INVALID_DEVICE_ID,
    BIRD_ERROR_FAILED_LOCKING_DEVICE,
    BIRD_ERROR_BOARD_MISSING_ITEMS,
    BIRD_ERROR_NOTHING_ATTACHED,
    BIRD_ERROR_SYSTEM_PROBLEM,
    BIRD_ERROR_INVALID_SERIAL_NUMBER,
    BIRD_ERROR_DUPLICATE_SERIAL_NUMBER,
    BIRD_ERROR_FORMAT_NOT_SELECTED,
    BIRD_ERROR_COMMAND_NOT_IMPLEMENTED,
    BIRD_ERROR_INCORRECT_BOARD_DEFAULT,
    BIRD_ERROR_INCORRECT_RESPONSE,
    BIRD_ERROR_NO_TRANSMITTER_RUNNING,
    BIRD_ERROR_INCORRECT_RECORD_SIZE,
    BIRD_ERROR_TRANSMITTER_OVERCURRENT,
    BIRD_ERROR_TRANSMITTER_OPEN_CIRCUIT,
    BIRD_ERROR_SENSOR_EEPROM_FAILURE,
    BIRD_ERROR_SENSOR_DISCONNECTED,
    BIRD_ERROR_SENSOR_REATTACHED,
    BIRD_ERROR_NEW_SENSOR_ATTACHED,
    BIRD_ERROR_UNDOCUMENTED,
    BIRD_ERROR_TRANSMITTER_REATTACHED,
    BIRD_ERROR_WATCHDOG,
    BIRD_ERROR_CPU_TIMEOUT_START,
    BIRD_ERROR_PCB_RAM_FAILURE,
    BIRD_ERROR_INTERFACE,
    BIRD_ERROR_PCB_EPROM_FAILURE,
    BIRD_ERROR_SYSTEM_STACK_OVERFLOW,
    BIRD_ERROR_QUEUE_OVERRUN,
    BIRD_ERROR_PCB_EEPROM_FAILURE,
    BIRD_ERROR_SENSOR_SATURATION_END,
    BIRD_ERROR_NEW_TRANSMITTER_ATTACHED,
    BIRD_ERROR_SYSTEM_UNINITIALIZED,
    BIRD_ERROR_12V_SUPPLY_FAILURE,
    BIRD_ERROR_CPU_TIMEOUT_END,
    BIRD_ERROR_INCORRECT_PLD,
```

```

BIRD_ERROR_NO_TRANSMITTER_ATTACHED,
BIRD_ERROR_NO_SENSOR_ATTACHED,
BIRD_ERROR_SENSOR_BAD,
BIRD_ERROR_SENSOR_SATURATED,
BIRD_ERROR_CPU_TIMEOUT,
BIRD_ERROR_UNABLE_TO_CREATE_FILE,
BIRD_ERROR_UNABLE_TO_OPEN_FILE,
BIRD_ERROR_MISSING_CONFIGURATION_ITEM,
BIRD_ERROR_MISMATCHED_DATA,
BIRD_ERROR_CONFIG_INTERNAL,
BIRD_ERROR_UNRECOGNIZED_MODEL_STRING,
BIRD_ERROR_INCORRECT_SENSOR,
BIRD_ERROR_INCORRECT_TRANSMITTER,
BIRD_ERROR_ALGORITHM_INITIALIZATION
BIRD_ERROR_LOST_CONNECTION
BIRD_ERROR_INVALID_CONFIGURATION
BIRD_ERROR_TRANSMITTER_RUNNING
};

```

Enumerator Value	Meaning
BIRD_ERROR_SUCCESS=0	No errors occurred. Call completed successfully
BIRD_ERROR_PCB_HARDWARE_FAILURE	The Tracker firmware initialization did not complete within 10 seconds. It is assumed the board is faulty or the firmware has hung up somewhere. If the error is repeatable there is an unrecoverable hardware failure.
BIRD_ERROR_TRANSMITTER_EEPROM_FAILURE	<handled internally>
BIRD_ERROR_SENSOR_SATURATION_START	<handled internally>
BIRD_ERROR_ATTACHED_DEVICE_FAILURE	<obsolete>
BIRD_ERROR_CONFIGURATION_DATA_FAILURE	<obsolete>
BIRD_ERROR_ILLEGAL_COMMAND_PARAMETER	Invalid constant of the selected enumerated type has been used.
BIRD_ERROR_PARAMETER_OUT_OF_RANGE	The parameter value passed to the function call was not within the legal range for the parameter selected.
BIRD_ERROR_NO_RESPONSE	<obsolete>
BIRD_ERROR_COMMAND_TIME_OUT	Tracker on-board controller has failed to respond to a command issued to it. If error is repeatable there is an unrecoverable hardware failure.
BIRD_ERROR_INCORRECT_PARAMETER_SIZE	The value of the parameter size passed did not match the expected size of the parameter either being passed or returned with this call.
BIRD_ERROR_INVALID_VENDOR_ID	<obsolete>
BIRD_ERROR_OPENING_DRIVER	Non-specific error opening driver. Make sure that the driver is properly installed.
BIRD_ERROR_INCORRECT_DRIVER_VERSION	The wrong version of the driver has been installed for this version of the API dll. Install or re-install the correct driver.

BIRD_ERROR_NO_DEVICES_FOUND	No Tracker hardware was found by the host system. Verify that hardware is installed and is of the correct type.
BIRD_ERROR_ACCESSING_PCI_CONFIG	The error occurred in the pciBird PCI interface. There is a problem with the PCI configuration registers. If error is repeatable there is an unrecoverable hardware failure.
BIRD_ERROR_INVALID_DEVICE_ID	The device ID passed was out of range for the system.
BIRD_ERROR_FAILED_LOCKING_DEVICE	Driver could not lock PCI/miroBIRD resources. Check that there is not another application using the hardware.
BIRD_ERROR_BOARD_MISSING_ITEMS	The required resources were not found defined in the PCI configuration registers. Possible corrupt configuration. If error is repeatable there is an unrecoverable hardware failure.
BIRD_ERROR_NOTHING_ATTACHED	<obsolete>
BIRD_ERROR_SYSTEM_PROBLEM	<obsolete>
BIRD_ERROR_INVALID_SERIAL_NUMBER	<obsolete>
BIRD_ERROR_DUPLICATE_SERIAL_NUMBER	<obsolete>
BIRD_ERROR_FORMAT_NOT_SELECTED	<obsolete>
BIRD_ERROR_COMMAND_NOT_IMPLEMENTED	This function has not been implemented yet.
BIRD_ERROR_INCORRECT_BOARD_DEFAULT	An unexpected response was received from the controller on the Tracker hardware. The board is responding to commands but the data returned is corrupt. If the error is repeatable there is an unrecoverable hardware failure.
BIRD_ERROR_INCORRECT_RESPONSE	<obsolete>
BIRD_ERROR_NO_TRANSMITTER_RUNNING	A request was made to turn off the current transmitter by passing the value -1 with the parameter SELECT_TRANSMITTER selected and there was no transmitter currently running.
BIRD_ERROR_INCORRECT_RECORD_SIZE	The record size of the buffer passed to the function does not match the size of the data format currently selected.
BIRD_ERROR_TRANSMITTER_OVERCURRENT	<handled internally>
BIRD_ERROR_TRANSMITTER_OPEN_CIRCUIT	<handled internally>
BIRD_ERROR_SENSOR_EEPROM_FAILURE	<handled internally>
BIRD_ERROR_SENSOR_DISCONNECTED	<handled internally>
BIRD_ERROR_SENSOR_REATTACHED	<handled internally>
BIRD_ERROR_NEW_SENSOR_ATTACHED	<obsolete>
BIRD_ERROR_UNDOCUMENTED	<handled internally>
BIRD_ERROR_TRANSMITTER_REATTACHED	<handled internally>
BIRD_ERROR_WATCHDOG	Tracker internal watchdog timer has elapsed. If this error is repeatable there is an unrecoverable hardware failure.

BIRD_ERROR_CPU_TIMEOUT_START	<handled internally>
BIRD_ERROR_PCB_RAM_FAILURE	<handled internally>
BIRD_ERROR_INTERFACE	<handled internally>
BIRD_ERROR_PCB_EPROM_FAILURE	<handled internally>
BIRD_ERROR_SYSTEM_STACK_OVERFLOW	<handled internally>
BIRD_ERROR_QUEUE_OVERRUN	<handled internally>
BIRD_ERROR_PCB_EEPROM_FAILURE	<handled internally>
BIRD_ERROR_SENSOR_SATURATION_END	<handled internally>
BIRD_ERROR_NEW_TRANSMITTER_ATTACHED	<obsolete>
BIRD_ERROR_SYSTEM_UNINITIALIZED	The Tracker hardware and system has not been initialized yet. The <a href="#">InitializeBIRDSysyem</a> function must be called first.
BIRD_ERROR_12V_SUPPLY_FAILURE	<handled internally>
BIRD_ERROR_CPU_TIMEOUT_END	<handled internally>
BIRD_ERROR_INCORRECT_PLD	The PLD version on the Tracker hardware is incompatible with this version of the API dll. Verify Tracker model installed.
BIRD_ERROR_NO_TRANSMITTER_ATTACHED	A request was made to do one of the following: <ol style="list-style-type: none"> <li>1) Turn off the currently running transmitter and there is no transmitter attached to the system</li> <li>2) Turn on the transmitter with the selected ID and there is no transmitter attached at that ID.</li> </ol>
BIRD_ERROR_NO_SENSOR_ATTACHED	Request for data record from a sensor channel where no sensor is attached or the sensor has been removed.
BIRD_ERROR_SENSOR_BAD	The attached sensor is not saturated but is exhibiting another unspecified problem, which prevents it from operating normally. Use the GetSensorStatus function to determine the precise problem.
BIRD_ERROR_SENSOR_SATURATED	The attached sensor, which is otherwise OK, is currently saturated. This may occur if the sensor is too close to the transmitter or if the sensor is too close to metal or an external magnetic field.
BIRD_ERROR_CPU_TIMEOUT	Tracker on-board controller had insufficient time to execute the position and orientation algorithm. This frequently occurs because the Tracker controller is being overwhelmed with user interface commands. Reduce the rate at which GetAsynchronousRecord is being called.
BIRD_ERROR_UNABLE_TO_CREATE_FILE	The call was unable to complete for some unspecified reason. Check the format of the file name string.
BIRD_ERROR_UNABLE_TO_OPEN_FILE	The call was unable to complete for some unspecified reason. Check the format of the file name string.
BIRD_ERROR_MISSING_CONFIGURATION_ITEM	A mandatory configuration item was missing from the initialization file. Review contents of initialization file or use SaveSystemConfiguration() to automatically save a



	correctly formatted initialization file.
BIRD_ERROR_MISMATCHED_DATA	Data item in the initialization file does not match a system parameter. For example the initialization file states the system has 3 boards (NumberOfBoards=3) but the system initialization routine – <a href="#">InitializeBIRDSystem</a> only detected two.
BIRD_ERROR_CONFIG_INTERNAL	Internal error in configuration file handler. Report to vendor.
BIRD_ERROR_UNRECOGNIZED_MODEL_STRING	The firmware is reporting a model string, which is unrecognized by the API dll. This could be due to a hardware failure causing the model string data to be corrupted, a corrupted board EEPROM, or the board installed is of a type not recognized by the API dll. If the error is repeatable return to vendor.
BIRD_ERROR_INCORRECT_SENSOR	An invalid sensor type has been attached to this card.
BIRD_ERROR_INCORRECT_TRANSMITTER	An invalid transmitter type has been attached to this card.
BIRD_ERROR_ALGORITHM_INITIALIZATION	Initialization of the nondipole transmitter algorithm failed.
BIRD_ERROR_LOST_CONNECTION	Connection with tracker lost
BIRD_ERROR_INVALID_CONFIGURATION	Invalid multi-unit configuration
BIRD_ERROR_TRANSMITTER_RUNNING	Operation illegal with transmitter running.
BIRD_ERROR_MAXIMUM_VALUE	Final error code place holder

## Requirements

**Windows NT/2000:** Requires Windows 2000 or later.

**Header:** Declared in ATC3DGm.h

**Library:** Use ATC3DGm.lib

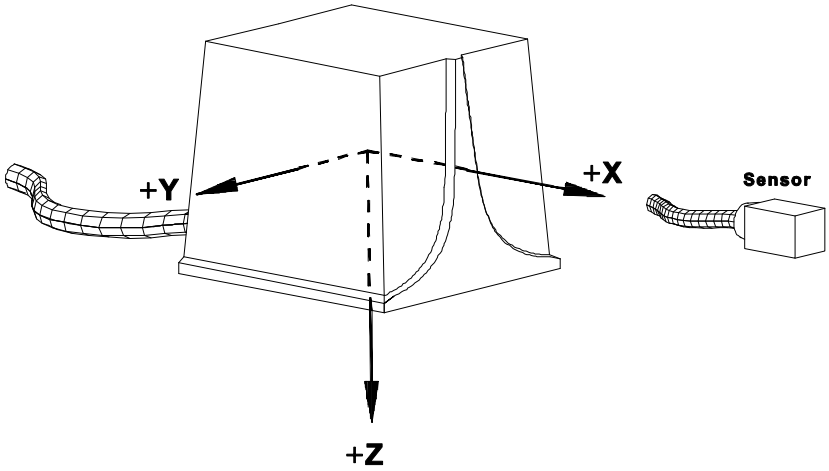
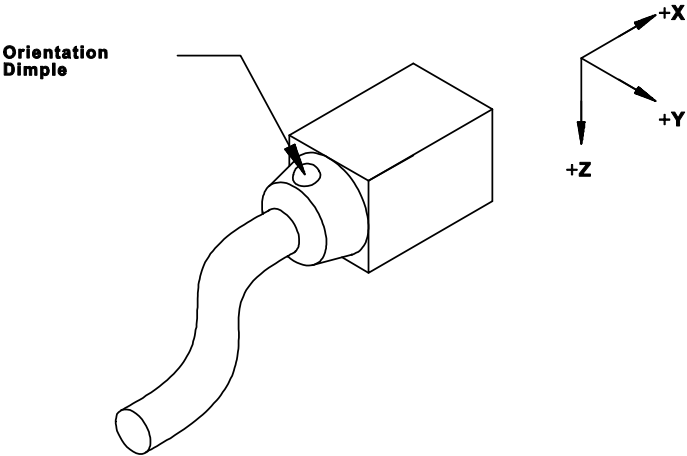
## See Also

## SENSOR\_PARAMETER\_TYPE

The **SENSOR\_PARAMETER\_TYPE** enumeration type defines values that are used with the `GetSensorParameter` and `SetSensorParameter` functions to specify the operational characteristics of an individual sensor.

```
enum SENSOR_PARAMETER_TYPE{
    DATA_FORMAT,
    ANGLE_ALIGN,
    HEMISPHERE,
    FILTER_AC_WIDE_NOTCH,
    FILTER_AC_NARROW_NOTCH,
    FILTER_DC_ADAPTIVE,
    FILTER_ALPHA_PARAMETERS,
    FILTER_LARGE_CHANGE,
    QUALITY,
    SERIAL_NUMBER_RX,
    SENSOR_OFFSET,
    VITAL_PRODUCT_DATA_RX,
    VITAL_PRODUCT_DATA_PREAMP
};
```

Enumerator Value	Meaning
DATA_FORMAT	See the Data Format Structures section for details
ANGLE_ALIGN	<p>By default, the angle outputs from the Tracker are measured in the coordinate frame defined by the transmitter's X, Y and Z axes, as shown <a href="#">Default Reference Frames</a>, and are measured with respect to rotations about the physical X, Y and Z axes of the sensor, also shown. The ANGLE_ALIGN parameter allows you to mathematically change the sensor's X, Y and Z axes to an orientation, which differs from that of the actual sensor.</p> <p>For example: Suppose that during installation you find it necessary, due to physical requirements, to rotate the sensor, resulting in its angle outputs reading Azim = 5 deg, Elev = 10 and Roll = 15 when it is in its normal "resting" position. To compensate, use the ANGLE_ALIGN parameter, passing as values 5, 10 and 15 degrees. After this sequence is sent, the sensor outputs will be zero, and orientations will be computed as if the sensor were not misaligned.</p> <p>ANGLE_ALIGN parameters are not meaningful for 5DOF sensors. ANGLE_ALIGN parameters applied to a 5DOF sensor will be ignored.</p>

<p>ANGLE_ALIGN, continued</p>	<p style="text-align: center;"><b>Transmitter</b></p>  <p style="text-align: center;">Measurement Reference Frame (Standard Transmitter)</p> <p><b>Orientation Dimple</b></p>  <p style="text-align: center;">Receiver Zero Orientation (8mm Sensor)</p>
<p>HEMISPHERE</p>	<p>The shape of the magnetic field transmitted by the Tracker is symmetrical about each of the axes of the transmitter. This symmetry leads to an ambiguity in determining the sensor's X, Y, Z position. The amplitudes will always be correct, but the signs (<math>\pm</math>) may all be wrong, depending upon the hemisphere of operation. In many applications, this will not be relevant, but if you desire an unambiguous measure of position, operation must be either confined to a defined hemisphere or your host computer must 'track' the location of the sensor.</p> <p>There is no ambiguity in the sensor's orientation angles as output in the ANGLES data formats, or in the rotation matrix as output in the MATRIX formats.</p> <p>The HEMISPHERE parameter is used to tell the Bird in which hemisphere, centered about</p>

	<p>the transmitter, the sensor will be operating. There are six hemispheres from which you may choose: the FRONT (forward), BACK (rear), TOP (upper), BOTTOM (lower), LEFT, and the RIGHT. If no HEMISPHERE parameter is specified, the FRONT is used by default.</p> <p>The ambiguity in position determination can be eliminated if your host computer's software continuously 'tracks' the sensor location. In order to implement tracking, you must understand the behavior of the signs (<math>\pm</math>) of the X, Y, and Z position outputs when the sensor crosses a hemisphere boundary. When you select a given hemisphere of operation, the sign on the position axes that defines the hemisphere direction is forced to positive, even when the sensor moves into another hemisphere. For example, the power-up default hemisphere is the front hemisphere. This forces X position outputs to always be positive. The signs on Y and Z will vary between plus and minus depending on where you are within this hemisphere. If you had selected the bottom hemisphere, the sign of Z would always be positive and the signs on X and Y will vary between plus and minus. If you had selected the left hemisphere, the sign of Y would always be negative, etc.</p> <p>Regarding the default front hemisphere, if the sensor moved into the back hemisphere, the signs on Y and Z would instantaneously change to opposite polarities while the sign on X remained positive. To 'track' the sensor, your host software, on detecting this sign change, would reverse the signs on The Tracker's X, Y, and Z outputs. In order to 'track' correctly: You must start 'tracking' in the selected hemisphere so that the signs on the outputs are initially correct, and you must guard against the case where the sensor legally crossed the Y = 0, Z = 0 axes simultaneously without having crossed the X = 0 axes into the other hemisphere.</p>
FILTER_AC_WIDE_NOTCH	The AC WIDE notch filter refers to a six tap FIR notch filter that is applied to the sensor data to eliminate sinusoidal signals with a frequency between 30 and 72 hertz. If your application requires minimum transport delay between measurement of the sensor's position/orientation and the output of these measurements, you may want to evaluate the effect on your application with this filter shut off and the AC NARROW notch filter on.
FILTER_AC_NARROW_NOTCH	The AC NARROW notch filter refers to a two tap finite impulse response (FIR) notch filter that is applied to signals measured by the Tracker's sensor to eliminate a narrow band of noise with sinusoidal characteristics. Use this filter in place of the AC WIDE notch filter when you want to minimize the transport delay between Tracker measurement of the sensor's position/orientation and the output of these measurements. The transport delay of the AC NARROW notch filter is approximately one third the delay of the AC WIDE notch filter.
FILTER_DC_ADAPTIVE	<p>The DC filter refers to an adaptive, infinite impulse response (IIR) low pass filter applied to the sensor data to eliminate high frequency noise. Generally, this filter is always required in the system unless your application can work with noisy outputs. When the DC filter is enabled, you can modify its noise/lag characteristics by changing alphaMin and Vm.</p> <p>To use the default filter settings, just set the FILTER_DC_ADAPTIVE value to 1.0. To disable the filter set the value to 0.0</p>
FILTER_ALPHA_PARAMETERS	<p>To modify the filter characteristics, configure the elements of the structure <a href="#">ADAPTIVE_PARAMETERS</a>.</p> <p>The alphaMin and alphaMax values define the lower and upper ends of the adaptive range that the filter constant alpha can assume in the DC filter, as a function of sensor to transmitter separation. When alphaMin = 0 Hex, the DC filter will provide an infinite amount of filtering (the outputs will never change even if you move the sensor). When alphaMin = 0.99996 = 7FFF Hex, the DC filter will provide no</p>

	<p>filtering of the data. At the shorter ranges you may want to increase alphaMin to obtain less lag while at longer ranges you may want to decrease alphaMin to provide more filtering (less noise/more lag). Note that alphaMin must always be less than alphaMax.</p> <p>When there is a fast motion of the sensor, the adaptive filter reduces the amount of filtering by increasing the ALPHA used in the filter. It will increase ALPHA only up to the limiting alphaMax value. By doing this, the lag in the filter is reduced during fast movements. When alphaMax = 0.99996 = 7FFF Hex, the DC filter will provide no filtering of the data during fast movements. Some users may want to decrease alphaMax to increase the amount of filtering if the Tracker's outputs are too noisy during rapid sensor movement.</p> <p>The 7 words that make up the Vm table values represent the expected noise that the DC filter will measure. By changing the table values, you can increase or decrease the DC filter's lag as a function of sensor range from the transmitter.</p> <p>The DC filter is adaptive in that it tries to reduce the amount of low pass filtering in the Tracker as it detects translation or rotation rates in the Tracker's sensor. Reducing the amount of filtering results in less filter lag.</p> <p>Unfortunately, electrical noise in the environment—when measured by the sensor—also makes it look like the sensor is undergoing a translation and rotation. As the sensor moves farther and farther away from the transmitter, the amount of noise measured by the sensor appears to increase because the measured transmitted signal level is decreasing and the sensor amplifier gain is increasing. In order to decide if the amount of filtering should be reduced, the Tracker has to know if the measured rate is a real sensor rate due to movement or a false rate due to noise. The Tracker gets this knowledge by the user specifying what the expected noise levels are in the operating environment as a function of distance from the transmitter. These noise levels are the 7 words that form the Vm table. The Vm values can range from 1 for almost no noise to 32767 for a lot of noise.</p>
FILTER_LARGE_CHANGE	<p>When the LARGE_CHANGE filter is selected, the position and orientation outputs are not allowed to change if the system detects a sudden large change in the outputs. Large undesirable changes may occur at large separation distances between the transmitter and sensor when the sensor undergoes a fast rotation or translation. If the LARGE_CHANGE value is TRUE the outputs will not be updated if a large change is detected. If value is FALSE, the outputs will change.</p>
QUALITY	<p>This data structure is used to adjust the output characteristics of the Quality number. Also referred to as the METAL error or Distortion number, this value is returned with certain data formats and gives the user an indication of the degree to which the position and angle measurements are in error. This error may be due to 'bad' metals located near the transmitter and sensor, or due to TRACKER 'system' errors. 'Bad' metals are metals with high electrical conductivity such as aluminum, or high magnetic permeability such as steel. 'Good' metals have low conductivity and low permeability such as 300 series stainless steel, or titanium. The QUALITY error number also reflects TRACKER 'system' errors resulting from accuracy degradations in the transmitter, sensor, or other electronic components. It will represent a level of accuracy degradation resulting from either movement of the sensor or environmental noise. A very small QUALITY error number indicates no or minimal position and angle errors depending on how sensitive you have set the error indicator. A large QUALITY error number indicates maximum error for the sensitivity level selected.</p> <p>Users of the QUALITY error number will find that as a metal detector, it is sensitive to the introduction of metals in an environment where no metals were initially present. This metal detector can fool you, however, if there are some metals</p>

	<p>initially present and you introduce new metals. It is possible for the new metal to cause a distortion in the magnetic field that reduces the existing distortion at the sensor. When this occurs you'll see the Error value initially decreases, indicating less error, and then finally start increasing again as the new metal causes more distortion. <b>Users beware. You need to evaluate your application for suitability of this metal detector.</b></p> <p>Because the TRACKER is used in many different applications and environments, the QUALITY error indicator needs to be sensitive to this broad range of environments. Some users may want the error indicator to be sensitive to very small amounts of metal in the environment while other applications may only want the error indicator sensitive to large amounts of metal. To accommodate this range of detection sensitivity, the <a href="#">SetSensorParameter()</a> allows the user to set a QUALITY Sensitivity setting that is appropriate to their application.</p> <p>The QUALITY error number will always show there is some error in the system even when there are no metals present. This error indication usually increases as the distance between the transmitter and sensor increases, and is due to the fact that TRACKER components cannot be made or calibrated perfectly. To minimize the amount of this inherent error in the Error value, a linear curve fit, defined by a <b>slope</b> and <b>offset</b>, is made to this inherent error and stored in each individual sensor's memory since the error depends primarily on the size of the sensor being used (25mm, 8mm, or 5 mm). The <a href="#">Quality Parameters</a> Structure (manipulated through the <a href="#">SetSensorParameter()</a> command) allows the user to eliminate or change these values. For example, maybe the user's standard environment has large errors and he or she wants to look at variations from this standard environment. To do this he or she would adjust the Slope and Offset settings to minimize the QUALITY error values.</p> <p>The QUALITY error number that is output is computed from the following equation:</p> $\text{QUALITY error} = \text{Sensitivity} \times (\text{ErrorSYSTEM} - (\text{Offset} + \text{Slope} \times \text{Range}))$ <p>Where Range is the distance between the transmitter and sensor.</p> <p><b>Sensitivity</b></p> <p>The user supplies a Sensitivity value based on how little or how much he or she wants the QUALITYerror value to reflect errors.</p> <p><b>Offset</b></p> <p>If you are trying to minimize the base errors in the system by adjusting the Offset you could set the Sensitivity =1, and the Slope=0 and read the Offset directly as the QUALITYerror number.</p> <p><b>Slope</b></p> <p>You can determine the slope by setting the Sensitivity=1 and looking at the change in QUALITYerror as you translate the sensor from range=0 to range max for the system (i.e. 36" ). Since its difficult to go from range =0 to max., you might just translate over say half the distance and double the error value change you measure.</p> <p><b>Alpha</b></p> <p>The QUALITYerror value is filtered before output to the user to minimize noise jitter. The Alpha value determines how much filtering is applied to the error value. Range is FFFF -&gt; 0 Users should think of it as a signed fractional value with a range of 0.9999 -&gt; 0 (negative numbers not allowed). A zero value is an infinite amount of</p>
--	---

	filtering, whereas a 0.9999 value is no filtering. As Alpha gets smaller the time lag between the insertion of metal in the environment and it being reported in the QUALITYerror value increases.
SERIAL_NUMBER_RX	Returns the serial number of the attached sensor.
SENSOR_OFFSETS	<p>Normally the position outputs from the 3DGuidance™ represent the x, y, z position of the center of the sensor with respect to the origin of the Transmitter. The SENSOR_OFFSETS command allows the user to specify a location that is offset from the center of the sensor.</p> <p>The x, y, z offset distances you supply in a DOUBLE_POSITION_RECORD with this command are measured in the sensor reference frame and are measured from the sensor center to the desired position on the tracked object.</p>
VITAL_PRODUCT_DATA_RX	<p>Used to read or write to individual bytes in the Vital Product Data (VPD) storage area on the sensor. The VPD section comprises 128 bytes of user modifiable data storage. It is the user's responsibility to define the contents and structure and to maintain that structure. The parameter passed with these commands is a structure <a href="#">VPD_COMMAND_PARAMETERS</a> which contains the address of the target byte and, in the case of the write command (<a href="#">SetSensorParameter</a>) the value of the byte to be written. In the case of the read command (<a href="#">GetSensorParameter</a>), the value of the byte read from the VPD is placed in the value location in the structure.</p> <p>Note: Reading or writing VPD is not allowed when the transmitter is running.</p>
VITAL_PRODUCT_DATA_PREAMP	<p>Used to read or write to individual bytes in the Vital Product Data (VPD) storage area on the preamp. The VPD section comprises 128 bytes of user modifiable data storage. It is the user's responsibility to define the contents and structure and to maintain that structure. The parameter passed with these commands is a structure <a href="#">VPD_COMMAND_PARAMETERS</a> which contains the address of the target byte and, in the case of the write command (<a href="#">SetSensorParameter</a>), the value of the byte to be written. In the case of the read command (<a href="#">GetSensorParameter</a>), the value of the byte read from the VPD is placed in the value location in the structure.</p> <p>Note: Reading or writing VPD is not allowed when the transmitter is running.</p>

## Requirements

**Windows NT/2000:** Requires Windows 2000 or later.

**Header:** Declared in ATC3DGm.h

**Library:** Use ATC3DGm.lib

## See Also

## MESSAGE\_TYPE

The **MESSAGE\_TYPE** enumeration type defines a value used with the `GetErrorText` function to define the type of message string returned from the call.

```
enum MESSAGE_TYPE{  
    SIMPLE_MESSAGE,  
    VERBOSE_MESSAGE  
};
```

Enumerator Value	Meaning
SIMPLE_MESSAGE	The call <code>GetErrorText</code> will return a short terse message string describing the meaning of the error code passed.
VERBOSE_MESSAGE	The call <code>GetErrorText</code> will return a message string containing a full and comprehensive description of the problem and possible resolutions where appropriate.

### Requirements

**Windows NT/2000:** Requires Windows 2000 or later.

**Header:** Declared in `ATC3DGm.h`

**Library:** Use `ATC3DGm.lib`

### See Also

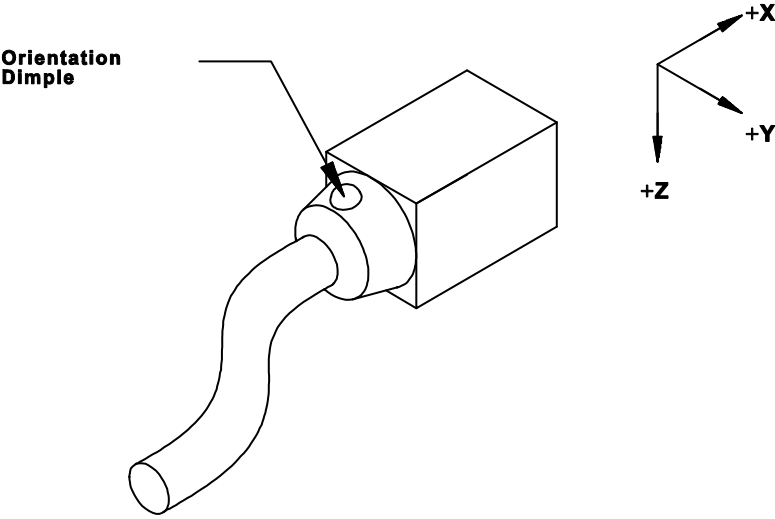


## TRANSMITTER\_PARAMETER\_TYPE

The **TRANSMITTER\_PARAMETER\_TYPE** enumeration type defines values that are used with the GetTransmitterParameter and SetTransmitterParameter functions to specify the operational characteristics of an individual transmitter.

```
enum TRANSMITTER_PARAMETER_TYPE{
    SERIAL_NUMBER_TX,
    REFERENCE_FRAME,
    XYZ_REFERENCE_FRAME,
    VITAL_PRODUCT_DATA_TX
};
```

Enumerator Value	Meaning
SERIAL_NUMBER_TX	This returns the serial number of the attached physical device
REFERENCE_FRAME	<p>By default, the Tracker's reference frame is defined by the transmitter's physical X, Y, and Z axes, as shown in <a href="#">Default Reference Frames</a>. In some applications, it may be desirable to have the orientation measured with respect to another reference frame. The REFERENCE FRAME parameter permits you to define a new reference frame by inputting the angles required to align the physical axes of the transmitter to the X, Y, and Z axes of the new reference frame. The alignment angles are defined as rotations about the Z, Y, and X axes of the transmitter. These angles are called the, Azimuth, Elevation, and Roll angles.</p> <p>Although a change to the REFERENCE FRAME parameter values will cause the Tracker's output angles to change, it has no effect on the position outputs. If you want The Tracker's XYZ position reference frame to also change with this parameter, then you must enable this mode using the XYZREFERENCE FRAME parameter.</p> <div data-bbox="565 1228 1380 1738" data-label="Image"> <p style="text-align: center;">Transmitter</p> <p style="text-align: center;">+Y      +X      Sensor</p> <p style="text-align: center;">+Z</p> <p style="text-align: center;">Measurement Reference Frame (Standard Transmitter)</p> </div>

	 <p>Orientation Dimple</p> <p>Receiver Zero Orientation (8mm Sensor)</p>
XYZ_REFERENCE_FRAME	<p>When the Boolean value XYZ_REFERENCE_FRAME is TRUE, the Tracker's XYZ measurement frame will also correspond to the new reference frame defined by the REFERENCE_FRAME parameter values. When the Boolean value is FALSE, the XYZ measurement frame reverts to the orientation of the transmitter's physical XYZ axes.</p>
VITAL_PRODUCT_DATA_TX	<p>Used to read or write to individual bytes in the Vital Product Data (VPD) storage area on the transmitter. The VPD section comprises 128 bytes of user modifiable data storage. It is the user's responsibility to define the contents and structure and to maintain that structure. The parameter passed with these commands is a structure <a href="#">VPD_COMMAND_PARAMETERS</a> which contains the address of the target byte and, in the case of the write command (<a href="#">SetTransmitterParameter</a>) the value of the byte to be written. In the case of the read command (<a href="#">GetTransmitterParameter</a>) the value of the byte read from the VPD is placed in the value location in the structure.</p> <p>Note: Reading or writing VPD is not allowed when the transmitter is running.</p>

## Requirements

**Windows NT/2000:** Requires Windows 2000 or later.

**Header:** Declared in ATC3DGm.h

**Library:** Use ATC3DGm.lib

## See Also

## BOARD\_PARAMETER\_TYPE

The **BOARD\_PARAMETER\_TYPE** enumeration type defines parameters that can be changed and/or inspected with the `GetBoardParameter` and `SetBoardParameter` functions. These parameters control the operational characteristics of the board. One of these enumerated values is passed as a parameter to the call to indicate the type and size of the actual parameter passed. The table below describes the actual type and size and purpose of the parameters passed for each of these types.

```
enum BOARD_PARAMETER_TYPE{
    SERIAL_NUMBER_PCB,
    BOARD_SOFTWARE_REVISIONS,
    POST_ERROR_PCB,

    DIAGNOSTIC_TEST_PCB,

    VITAL_PRODUCT_DATA_PCB
};
```

Enumerator Value	Meaning
SERIAL_NUMBER_PCB	Returns the serial number of the 3DGuidance™ board.
BOARD_SOFTWARE_REVISIONS	Returns the board software revisions in a <a href="#">BOARD_SOFTWARE_REVISIONS</a> structure.
POST_ERROR_PCB	Not supported.
DIAGNOSTIC_TEXT_PCB	Not supported.
VITAL_PRODUCT_DATA_PCB	Used to read or write to individual bytes in the Vital Product Data (VPD) storage area on the electronics unit. The VPD section comprises 128 bytes of user modifiable data storage. It is the user's responsibility to define the contents and structure and to maintain that structure. The parameter passed with these commands is a structure <a href="#">VPD_COMMAND_PARAMETERS</a> which contains the address of the target byte and, in the case of the write command ( <a href="#">SetBoardParameter</a> ) the value of the byte to be written. In the case of the read command ( <a href="#">GetBoardParameter</a> ), the value of the byte read from the VPD is placed in the value location in the structure.  Note: Reading or writing VPD is not allowed when the transmitter is running.

### Requirements

**Windows NT/2000:** Requires Windows 2000 or later.

**Header:** Declared in ATC3DGm.h

**Library:** Use ATC3DGm.lib

### See Also

## SYSTEM\_PARAMETER\_TYPE

The **SYSTEM\_PARAMETER\_TYPE** enumeration type defines parameters that can be changed and/or inspected with the `GetSystemParameter` and `SetSystemParameter` functions. These parameters control the operational characteristics of the system. One of these enumerated values is passed as a parameter to the call to indicate the type and size of the actual parameter passed. The table below describes the actual type and size and purpose of the parameters passed for each of these types.

```
enum SYSTEM_PARAMETER_TYPE{
    SELECT_TRANSMITTER,
    POWER_LINE_FREQUENCY,
    AGC_MODE,
    MEASUREMENT_RATE,
    MAXIMUM_RANGE,
    METRIC,
    VITAL_PRODUCT_DATA,
    POST_ERROR,
    DIAGNOSTIC_TEST,
    REPORT_RATE
};
```

Enumerator Value	Meaning
SELECT_TRANSMITTER	Either select and turn on a specific transmitter or turn off the current transmitter. The parameter passed is a short int, which contains the id of the transmitter selected to be turned on. If the current transmitter needs to be turned off this value should be set to -1.
POWER_LINE_FREQUENCY	Informs the hardware of the frequency of the AC power source. The parameter passed is a double value describing the frequency in Hz. There are only two valid values: either 50.0 or 60.0 Hz.
AGC_MODE	Select the automatic gain control (AGC) mode. The parameter passed is one of the enumerated type <a href="#">AGC_MODE_TYPE</a> .
MEASUREMENT_RATE	Set the measurement rate. The parameter passed is a double value and represents the measurement rate in Hz. The valid range of values is $20.0 < \text{rate} < 110.0$
MAXIMUM_RANGE	Sets the system maximum range. The parameter passed is a double value representing the maximum range in any of the 3 axes in inches. There is only one valid range and that is 36.0 inches.
METRIC	Enables/disables metric position reporting. The parameter passed is a bool. If the value is true then metric reporting is selected otherwise if the value is false then metric reporting is turned off. Metric data is reported in millimeters. Non-metric data is reported in inches.
VITAL_PRODUCT_DATA	Not supported.
POST_ERROR	Not supported.
DIAGNOSTIC_TEST	Not supported.
REPORT_RATE	Sets the number of updates before a new data record is returned when streaming data.
END_OF_LIST	Final system parameter type place holder

**Requirements**

**Windows NT/2000:** Requires Windows 2000 or later.

**Header:** Declared in ATC3DGm.h

**Library:** Use ATC3DGm.lib

**See Also**

## HEMISPHERE\_TYPE

The **HEMISPHERE\_TYPE** enumeration type defines values that are used when setting the HEMISPHERE with the SetSensorParameter call. The default HEMISPHERE\_TYPE is FRONT.

```
enum HEMISPHERE_TYPE{
    FRONT,
    BACK,
    TOP,
    BOTTOM,
    LEFT,
    RIGHT
};
```

Enumerator Value	Meaning
FRONT	The FRONT is the forward hemisphere in front of the transmitter. The front of the transmitter is the side with the Ascension logo molded into the case. It is the side opposite the side with the 2 positioning holes. This is the default.
BACK	The BACK is the opposite hemisphere to the FRONT hemisphere.
TOP	The TOP hemisphere is the upper hemisphere. When the transmitter is sitting on a flat surface with the locating holes on the surface the TOP hemisphere is above the transmitter.
BOTTOM	The BOTTOM hemisphere is the opposite hemisphere to the TOP hemisphere.
LEFT	The LEFT hemisphere is the hemisphere to the left of the observer when looking at the transmitter from the back.
RIGHT	The RIGHT hemisphere is the opposite hemisphere to the LEFT hemisphere. The LEFT hemisphere is on the left side of the observer when looking at the transmitter from the back.

### Requirements

**Windows NT/2000:** Requires Windows 2000 or later.

**Header:** Declared in ATC3DGm.h

**Library:** Use ATC3DGm.lib

### See Also

## AGC\_MODE\_TYPE

The **AGC\_MODE\_TYPE** enumeration type defines values that are used when setting the AGC\_MODE with the SetSensorParameter call. The default is TRANSMITTER\_AND\_SENSOR\_AGC.

```
enum AGC_MODE_TYPE{
    TRANSMITTER_AND_SENSOR_AGC,
    SENSOR_AGC_ONLY
};
```

Enumerator Value	Meaning
TRANSMITTER_AND_SENSOR_AGC	Select both transmitter power switching and sensor gain control for the AGC implementation. This is the default. NOTE: As the sensor moves away from the transmitter the signal decreases so it is necessary to increase the gain of the sensor amplifier. As the sensor approaches the transmitter the signal increases so the sensor amp gain needs to be reduced. But, there comes a point where the sensor is so close to the transmitter that the signal saturates the sensor and at that point it becomes necessary to reduce the power of the transmitter. Doing this allows the sensor to be used close to the transmitter. All transmitter power switching and sensor amp gain control is handled automatically for the user.
SENSOR_AGC_ONLY	Disable transmitter power switching and use only sensor gain control for the AGC implementation. NOTE: When the power switching is disabled the transmitter will run at full power all the time. This means that there comes a point at which the sensor as it is approaching the transmitter will saturate. Since the transmitter will not reduce power this is the minimum limiting range for this mode of operation.

### Requirements

**Windows NT/2000:** Requires Windows 2000 or later.

**Header:** Declared in ATC3DGm.h

**Library:** Use ATC3DGm.lib

### See Also

## DATA\_FORMAT\_TYPE

The **DATA\_FORMAT\_TYPE** enumeration type

```
enum DATA_FORMAT_TYPE{
    NO_FORMAT_SELECTED=0,
    SHORT_POSITION,
    SHORT_ANGLES,
    SHORT_MATRIX,
    SHORT_QUATERNIONS,
    SHORT_POSITION_ANGLES,
    SHORT_POSITION_MATRIX,
    SHORT_POSITION_QUATERNION,
    DOUBLE_POSITION,
    DOUBLE_ANGLES,
    DOUBLE_MATRIX,
    DOUBLE_QUATERNIONS,
    DOUBLE_POSITION_ANGLES,
    DOUBLE_POSITION_MATRIX,
    DOUBLE_POSITION_QUATERNION,
    DOUBLE_POSITION_TIME_STAMP,
    DOUBLE_ANGLES_TIME_STAMP,
    DOUBLE_MATRIX_TIME_STAMP,
    DOUBLE_QUATERNIONS_TIME_STAMP,
    DOUBLE_POSITION_ANGLES_TIME_STAMP,
    DOUBLE_POSITION_MATRIX_TIME_STAMP,
    DOUBLE_POSITION_QUATERNION_TIME_STAMP,
    DOUBLE_POSITION_TIME_Q,
    DOUBLE_ANGLES_TIME_Q,
    DOUBLE_MATRIX_TIME_Q,
    DOUBLE_QUATERNIONS_TIME_Q,
    DOUBLE_POSITION_ANGLES_TIME_Q,
    DOUBLE_POSITION_MATRIX_TIME_Q,
    DOUBLE_POSITION_QUATERNION_TIME_Q,
    SHORT_ALL,
    DOUBLE_ALL,
    DOUBLE_ALL_TIME_STAMP,
    DOUBLE_ALL_TIME_STAMP_Q,
    DOUBLE_ALL_TIME_STAMP_Q_RAW,
    DOUBLE_POSITION_ANGLES_TIME_Q_BUTTON,
    DOUBLE_POSITION_MATRIX_TIME_Q_BUTTON,
    DOUBLE_POSITION_QUATERNION_TIME_Q_BUTTON,
    DOUBLE_POSITION_ANGLES_MATRIX_QUATERNION_TIME_Q_BUTTON,

    MAXIMUM_FORMAT_CODE
};
```

Enumerator Value	Selects Data Record of Structure Type:
NO_FORMAT_SELECTED=0,	No data format selected.
SHORT_POSITION,	<a href="#">SHORT_POSITION_RECORD</a>
SHORT_ANGLES,	<a href="#">SHORT_ANGLES_RECORD</a>
SHORT_MATRIX,	<a href="#">SHORT_MATRIX_RECORD</a>



SHORT_QUATERNIONS,	<a href="#">SHORT_QUATERNIONS_RECORD</a>
SHORT_POSITION_ANGLES,	<a href="#">SHORT_POSITION_ANGLES_RECORD</a>
SHORT_POSITION_MATRIX,	<a href="#">SHORT_POSITION_MATRIX_RECORD</a>
SHORT_POSITION_QUATERNION,	<a href="#">SHORT_POSITION_QUATERNION_RECORD</a>
DOUBLE_POSITION,	<a href="#">DOUBLE_POSITION_RECORD</a>
DOUBLE_ANGLES,	<a href="#">DOUBLE_ANGLES_RECORD</a>
DOUBLE_MATRIX,	<a href="#">DOUBLE_MATRIX_RECORD</a>
DOUBLE_QUATERNIONS,	<a href="#">DOUBLE_QUATERNIONS_RECORD</a>
DOUBLE_POSITION_ANGLES,	<a href="#">DOUBLE_POSITION_ANGLES_RECORD</a>
DOUBLE_POSITION_MATRIX,	<a href="#">DOUBLE_POSITION_MATRIX_RECORD</a>
DOUBLE_POSITION_QUATERNION,	<a href="#">DOUBLE_POSITION_QUATERNION_RECORD</a>
DOUBLE_POSITION_TIME_STAMP,	<a href="#">DOUBLE_POSITION_TIME_STAMP_RECORD</a>
DOUBLE_ANGLES_TIME_STAMP,	<a href="#">DOUBLE_ANGLES_TIME_STAMP_RECORD</a>
DOUBLE_MATRIX_TIME_STAMP,	<a href="#">DOUBLE_MATRIX_TIME_STAMP_RECORD</a>
DOUBLE_QUATERNIONS_TIME_STAMP,	<a href="#">DOUBLE_QUATERNIONS_TIME_STAMP_RECORD</a>
DOUBLE_POSITION_ANGLES_TIME_STAMP,	<a href="#">DOUBLE_POSITION_ANGLES_TIME_STAMP_RECORD</a>
DOUBLE_POSITION_MATRIX_TIME_STAMP,	<a href="#">DOUBLE_POSITION_MATRIX_TIME_STAMP_RECORD</a>
DOUBLE_POSITION_QUATERNION_TIME_STAMP,	<a href="#">DOUBLE_POSITION_QUATERNION_TIME_STAMP_RECORD</a>
DOUBLE_POSITION_TIME_Q,	<a href="#">DOUBLE_POSITION_TIME_Q_RECORD</a>
DOUBLE_ANGLES_TIME_Q,	<a href="#">DOUBLE_ANGLES_TIME_Q_RECORD</a>
DOUBLE_MATRIX_TIME_Q,	<a href="#">DOUBLE_MATRIX_TIME_Q_RECORD</a>
DOUBLE_QUATERNIONS_TIME_Q,	<a href="#">DOUBLE_QUATERNIONS_TIME_Q_RECORD</a>
DOUBLE_POSITION_ANGLES_TIME_Q,	<a href="#">DOUBLE_POSITION_ANGLES_TIME_Q_RECORD</a>
DOUBLE_POSITION_MATRIX_TIME_Q,	<a href="#">DOUBLE_POSITION_MATRIX_TIME_Q_RECORD</a>
DOUBLE_POSITION_QUATERNION_TIME_Q,	<a href="#">DOUBLE_POSITION_QUATERNION_TIME_Q_RECORD</a>
SHORT_ALL,	<a href="#">SHORT_ALL_RECORD</a>
DOUBLE_ALL,	<a href="#">DOUBLE_ALL_RECORD</a>
DOUBLE_ALL_TIME_STAMP,	<a href="#">DOUBLE_ALL_TIME_STAMP_RECORD</a>
DOUBLE_ALL_TIME_STAMP_Q,	<a href="#">DOUBLE_ALL_TIME_STAMP_Q_RECORD</a>
DOUBLE_ALL_TIME_STAMP_Q_RAW,	<a href="#">DOUBLE_ALL_TIME_STAMP_Q_RAW_RECORD</a>
DOUBLE_POSITION_ANGLES_TIME_Q_BUTTON,	<a href="#">DOUBLE_POSITION_ANGLES_TIME_Q_BUTTON_RECORD</a>
DOUBLE_POSITION_MATRIX_TIME_Q_BUTTON,	<a href="#">DOUBLE_POSITION_MATRIX_TIME_Q_BUTTON_RECORD</a>
DOUBLE_POSITION_QUATERNION_TIME_Q_BUTTON,	<a href="#">DOUBLE_POSITION_QUATERNION_TIME_Q_BUTTON_RECORD</a>
MAXIMUM_FORMAT_CODE	End of table place holder

**Requirements**

**Windows NT/2000:** Requires Windows 2000 or later.

**Header:** Declared in ATC3DGm.h

**Library:** Use ATC3DGm.lib

**See Also**

## BOARD\_TYPES

A value of the **BOARD\_TYPES** enumeration type is returned from a call to GetBoardConfiguration in the *type* parameter location of the structure BOARD\_CONFIGURATION.

```
enum BOARD_TYPES
{
    ATC3DG_MEDSAFE,                // Standalone, DSP, 4 sensor
    BIRD_UNKNOWN                    // default
};
```

Enumerator Value	Meaning
ATC3DG_MEDSAFE	3D Guidance medSAFE
BIRD_UNKNOWN	Unknown board

### Requirements

**Windows NT/2000:** Requires Windows 2000 or later.

**Header:** Declared in ATC3DGm.h

**Library:** Use ATC3DGm.lib

### See Also

## DEVICE\_TYPES

A value of the **DEVICE\_TYPES** enumeration type is returned in the *type* parameter location of either the **SENSOR\_CONFIGURATION** or the **TRANSMITTER\_CONFIGURATION** structures, which are returned from a call to either **GetSensorConfiguration** or **GetTransmitterConfiguration**.

```
enum DEVICE_TYPES{
    STANDARD_SENSOR,
    TYPE_800_SENSOR,
    STANDARD_TRANSMITTER,
    MINIBIRD_TRANSMITTER,
    SMALL_TRANSMITTER,
    TYPE_500_SENSOR
};

enum DEVICE_TYPES
{
    STANDARD_SENSOR,                // 25mm standard sensor
    TYPE_800_SENSOR,                // 8mm sensor
    STANDARD_TRANSMITTER,           // TX for 25mm sensor
    MINIBIRD_TRANSMITTER,           // TX for 8mm sensor
    SMALL_TRANSMITTER,              // "compact" transmitter
    TYPE_500_SENSOR,                // 5mm sensor
    TYPE_180_SENSOR,                // 1.8mm microsensor
    TYPE_130_SENSOR,                // 1.3mm microsensor
    TYPE_TEM_SENSOR,                // 1.8mm, 1.3mm, 0.Xmm microsensors
    UNKNOWN_SENSOR,                 // default
    UNKNOWN_TRANSMITTER             // default
};
```

Enumerator Value	Meaning
STANDARD_SENSOR	Standard Flock sensor with mini-DIN connector
TYPE_800_SENSOR	8mm sensor
STANDARD_TRANSMITTER	Standard Mid Range transmitter
MINIBIRD_TRANSMITTER	Standard miniBird transmitter
SMALL_TRANSMITTER	Compact transmitter
TYPE_500_SENSOR	5mm sensor with mini-DIN connector
TYPE_180_SENSOR	1.8 mm sensor
TYPE_130_SENSOR	1.3 mm sensor
TYPE_TEM_SENSOR	<obsolete>
UNKNOWN_SENSOR	Unknown sensor
UNKNOWN_TRANSMITTER	Unknown transmitter

## Requirements

**Windows NT/2000:** Requires Windows 2000 or later.

**Header:** Declared in ATC3DGm.h  
**Library:** Use ATC3DGm.lib

**See Also**

## 3D Guidance API Status/Error Bit Definitions

The following bit definitions are used with the tracking system.

[ERRORCODE](#)

[DEVICE\\_STATUS](#)

## ERRORCODE

The **ERRORCODE** *int* has the following format:

Bit	Meaning
0-15	Enumerated error code of type <a href="#">BIRD_ERROR_CODES</a>
16-19	Address ID of device reporting error.
20-25	Reserved (Unused)
26	If bit = 1 there are more error messages pending <obsolete>
27 - 29	Error source code: 000 = System error 001 = Tracker board error 010 = Sensor error 100 = Transmitter error Note: All other source codes are invalid
30 - 31	Bits 30 and 31 provide the following advisory error level code Note: It is recommended that all error messages be resolved before proceeding. 00 = Warning 01 = Warning 10 = Fatal Error

## DEVICE\_STATUS

The **DEVICE\_STATUS** is a *typedef* for an *unsigned long* (32 bits) and has the following error bit definitions:

Bit	Name	Meaning	S	B	R	T
0	GLOBAL_ERROR	Global error bit. If any other the other error status bits are set then this bit will be set.	x	x	x	x
1	NOT_ATTACHED	No physical device attached to this device channel.			x	x
2	SATURATED	Sensor currently saturated.			x	
3	BAD_EEPROM	PCB or attached device has a corrupt or unresponsive EEPROM		x	x	x
4	HARDWARE	Unspecified hardware fault condition is preventing normal operation of this device channel, board or the system.	x	x	x	x
5	NON_EXISTENT	The device ID used to obtain this status word is invalid. This device channel or board does not exist in the system.		x	x	x
6	UNINITIALIZED	The system has not been initialized yet. The system must be initialized at least once before any other commands can be issued. The system is initialized by calling <a href="#">InitializeBIRDSystem</a>	x	x	x	x
7	NO_TRANSMITTER	Either an invalid system – transmitter command was issued or an attempt was made to call <a href="#">GetAsynchronousRecord</a> or <a href="#">GetSynchronousRecord</a> when no transmitter was running.	x	x	x	
8	BAD_12V	The +12V power supply has not been connected to this transmitter channel or to this board. In the case of the system error it indicates there is no +12V anywhere in the system and the system cannot run.	x	x		x
9	CPU_TIMEOUT	CPU ran out of time while executing the position and orientation algorithm.		x	x	
10	INVALID_DEVICE	Invalid sensor or transmitter has been attached to this sensor/transmitter channel. This will be set for example if a standard sensor is plugged into an 8mm PCIBird card.			x	x
11 - 31	<reserved>	Always returns zero.	x	x	x	x

The 4 columns with the headings S, B, R and T indicate whether or not the bits are applicable depending on which device status is being acquired. S = system, B = board, R = sensor and T = transmitter.



## 3D Guidance Initialization Files

The Initialization File is used to set a Tracker system to a predetermined state.

### 3D Guidance Initialization File Format Reference

The following sections describe the syntax and meaning of the items used in each type of initialization file section. Initialization files must follow these general rules:

- ❖ Sections begin with the section name enclosed in brackets.
- ❖ A **System** section must be included in any initialization file used with the Tracker hardware. The **System** section contains mandatory items that must be present for the file to be valid. These items are used to verify the applicability of this file to the system being initialized.

The following initialization sections are used to initialize the Tracker system:

[\[System\]](#)

**[ErrorHandling]** (Reserved for future enhancements)

[\[Sensor \$x\$ \]](#) (Where  $x$  is replaced with a decimal number representing the *id* of the sensor.)

[\[Transmitter \$x\$ \]](#) (Where  $x$  is replaced with a decimal number representing the *id* of the transmitter.)

## [System]

The **System** section must be included in all initialization files formatted for use with the Tracker hardware.

```
[System]
NumberOfBoards=number-boards
TransmitterIDRunning=Tx-ID
MeasurementRate=sample-rate
ReportRate=report-rate
Metric=metric-switch
PowerLineFrequency=power
AGCMode=mode
MaximumRange=range
```

*number-boards*

This parameter is a decimal number and represents the number of Tracker cards installed in the PC. This number must match the current number of cards for the file to be accepted. This item is mandatory.

*Tx-ID*

This parameter is a decimal number and represents the index number of the transmitter selected to run after initialization. It assumes that a transmitter is attached at that index location. If no transmitter is attached a bad status will be generated for the sensors. If this value is set to -1 then no transmitter will be selected and all transmitters (if any are attached) will be turned off.

*sample-rate*

This parameter selects the system measurement rate. It will determine how fast the transmitters are driven and the rate at which a new data sample will be produced. The parameter is an positive floating point value describing the measurement rate in Hz.

*report-rate*

This parameter selects the system report rate. It will determine the number of tracker updates before a new data sample will be returned when streaming data. The parameter is positive integer value describing the report rate.

*metric-switch*

This parameter is a Boolean switch, which may have either one of two values. The valid settings are YES or NO. When the value YES is selected the position data will be output with millimeter dimensions. If the value is set to NO the output will be in inches.

*power*

This parameter is a floating point value representing the AC power line frequency in Hz. Currently, only two values are valid: These are 50 and 60 Hz.

*mode*

This parameter is a string describing the AGC mode to be used for the system.

*range*

This parameter is a floating point value representing the maximum range that the system will report in inches. Currently the only valid value is 36 (inches).

The following example shows a typical **System** section:

```
[System]
NumberOfBoards=1
TransmitterIDRunning=0
```

```
ReportRate=1  
MeasurementRate=103.3  
Metric=YES  
PowerLineFrequency=60  
AGCMode=SENSOR_AGC_ONLY  
MaximumRange=36
```

## [Sensorx]

The **Sensor** section is optional.

```
[Sensorx]
Format=format-type
Hemisphere=hemisphere-type
AC_Narrow_Filter=narrow-flag
AC_Wide_Filter=wide-flag
DC_Filter=dc-flag
Alpha_Min=min-params
Alpha_Max=max-params
Vm=vm-params
Angle_Align=align-angles
Filter_Large_Change=change-flag
Distortion=distortion-params
```

### *Format-type*

This parameter takes the form of [DATA\\_FORMAT\\_TYPE](#) enumerated constant listed in the ATC3DGm.h file. Use the exact spelling and case as found in the header file.

### *Hemisphere-type*

This parameter takes the form of the [HEMISPHERE\\_TYPE](#) enumerated constant listed in the ATC3DGm.h file. Use the exact spelling and case as found in the header file.

### *Narrow-flag*

This parameter is a Boolean and is selected by entering either yes or no.

### *Wide-flag*

This parameter is a Boolean and is selected by entering either yes or no.

### *dc-flag*

This parameter is a Boolean and is selected by entering either yes or no.

### *Min-params*

These parameters are entered as a sequence of 6 comma separated floating point numbers in the range 0 to +1.0. Note: A Min\_param cannot exceed its equivalent Max\_param in value.

### *max-params*

These parameters are entered in the same format as the Min\_params. Note a Max\_param may never have a value lower than its equivalent Min\_param.

### *vm-params*

These parameters are entered as 6 comma-separated integers. The valid range for the integers is from a minimum of 1 to a maximum of 32767.

### *Align-angles*

These parameters are entered as 3 comma-separated floating point values. The parameters represent azimuth, elevation and roll. The azimuth and roll values must lie with the range -180 to +180 degrees and the elevation value must lie within the range -90 to +90 degrees.

### *Change-flag*

This parameter is a Boolean and is selected by entering either yes or no.

*Distortion-params*

These parameters are entered as 4 comma-separated integers. The 4 values are defined as follows: error-slope, error-offset, error-sensitivity and filter-alpha. The slope should have a value between -256 and +256. (Default is 164) The offset should have a value between -127 and +127. (The default is 0) The sensitivity should have a value between 0 and +127 (Default is 32) and the alpha should have a value between 0 and 512. (The default is 327)

The following example shows a typical **Sensor** section from a configuration file:

```
[Sensor2]
Format=SHORT_POSITION_ANGLES
Hemisphere=FRONT
AC_Narrow_Filter=no
AC_Wide_Filter=yes
DC_Filter=yes
Alpha_Min=0.02,0.02,0.02,0.02,0.02,0.02
Alpha_Max=0.09,0.09,0.09,0.09,0.09,0.09
Vm=2,4,8,32,64,256,512
Angle_Align=0,0,0
Filter_Large_Change=NO
Distortion=164,0,32,327
```

## [Transmitterx]

The **Transmitter** section is optional.

```
[Transmitterx]  
XYZ_Reference=reference-flag  
XYZ_Reference_Angles=reference-angles
```

### *Reference-flag*

This parameter is a Boolean and should be entered as yes or no. If yes is selected a new sensor position will be calculated for the new reference frame defined by the reference frame angles.

### *Reference-angles*

These parameters take the form a sequence of 3 comma-separated floating point values, which represent the azimuth, elevation and roll of the new transmitter reference frame. The azimuth and roll must have values in the range -180 to +180 degrees. The elevation value must be in the range -90 to +90 degrees.

The following example shows a typical **Transmitter** section from a configuration file:

```
[Transmitter1]  
XYZ_Reference=no  
XYZ_Reference_Angles=0,45,0
```

# 6

## Chapter 6: Ascension RS232 Interface Reference

*This chapter describes how you can communicate with your tracker using the RS232 interface protocol and command set.*

### RS232 Signal Description

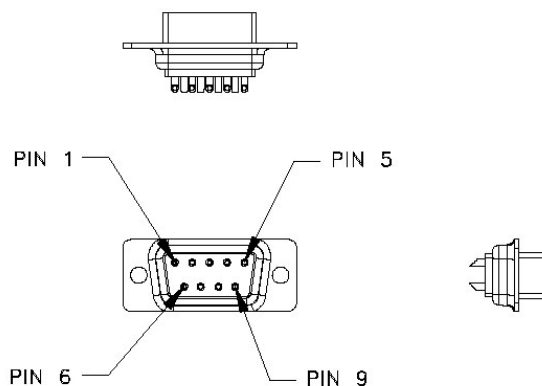
A pinout and signal description of the RS-232C interface is found below. Note that the tracker requires connections only to pins 2, 3 and 5 of the 9-pin interface connector.

The tracker's 9-pin RS-232C connector is arranged as follows:

<u>PIN</u>	<u>RS232 SIGNAL<sup>1</sup></u>	<u>DIRECTION</u>	<u>DESCRIPTION</u>
2	Receive Data	Tracker to host	Serial data output from The Tracker to the host
3	Transmit Data	host to Tracker	Serial data output from the host to The Tracker
5	Signal Ground	Tracker to host	Signal reference
7	Request to Send	host to Tracker	Holds The Tracker in RESET when high

Note:

- 1) These are the Electronic Industries Association (EIA) RS232 signals names. The tracker is configured as Data Communication equipment (DCE) and therefore Transmit Data is an input and Receive Data is an output.



REAR VIEW : 9 PIN D-SUBMINIATURE CONNECTOR

### Using the 'reset on CTS' feature

medSAFE can be configured to perform a system reset when pin 7, its CTS line (RTS on HOST side) is held high. This provides a method of reinitializing the system if the state of the firmware is unknown. This feature is enabled only through placement of an internal jumper on the main PCB. Please contact our Tech Support group for more information.

NOTE: Users running applications in a Windows environment that would like to enable this feature, should take steps to ensure that command of this line (and of the serial port) is clearly asserted.

## RS232 Commands

Each RS232 command consists of a single **command byte** followed by **N command data bytes**, where N depends upon the command. A command is an 8-bit value that the host computer transmits to your tracker.

The RS232 **command format** is as follows:

	MS BIT								LS BIT	
	Stop	7	6	5	4	3	2	1	0	Start
RS232										
Command	1	BC7	BC6	BC5	BC4	BC3	BC2	BC1	BC0	0

where BC7-BC0 is the 8-bit command value (see RS232 Command Reference)  
and the MS BIT (Stop = 1) and LS BIT (Start = 0) refers to the bit values that the UART in your computer's RS232 port automatically inserts into the serial data stream.

The RS232 **command data format** is as follows:


	MS BIT								LS BIT	
	Stop	7	6	5	4	3	2	1	0	Start
RS232										
Data	1	BD7	BD6	BD5	BD4	BD3	BD2	BD1	BD0	0

where BD7-BD0 is the 8-bit data value associated with a given command.



## Command Summary

The following summarizes the action of each RS232 command. The details of command usage are presented later in this chapter in the section entitled '[RS232 Command Reference](#)'.

 **Tip:** For a listing of valid system parameters to use with the CHANGE or EXAMINE VALUE commands, see [CHANGE VALUE](#) in the '[Command Reference](#)' section.

<u>Command Name</u>	<u>Description</u>
<a href="#"><u>ANGLES</u></a>	Data record contains 3 rotation angles.
<a href="#"><u>ANGLE ALIGN</u></a>	Aligns Tracker to reference direction.
<a href="#"><u>BORESIGHT</u></a>	Aligns sensor to the reference frame
<a href="#"><u>BORESIGHT REMOVE</u></a>	Remove the sensor BORESIGHT'
<a href="#"><u>CHANGE VALUE</u></a>	Changes the value of a selected Tracker system parameter.
<a href="#"><u>EXAMINE VALUE</u></a>	Reads and examines a selected Tracker system parameter.
<a href="#"><u>HEMISPHERE</u></a>	Changes the hemisphere for dipole tracking.
<a href="#"><u>MATRIX</u></a>	Data record contains 9-element rotation matrix.
<a href="#"><u>METAL</u></a>	Enables reporting of the metal/quality/distortion data.
<a href="#"><u>OFFSET</u></a>	Configures positional outputs from the tracker to specify a location that is offset from the center of the Sensor.
<a href="#"><u>POINT</u></a>	One data record is output from the 3DGuidance unit for each B command issued.
<a href="#"><u>POSITION</u></a>	Data record contains X, Y, Z position of sensor.
<a href="#"><u>POSITION/ANGLES</u></a>	Data record contains POSITION and ANGLES.
<a href="#"><u>POSITION/MATRIX</u></a>	Data record contains POSITION and MATRIX.
<a href="#"><u>POSITION/QUATERNION</u></a>	Data record contains POSITION and QUATERNION.
<a href="#"><u>QUATERNION</u></a>	Data record contains Quaternions.
<a href="#"><u>READ VPD</u></a>	Reads a single byte from Vital Product Data section of board, transmitter, sensor or preamplifier EEPROM.

**REFERENCE FRAME**

Defines new measurement reference frame.

**REPORT RATE**

Number of data records/second output in STREAM mode.

**\*RESET**

Performs a system reset

**RS232 TO FBB**

Selects sensor.

**RUN**

Turns Transmitter ON and starts running after SLEEP.

**SLEEP**

Turns Transmitter OFF and suspends system operation.

**STREAM**

Data records are transmitted continuously from the selected 3DGuidance™ unit. If GROUP mode is enabled then data records are output continuously from all running 3DGuidance™ units.

**STREAM STOP**

Stops any data output that was started with the STREAM.


**WRITE VPD**

Write a single byte to Vital Product Data section of board, transmitter, sensor or preamplifier EEPROM.

\*Reset command not supported at this time.

## Command Utilization

Your host computer may tell the tracker what type of data to send when a data request is issued. Sending one of the following data record commands indicates the desired type of data: ANGLES, MATRIX, POSITION, QUATERNION, POSITION/ANGLES, POSITION/MATRIX or POSITION/QUATERNION. These commands do not cause the tracker to transmit data to the host. For the host to receive data, it must issue a data request. Use the POINT data request each time you want one data record or use the STREAM data request to initiate a continuous flow of data records from the tracker. If you want to reduce the rate at which data STREAMs from the tracker, use the REPORT RATE command. All commands can be issued in any order and at any time to change the tracker's output characteristics.

 **Tip:** Check out the sample program ["Terminal"](#) in Chapter 4 (and on the CD-ROM) for further illustration of the command usage.

The following is a hypothetical command sequence, issued after power-up, which illustrates the use of some of the commands.

COMMAND	ACTION
ANGLES	Output records will contain angles only.
POINT	Tracker outputs ANGLES data record.
STREAM	ANGLE data records start streaming from Tracker and will not stop until the mode is changed to POINT or the STREAM STOP command is issued.
POINT	An ANGLE data record is output and the streaming is stopped.

## Response Format

Two types of binary data are returned from the tracker:

1. **Position/Orientation** data
2. **CHANGE/EXAMINE VALUE** data

**Position/orientation** data are the data returned from the tracker in the ANGLES, POSITION, MATRIX, POSITION/ANGLES, POSITION/MATRIX, POSITION/QUATERNION and QUATERNION formats. This data is returned in one or more 8-bit data bytes, using a special format described below.

All other types of data that the tracker returns are in the **CHANGE/EXAMINE VALUE** data format. This data is also returned in one or more 8-bit data bytes, using the response format described with each Change/Examine value command. (see the [RS232 Command Reference](#) section for details).

The Change/Examine value data is not shifted and does not contain the 'phasing' bits found in the Position/Orientation data.

### Position/Orientation Data Format

The Position/Orientation information generated by the tracker is sent in a form called a data record. The number of bytes in each record is dependent on the output format selected by the user. Each 2-byte word is in a binary format dependent on the word type (i.e. Position, Angles, etc.). The binary formats consist of the 14 most significant bits (bits B15 - B2) of the sixteen bits (bits B15 - B0), which define each word. The tracker does not use the two least significant bits (bits B1 and B0). The first bit of the first byte transmitted is always a one (1) while the first bit of all other transmitted bytes in the record is always a zero (0). These "phasing" bits are required for the host computer to identify the start of a record when the data is streaming from the tracker without individual record requests. In general, the output data will appear as follows:

MS BIT						LS BIT		WORD #
7	6	5	4	3	2	1	0	
1	B8	B7	B6	B5	B4	B3	B2	#1 LSbyte
0	B15	B14	B13	B12	B11	B10	B9	#1 MSbyte
0	C8	C7	C6	C5	C4	C3	C2	#2 LSbyte
0	C15	C14	C13	C12	C11	C10	C9	#2 MSbyte
0	.	.	.	.	.	.	.	.
0	.	.	.	.	.	.	.	.
0	.	.	.	.	.	.	.	.
0	N8	N7	N6	N5	N4	N3	N2	#N LSbyte
0	N15	N14	N13	N12	N11	N10	N9	#N MSbyte

The MS (most significant) bits are the phasing bits, and are not part of the data.

For example, the tracker is about to send a data record consisting of these three data words:

Word#	Decimal	Hex	Binary (2 bytes)	
			MSbyte	LSbyte
#1	4386	1122	00010001	00100010
#2	13124	3344	00110011	01000100
#3	21862	5566	01010101	01100110

The conversion to the binary data format by the tracker is as follows:

#### TRACKER

1) Shifts each data word right one bit

```

MS      LS
00001000 10010001
00011001 10100010
00101010 10110011
```

2) Breaks each word into MSByte LSByte pairs

```

10010001 LS
00001000 MS
10100010 LS
00011001 MS
10110011 LS
```

- 3) Shifts each LSByte right one more bit (Marks with "1" if first byte)
- 4) Transmits all bytes in stream

00101010 MS

MS BIT				LS BIT				WORD #
7	6	5	4	3	2	1	0	
1	1	0	0	1	0	0	0	#1 LSByte
0	0	0	0	1	0	0	0	#1 MSByte
0	1	0	1	0	0	0	1	#2 LSByte
0	0	0	1	1	0	0	1	#2 MSByte
0	1	0	1	1	0	0	1	#3 LSByte
0	0	1	0	1	0	1	0	#3 MSByte

The user's computer can identify the beginning of the data record by catching the leading "1", and converting subsequent data bytes back to their proper binary values.

HOST:

- 1) Receives data bytes in stream after catching first marked "1" (Changes that "1" back to a "0")
- 2) Shifts each LSByte left one bit
- 3) Combines each MSByte/LSByte pair into data words
- 4) Shifts each word left one more bit, giving the correct original binary value

01001000 LS  
00001000 MS  
01010001 LS  
00011001 MS  
01011001 LS  
00101010 MS

10010000 LS  
00001000 MS  
10100010 LS  
00011001 MS  
10110010 LS  
00101010 MS

MS LS  
00001000 10010000  
00011001 10100010  
00101010 10110010

MS LS  
00010001 00100000  
00110011 01000100  
01010101 01100100

You don't need to worry about the fact that the two least significant bits are different because the data words do not use these bits.

## RS232 Command Reference

All commands are listed alphabetically in the following section. Each command description contains the command codes required to initiate the commands, as well as the format and scaling of the data records that the tracker will output to the host computer.

## ANGLES

	ASCII	HEX	DECIMAL	BINARY
Command Byte	W	57	87	01010111

In the ANGLES mode, the tracker outputs the orientation angles of the sensor with respect to the Transmitter. The orientation angles are defined as rotations about the Z, Y, and X axes of the sensor. These angles are called Zang, Yang, and Xang or, in Euler angle nomenclature, Azimuth, Elevation, and Roll. The output record is in the following format for the six transmitted bytes:

MSB	7	6	5	4	3	2	1	LSB	0	BYTE #
1	Z8	Z7	Z6	Z5	Z4	Z3	Z2			#1 LSbyte Zang
0	Z15	Z14	Z13	Z12	Z11	Z10	Z9			#2 MSbyte Zang
0	Y8	Y7	Y6	Y5	Y4	Y3	Y2			#3 LSbyte Yang
0	Y15	Y14	Y13	Y12	Y11	Y10	Y9			#4 MSbyte Yang
0	X8	X7	X6	X5	X4	X3	X2			#5 LSbyte Xang
0	X15	X14	X13	X12	X11	X10	X9			#6 MSbyte Xang

Zang (Azimuth) takes on values between the binary equivalent of +/- 180 degrees. Yang (Elevation) takes on values between +/- 90 degrees, and Xang (Roll) takes on values between +/- 180 degrees. As Yang (Elevation) approaches +/- 90 degrees, the Zang (Azimuth) and Xang (Roll) become very noisy and exhibit large errors. At 90 degrees the Zang (Azimuth) and Xang (Roll) become undefined. This behavior is not a limitation of the tracker. Rather, it is an inherent characteristic of Euler angles. If you need a stable representation of the sensor orientation at high Elevation angles, use the MATRIX output mode.

The scaling of all angles is full scale = 180 degrees. That is, +179.99 deg = 7FFF Hex, 0 deg = 0 Hex, -180.00 deg = 8000 Hex.

**To convert the numbers into angles (degrees)** first cast it into a signed integer. This will give you a number from +/- 32767. Second multiply by 180 and finally divide the number by 32768 to get the angle. The equation should look something like this:

$$(\text{signed int}(\text{Hex \#}) * 180) / 32768$$


## ANGLE ALIGN

	ASCII	HEX	DECIMAL	BINARY
Command Byte	q	71	113	01110001
Command Data	A, E, R			

By default, the angle outputs from each sensor are measured in the coordinate frame defined by the Transmitter's X, Y and Z axes (see [Default Reference Frames](#)) and are measured with respect to rotations about the physical X, Y and Z axes of the sensor. The ANGLE ALIGN1 command allows you to mathematically change each sensor's X, Y and Z axes to an orientation that differs from that of the actual sensor.

For example:

Suppose that during installation you find it necessary, due to physical requirements, to cock the sensor, resulting in its angle outputs reading Azim = 5 deg, Elev = 10 and Roll = 15 when it is in its normal "resting" position. To compensate, use the ANGLE ALIGN command, passing as Command Data the angles of 5, 10 and 15 degrees. After this sequence is sent, the sensor outputs will be zero, and orientations will be computed as if the sensor were not misaligned.

 **Note:**  
The ANGLE ALIGN command only affects the computation of orientation - it has no effect on position.

The host computer must send the *Command Data* immediately following the *Command Byte*. Command data consists of the angles.

The Command Byte and Command Data must be transmitted to the tracker in the following seven-byte format:

 **Note:**  
The ANGLE ALIGN command is ignored for 5DOF sensors.

MSB				LSB				BYTE #
7	6	5	4	3	2	1	0	
0	1	1	1	0	0	1	0	#1 Command Byte
B7	B6	B5	B4	B3	B2	B1	B0	#2 LSbyte A
B15	B14	B13	B12	B11	B10	B9	B8	#3 MSbyte A
B7	B6	B5	B4	B3	B2	B1	B0	#4 LSbyte E
B15	B14	B13	B12	B11	B10	B9	B8	#5 MSbyte E
B7	B6	B5	B4	B3	B2	B1	B0	#6 LSbyte R
B15	B14	B13	B12	B11	B10	B9	B8	#7 MSbyte R

See the ANGLES command for the format and scaling of the angle values sent.



## BORESIGHT

	ASCII	HEX	DECIMAL	BINARY
Command Byte	u	75	117	01110101

Sending the single byte BORESIGHT command to your tracker causes the sensor to be aligned to the tracker's REFERENCE FRAME. In other words, when you send the command, the sensor's orientation outputs will go to zero, making it appear as if it was physically aligned with the Tracker's REFERENCE FRAME. All orientation outputs thereafter are with respect to this BORESIGHT orientation. This command is equivalent to taking the angle outputs from the Tracker and using them in the ANGLE ALIGN commands but without the need to supply any angles with the command. This command does not change any angles you may have set using the ANGLE ALIGN command. However, if you use the ANGLE ALIGN command after you send the BORESIGHT command, these new ANGLE ALIGN will remove the effect of the BORESIGHT command and replace them with the ANGLE ALIGN angles.

Use the BORESIGHT REMOVE command to revert to the sensor outputs as measured by the orientation of the sensor

## BORESIGHT REMOVE

	ASCII	HEX	DECIMAL	BINARY
Command Byte	v	76	118	01110110

Sending the single byte BORESIGHT REMOVE command to your tracker causes the sensor's orientation outputs to revert to their values before you sent the BORESIGHT command. That is, if there were no ANGLE ALIGN values present, the sensor's orientation outputs will now be with respect to the sensor's physical orientation. If there were ANGLE ALIGN values present before the BORESIGHT command was given, then after the BORESIGHT REMOVE command is given, the sensor's orientation outputs will be with respect to this mathematically defined ANGLE ALIGN sensor orientation.

## BUTTON MODE

	ASCII	HEX	DECIMAL	BINARY
Command Byte	M	4D	77	01001101

Command Data	MODE
--------------	------

The BUTTON MODE command is used to set how the state of an external button ([SWITCH connector](#) on rear panel of tracker) will be reported to the host computer. The BUTTON MODE Command Byte must be followed by a single Command Data byte which specifies the desired report format. The button state is reported to the host via a single Button Value byte. This byte can be sent by the Tracker after the last data record element is transmitted, or can be read at any time using the BUTTON READ command. If you set the Command Data byte equal to 0 Hex, the Button Value byte is not appended to the data record, and you must use the BUTTON READ command to examine the status of the button. If you set the Command Data byte equal to 1, the Button Value byte will be appended to the end of each transmitted data record unless the Metal indicator byte is output also, in which case the Metal indicator byte will be the last byte and the Button value byte will be next to last. For example, you had selected the POSITION/ANGLE mode, the output sequence would now be: x, y, z, az, el, rl, button, for a total of 13 bytes instead of the normal 12 bytes.

The BUTTON MODE command must be issued to the Tracker in the following 2-byte sequence:

MSB								LSB	
7	6	5	4	3	2	1	0		BYTE #
0	1	0	0	1	1	0	1		#1 Command Byte
0	0	0	0	0	0	0	D0		#2 Command Data

Where D0 is either 0 or 1.

For a description of the values which may be returned in the Button Value byte, see the BUTTON READ command.

## BUTTON READ

	ASCII	HEX	DECIMAL	BINARY
Command Byte	N	4E	78	01001110

The **BUTTON READ** command allows you to determine at any time the state of an external button (contact closure/switch) that the user has connected to the [SWITCH connector](#) on rear panel of tracker. This command is especially useful when you want to read the buttons but do not have **BUTTON MODE** set to 1 (which would append the Button Value byte to every transmitted record).

Immediately after you send the **BUTTON READ** Command Byte, the Tracker will return a single byte containing the button value. The Button Value byte can assume the following Hex values:

- 0 Hex = 0: No button pressed.
- 1 Hex = 1: Button pressed



**Note:** The Button Value byte does not contain the phasing bits normally included in the Bird's transmitted data records. The above values are the ones actually sent to the host.

The Tracker updates its button reading every transmitter axis cycle (3 times per measurement cycle for mid and short-range transmitters), whether you request the value or not. Thus, the system does not store previous button presses, and indicates only whether the button has been pressed within the last transmitter axis cycle.

## CHANGE VALUE

	ASCII	HEX	DECIMAL	BINARY
CHANGE VALUE Command Byte	P	50	80	01010000

CHANGE VALUE Command Byte	PARAMETERnumber	PARAMETERvalue
------------------------------	-----------------	----------------

The CHANGE VALUE command allows you to change the value of the tracker parameter defined by the PARAMETERnumber byte and the PARAMETERvalue byte(s) sent with the command.

## EXAMINE VALUE

	ASCII	HEX	DECIMAL	BINARY
EXAMINE VALUE Command Byte	O	4F	79	01001111

EXAMINE VALUE Command Byte	PARAMETERnumber
-------------------------------	-----------------

The EXAMINE VALUE command allows you to read the value of the tracker parameter defined by the PARAMETERnumber sent with the command. Immediately after the tracker receives the command and command data, it will return the parameter value as a multi-byte response.

## VALID PARAMETERS

Valid CHANGE VALUE and EXAMINE VALUE PARAMETERnumbers are listed in the table below.

*Note: not all PARAMETERnumbers are CHANGEable, but ALL are EXAMINEable.*

PARAMETER #		CHANGEable	CHANGE bytes send	EXAMINE bytes send	EXAMINE bytes receive	PARAMETER DESCRIPTION
Dec	Hex					
0	0	No	0	2	2	Tracker status
1	1	No	0	2	2	Software revision number
2	2	No	0	2	2	Tracker computer crystal speed
3	3	Yes	4	2	2	Position scaling
4	4	Yes	4	2	2	Filter on/off status
5	5	Yes	16	2	14	DC Filter constant table ALPHA_MIN
7	7	Yes	4	2	2	Measurement rate
8	8	Yes	3	2	1	Data ready output character
9	9	Yes	3	2	1	Changes data ready character
10	A	No	0	2	1	Tracker outputs an error code
12	C	Yes	16	2	14	DC filter constant table Vm
13	D	Yes	16	2	14	DC filter constant table ALPHA_MAX

14	E	Yes	3	2	1	Sudden output change elimination
15	F	No	0	2	10	System Model Identification
17	11	Yes	3	2	1	XYZ Reference Frame
20	14	Yes	3	2	1	Filter line frequency
22	16	Yes	4	2	2	Hemisphere
23	17	Yes	8	2	6	Angle Align2
24	18	Yes	8	2	6	Reference Frame2
25	19	No	0	2	2	Tracker (Electronics) Serial Number
26	1A	No	0	2	2	Sensor Serial Number
27	1B	No	0	2	2	Xmtr Serial Number
28	1C	Yes	12	2	10	Metal Detection
29	1D	Yes	1	2	1	Report Rate
35	23	Yes	3	2	1	Group Mode
36	24	No	0	2	14	System Status
50	32	Yes	3	2	5	AutoConfig
71	47	Yes	8	2	6	Sensor Offsets
130	82	No	0	2	2	Boot Loader Firmware Revision
131	83	No	0	2	2	MDSP Firmware Revision
133	85	No	0	2	2	NonDipole POserver Firmware Revision
135	87	No	0	2	2	5DOF Firmware Revision
136	88	No	0	2	2	6DOF Firmware Revision
137	89	No	0	2	2	Dipole POserver Firmware Revision

The **CHANGE VALUE** command must be issued to the tracker in the following N-byte sequence:

MSB							LSB	
7	6	5	4	3	2	1	0	BYTE #
0	1	0	1	0	0	0	0	#1 Command Byte, 'P'
N7	N6	N5	N4	N3	N2	N1	N0	#2 PARAMETERnumber
B7	B6	B5	B4	B3	B2	B1	B0	#3 PARAMETERdata LSbyte
B7	B6	B5	B4	B3	B2	B1	B0	#4 PARAMETERdata MSbyte
B7	B6	B5	B4	B3	B2	B1	B0	#N PARAMETERdata

Where N7-N0 represent a PARAMETERnumber (i.e. 00000011 or 00000100), and B7-B0 represent N-bytes of PARAMETERdata. If the PARAMETERdata is a word then the Least Significant byte (LSbyte) is transmitted before the Most Significant byte (MSbyte). If the PARAMETERdata is numeric, it must be in 2's complement format. You do not shift and add 'phasing' bits to the data.

The **EXAMINE VALUE** command must be issued to the tracker in the following 2-byte sequence:

MSB							LSB	
7	6	5	4	3	2	1	0	BYTE #
0	1	0	0	1	1	1	1	#1 Command Byte
N7	N6	N5	N4	N3	N2	N1	N0	#2 PARAMETERnumber


Where N7-N0 represent a PARAMETERnumber, i.e. 00000000 or 00000001, etc.

If the PARAMETERdata returned is a word then the Least Significant byte (LSbyte) is received before the Most Significant byte (MSbyte). If the PARAMETERdata is numeric, it is in 2's complement format. The PARAMETERdata received does not contain 'phasing' bits. The PARAMETER data value, content and scaling depends on the particular parameter requested. See the following discussion of each parameter.

## TRACKER STATUS

When PARAMETERnumber = 0 during EXAMINE, the tracker returns a status word to tell the user in what mode the unit is operating. The bit assignments for the two-byte response are:

B15	1 if Tracker is a Master Tracker 0 if Tracker is a Slave Tracker
B14	1 if Tracker has been initialized following Auto-Config 0 if Tracker has not been initialized
B13	1 if errors exist in the SYSTEM ERROR register (error(s) detected) 0 if no errors exist in the register (no errors detected)
B12	1 if Tracker is RUNNING 0 if Tracker is not RUNNING
B6-B11	Not currently used by 3DGuidance™
B5	1 if The Tracker is in SLEEP mode. (Opposite of B12) 0 if The Tracker is in RUN mode
B4, B3, B2, B1	0001 if POSITION outputs selected 0010 if ANGLE outputs selected 0011 if MATRIX outputs selected 0100 if POSITION/ANGLE outputs selected 0101 if POSITION/MATRIX outputs selected 0110 factory use only 0111 if QUATERNION outputs selected 1000 if POSITION/QUATERNION outputs selected
B0	0 if POINT mode selected 1 if STREAM mode selected (Note: in STREAM mode you can not examine status)

 **Tip:** To retrieve the error code(s) indicated by this 'flag' (B13), send '10' as the parameter. See [Error Code](#) below.

## SOFTWARE REVISION NUMBER

When PARAMETERnumber = 1 during EXAMINE, the tracker returns the two byte revision number of the software located in its Flash memory. The revision number in base 10 is expressed as INT.FRA where INT is the integer part of the revision number and FRA is the fractional part. For example, if the revision number is 2.13 then INT = 2 and FRA = 13. The value of the most significant byte returned is FRA. The value of the least significant byte returned is INT. Thus, in the above example the value returned in the most significant byte would have been 0D Hex and the value of the least significant byte would have been 02 Hex. If the revision number were 3.1 then the bytes would be 03 and 01 Hex.



## TRACKER COMPUTER CRYSTAL SPEED

When PARAMETERnumber = 2 during EXAMINE, the tracker returns the speed of its computer's crystal in megahertz (MHz). The most significant byte of the speed word is equal to zero, and the base 10 value of the least significant byte represents the speed of the crystal. For example, if the least significant byte = 19 Hex, the crystal speed is 25 MHz. The tracker always reports 325MHz.

## POSITION SCALING

When PARAMETERnumber = 3 during EXAMINE, the tracker returns a code that describes the scale factor used to compute the position of the sensor with respect to the transmitter. If the separation exceeds this scale factor, the tracker's position outputs will not change to reflect this increased distance, rendering the measurements useless. The most significant byte of the parameter word returned is always zero. If the least significant byte = 0, the scale factor is 36 inches for a full-scale position output. If the least significant byte is = 1, the full-scale output is 72 inches

To CHANGE the scale factor send the tracker two bytes of PARAMETERdata with the most significant byte set to zero and the least significant set to zero or one.

Note: Changing the scale factor from the default 36 inches to 72 inches reduces by half the resolution of the output X, Y, Z coordinates.

## FILTER ON/OFF STATUS

When PARAMETERnumber = 4 during EXAMINE, the tracker returns a code that tells you what software filters are turned on or off in the unit. You most likely will not need to change the filters, but it is possible to do so. The most significant byte returned is always zero. The bits in the least significant byte are coded per the following:

### BIT NUMBER    MEANING

B7-B3	0
B2	0 if the AC NARROW notch filter is ON 1 if the AC NARROW notch filter is OFF (default)
B1	0 if the AC WIDE notch filter is ON (default) 1 if the AC WIDE notch filter is OFF
B0	0 if the DC (Adaptive) filter is ON (default) 1 if the DC filter is OFF

The **AC NARROW** notch filter refers to a two-tap finite impulse response (FIR) notch filter that is applied to signals measured by the tracker's sensor to eliminate a narrow band of noise with sinusoidal characteristics. Use this filter in place of the AC WIDE notch filter when you want to minimize the transport delay between tracker measurement of the sensor's position/orientation and the output of these measurements. The transport delay of the AC NARROW notch filter is approximately one third the delay of the AC WIDE notch filter.

The **AC WIDE** notch filter refers to a six tap FIR notch filter that is applied to the sensor data to eliminate sinusoidal signals with a frequency between 30 and 72 hertz. If your application requires minimum transport delay between measurement of the sensor's position/orientation and the output of these measurements, you may want to evaluate the effect on your application with this filter shut off and the AC NARROW notch filter on. If you are running the tracker synchronized to a CRT, you can usually shut this filter off without experiencing an increase in noise.

The **DC** filter refers to an adaptive, low pass filter applied to the sensor data to eliminate high frequency noise. When the DC filter is turned on, you can modify its noise/lag characteristics by changing ALPHA\_MIN and Vm.


To CHANGE the FILTER ON/OFF STATUS send the tracker two bytes of PARAMETERdata with the most significant byte set to zero and the least significant set to the code in the table above.

#### DC FILTER CONSTANT TABLE ALPHA\_MIN

When PARAMETERnumber = 5 during EXAMINE, The tracker returns 7 words (14 bytes) which define the lower end of the adaptive range that filter constant ALPHA\_MIN can assume in the DC filter. When ALPHA\_MIN = 0 Hex, the DC filter will provide an infinite amount of filtering (the outputs will never change even if you move the sensor). When ALPHA\_MIN = 0.99996 = 7FFF Hex, the DC filter will provide no filtering of the data.

The default values are:

ALPHA_MIN (decimal)	
Mid-Range Transmitter Range	ALPHA MIN
(inches)	(decimal)
0 to 17	0.02 = 028F Hex
17 to 22	0.02
22 to 27	0.02
27 to 34	0.02
34 to 42	0.02
42 to 54	0.02
54 +	0.02

 **Note:** Only the first X words (X bytes) are used for the microBIRD. Additional bytes included so as to be compatible with legacy products.

To CHANGE ALPHA\_MIN, send the tracker seven words of PARAMETERdata corresponding to the ALPHA\_MIN table defined above. Increase ALPHA\_MIN to obtain less; decrease ALPHA\_MIN to provide more filtering (less noise/more lag). ALPHA\_MIN must always be less than ALPHA\_MAX.

#### MEASUREMENT RATE

During EXAMINE, the tracker returns a word that is used to determine the measurement rate of the unit. The word returned is the measurement rate in cycles/sec times 256.

The measurement rate in cycles/sec is computed from:

$$\text{measurement rate} = (\text{word returned})/256.$$

To CHANGE the MEASUREMENT RATE, send the tracker one word of PARAMETERdata corresponding to (measurement rate) \* 256.

## DISABLE/ENABLE DATA READY OUTPUT

Enabling the DATA READY character provides a method for notifying you as soon as the newest position and orientation data has been computed. Typically, you would issue a POINT data request as soon as you receive the DATA READY command. If you are running in STREAM mode you should not use the DATA READY character since the position and orientation is sent to you automatically as soon as it is ready.

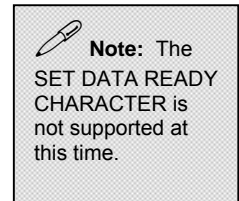
When PARAMETERnumber = 8 during EXAMINE, the tracker outputs one byte of data, equal to 1 if Data Ready Output is enable or a 0 if disabled.

To CHANGE DATA READY, send the tracker one byte of PARAMETERdata = 1 if the tracker is to output the Data Ready Character every measurement cycle as soon as a new measurement is ready for output. The default Data Ready Character is a comma (2C Hex, 44 Dec).

## SET DATA READY CHARACTER

When PARAMETERnumber = 9 during EXAMINE, the tracker returns one byte, the current ASCII value of the Data Ready Character.

To CHANGE the DATA READY CHARACTER, send the tracker one byte of PARAMETERdata equal to the character value that the tracker should use as the Data Ready Character.



## ERROR CODE

When PARAMETERnumber = 10 during EXAMINE, the tracker will output a one byte Error code, indicating a particular system condition was detected. The byte returned represents the earliest error code sent to the SYSTEM ERROR register. See the **Error Reporting** section below, for details.

## DC FILTER TABLE Vm

When PARAMETERnumber = 12 during EXAMINE, the tracker returns a 7 word (14 byte) table, or during CHANGE, the user sends the tracker a 14 byte table representing the expected noise that the DC filter will measure. By changing the table values the user can increase or decrease the DC filter's lag as a function of sensor distance from the transmitter.

The DC filter is adaptive in that it tries to reduce the amount of low pass filtering in the tracker as it detects translation or rotation rates in its sensor. Reducing the amount of filtering results in less filter lag. Unfortunately electrical noise in the environment, when measured by the transmitter, may also

make it look like the sensor is undergoing a translation and rotation. The tracker has to know if the measured events are real due to movement or false due to noise. The tracker gets this knowledge by the user specifying what the expected noise levels are in the operating environment. These noise levels are the 7 words that form the Vm table. The Vm values can range from 1 for almost no noise to 32767 for a lot of noise.

The default values as a function of transmitter to sensor separation range for the standard range transmitters are:

Mid-Range Transmitter (inches)	Vm (integer)
0 to 17	2
17 to 22	4
22 to 27	4
27 to 34	4
34 to 42	4
42 to 54	4
54 +	4

As Vm increases so does the amount of filter lag. To reduce the amount of lag, reduce the larger Vm values until the noise in the tracker's output is too large for your application.

#### DC FILTER CONSTANT TABLE ALPHA\_MAX

When PARAMETERnumber = 13 during EXAMINE, the tracker returns 7 words (14 bytes) that define the upper end of the adaptive range that filter constant ALPHA\_MAX can assume in the DC filter as a function of sensor to transmitter separation. When there is a fast motion of the sensor, the adaptive filter reduces the amount of filtering by increasing the ALPHA used in the filter. It will increase ALPHA only up to the limiting ALPHA\_MAX value. By doing this, the lag in the filter is reduced during fast movements. When ALPHA\_MAX = 0.99996 = 7FFF Hex, the DC filter will provide no filtering of the data during fast movements.

The default values as a function of transmitter to sensor separation range for the standard range and transmitters are:

Mid-Range Transmitter Range (inches)	Std. Range Xmtr ALPHA_MAX (fractional)
0 to 17	0.9 = 07333 Hex.
17 to 22	0.9
22 to 27	0.9
27 to 34	0.9
34 to 42	0.9
42 to 54	0.9
54 +	0.9

To CHANGE ALPHA\_MAX send the tracker seven words of PARAMETERdata corresponding to ALPHA\_MAX. During CHANGE, you may want to decrease ALPHA\_MAX to increase the amount of filtering if its outputs are too noisy during rapid sensor movement. ALPHA\_MAX must always be greater than ALPHA\_MIN.

## SUDDEN OUTPUT CHANGE LOCK

When PARAMETERnumber = 14, during EXAMINE, the tracker returns a byte which indicates if the position and orientation outputs will be allowed to change if the system detects a sudden large change in the outputs. Large undesirable changes may occur at large separation distances between the transmitter and sensor when the sensor undergoes a fast rotation or translation. The byte returned will = 1 to indicate that the outputs will not be updated if a large change is detected. If the byte returned is zero, the outputs will change.

To change SUDDEN OUTPUT CHANGE LOCK send the tracker one byte of PARAMETERdata = 0 to unlock the outputs or send one byte = 1 to lock the outputs.

## SYSTEM MODEL IDENTIFICATION

When PARAMETERnumber = 15 during EXAMINE, the tracker returns 10 bytes which will represent the device that was found.

Device Description String	Device
"6DFOB "	Stand alone (SRT)
"6DERC "	Extended Range Controller
"6DBOF "	MotionStar (old name)
"6DMC180-4"	3D Guidance (4 sensor 3DGuidance <sup>M</sup> )
"PCBIRD "	pcBIRD
"SPACEPAD "	SpacePad
"MOTIONSTAR"	MotionStar (new name)
"WIRELESS "	MotionStar Wireless
" LaserBird2"	laserBIRD 2
"phasorBIRD"	phasorBIRD

## XYZ REFERENCE FRAME

By default, the XYZ measurement frame is the reference frame defined by the physical orientation of the transmitter's XYZ axes even when the REFERENCE FRAME command has been used to specify a new reference frame for measuring orientation angles. When PARAMETERnumber = 17, during CHANGE, if the one byte of PARAMETER DATA sent to the tracker is = 1, the XYZ measurement frame will also correspond to the new reference frame defined by the REFERENCE FRAME command. When the PARAMETER DATA sent is a zero, the XYZ measurement frame reverts to the orientation of the transmitter's physical XYZ axes.

During EXAMINE, the tracker returns a byte value of 0 or 1 to indicate that the XYZ measurement frame is either the transmitter's physical axes or the frame specified by the REFERENCE FRAME command.

## FILTER LINE FREQUENCY

When PARAMETERnumber = 20, during EXAMINE, the tracker returns a byte whose value is the Line Frequency which is being used to determine the Wide Notch Filter coefficients. The default Line Frequency is 60 Hz.

To CHANGE the Line Frequency send 1 byte of PARAMETERdata corresponding to the desired Line Frequency. The range of Line Frequencies available are 1 -> 255.

Example: To change the Line Frequency to 50Hz you would first send a Change Value command (50 Hex), followed by a Filter Line Frequency command (14 Hex), followed by the line frequency for 50 Hz (32 Hex).

## HEMISPHERE

When PARAMETERnumber = 22, during EXAMINE, the tracker will return 2 bytes of data defining the current Hemisphere. These are as follows:

Hemisphere	HEMI_AXIS		HEMI_SIGN	
	ASCII	HEX	ASCII	HEX
Forward	nul	00	nul	00
Aft (Rear)	nul	00	soh	01
Lower	ff	0C	nul	00
Upper	ff	0C	soh	01
Right	ack	06	nul	00
Left	ack	06	soh	01

### Notes:

- 1) Please note that these are the same PARAMETERdata values as are used by the HEMISPHERE command 'L' (4C Hex).

To CHANGE the Hemisphere, send 2 PARAMETERdata bytes as described above.

- 2) This command operates in exactly the same way as the HEMISPHERE command. The command is now included in the CHANGE/EXAMINE command set in order to allow users to examine the values which were previously inaccessible.
- 3) The values can only be EXAMINED with this command if they were previously CHANGED by this command.

## ANGLE ALIGN

When PARAMETERnumber = 23 during EXAMINE, The tracker will return 3 words (6 bytes) of data corresponding to the Azimuth, Elevation, and Roll angles used in the ANGLE ALIGN command. This command differs from the ANGLE ALIGN command only in that it allows both reading and writing of the angles. See ANGLE ALIGN for a full explanation of its use.

To CHANGE the angles send 6 bytes of PARAMETERdata after the 2 command bytes.

## REFERENCE FRAME

When PARAMETERnumber = 24 during EXAMINE, the tracker will return 3 words (6 bytes) of data corresponding to the Azimuth, Elevation and Roll angles used in the REFERENCE FRAME command.

See REFERENCE FRAME2 command for an explanation.

To CHANGE the angles send 6 bytes of PARAMETERdata after the 2 command bytes.

## TRACKER SERIAL NUMBER

When PARAMETERnumber = 25 during EXAMINE, The tracker will return a 1 word (2 byte) value corresponding to the Serial Number of the Electronics Unit.

Note: This number cannot be changed.

## SENSOR SERIAL NUMBER

When PARAMETERnumber = 26, during EXAMINE, the tracker will return a 1 word (2 byte) value corresponding to the Serial Number of its sensor. This number cannot be changed.

## TRANSMITTER SERIAL NUMBER

When PARAMETERnumber = 27, during EXAMINE, the tracker will return a 1 word (2 byte) value corresponding to the Serial Number of its transmitter. You cannot swap transmitters while the tracker is switched ON. If you do you will get the Serial Number of the transmitter that was attached to the tracker when it was first turned on. This number cannot be changed.

## METAL

When PARAMETERnumber=28, during EXAMINE, the tracker that this command is sent to, returns 5 words (10 bytes) of data that define the metal detection parameters. The order of the returned words is:

- METALflag
- METAL sensitivity
- METALoffset
- METALslope
- METALalpha

The least significant byte of each parameter, which is sent first, contains the parameter value. The most significant byte is always zero.

On CHANGE, the user sends the tracker, 5 words of metal detection parameter data as defined above in the EXAMINE command.

If you only want to change one metal parameter at a time, refer to the [METAL](#) command.

## REPORT RATE

When PARAMETERnumber = 29 during EXAMINE, the tracker will return a single byte of data that defines how often its outputs data to your host computer when in STREAM mode. This change parameter value is similar to the REPORT RATE command except the user is not limited to a report rate of every first, second, eighth, or thirty-second cycles.

During CHANGE, the user supplies one byte with this command with any value between 1 and 127 that defines how many updates occur before position and orientation data are output when the tracker is in STREAM mode.

## GROUP MODE

The GROUP MODE command is used if you have multiple sensors and you want to get data from all the sensors by issuing a single request.

When PARAMETERnumber = 35, during EXAMINE VALUE, the tracker will respond with one byte of data indicating if the Tracker is in GROUP MODE. If the data is a 1, the Tracker is in GROUP MODE and if the data is 0 The Tracker is not in GROUP MODE. When in GROUP MODE, in response to the POINT or STREAM commands, the tracker will send data records from all sensors attached to the system. Information is output from the sensor with the smallest address first. The last byte of the data record from each sensor contains the address of that sensor. This address byte contains no phasing bits. Each sensor can be in a different data output format if desired. For example, if 3 sensors are in the system, and the first is configured to output POSITION data only (6 data bytes plus 1 address byte) and the other two are configured to output POSITION/ANGLES data (12 data bytes plus 1 address byte), the system will respond with 33 bytes when a data request is made.

During a CHANGE VALUE command, the host must send one data byte equal to a 1 to enable GROUP MODE or a 0 to disable GROUP MODE.



## SYSTEM STATUS

When PARAMETERnumber = 36, during EXAMINE, the trackers returns to the host computer 14 bytes defining the physical configuration.. This command can be sent to the Master either before or after the tracker is running. The response has the following format, where one byte is returned for each possible FBB address:

BYTE 1	- address 1 configuration
BYTE 2	- address 2 configuration
.	.
.	.
BYTE 14	- address 14 configuration

Each byte has the following format:

BIT 7            If 1, device is accessible on FBB. If 0, device is not accessible. A device is accessible when its fly switch is on. It may or may not be running.

BIT 6    If 1, device is running. If 0, device is not running. A device is running when the power switch is on, it has been AUTO-CONFIGed and it is AWAKE. A device is not running when the power switch is on and it has not been AUTO-CONFIGed or it has been AUTO-CONFIGed and it is ASLEEP.

BIT 5            If 1, device has a sensor. If 0, device does not have a sensor

BIT 4            If 1, transmitter is an ERT. If 0, transmitter is standard range

BIT 3            If 1, ERT #3 is present. If 0, not present

BIT 2            If 1, ERT #2 is present. If 0, not present

BIT 1            If 1, ERT #1 is present. If 0, not present

BIT 0    If 1, ERT #0 or standard range transmitter is present. If 0, not present.

Note that currently, 3D Guidance medSAFE does not support Extended Range Transmitters (ERTs).

## AUTOCONFIG

The AUTO-CONFIGURATION command is used to start running multiple Trackers, i.e., multiple sensors/tracked-objects. In the case of the 3D Guidance medSAFE, there are four Trackers.

When PARAMETERnumber = 50, during an CHANGE VALUE command, the Master Tracker will perform all the necessary configurations for a one transmitter/multiple sensor configuration. The tracker expects one byte of data corresponding to the number of Trackers that should be used in the 1 transmitter/multiple sensor mode. The command sequence to AutoConfig for 4 sensors would look like 0x50, followed by 0x32, followed by 0x04.

When PARAMETERnumber = 50, during an EXAMINE VALUE command, the Tracker returns 5 bytes of configuration information. Three pieces of information are passed, FBB CONFIGURATION MODE, FBB DEVICES, and FBB DEPENDENTS. FBB CONFIGURATION MODE, indicates the current

Tracker configuration as either Standalone or One Transmitter/Multiple Sensors mode. In the case of 3D Guidance medSAFE, the system is always in One Transmitter/Multiple Sensors mode. FBB DEVICES is used to tell which Trackers on the FBB are running. FBB DEPENDENTS informs the trackers that Slaves on the FBB will be using the signal transmitted from the current Master. In the case of 3D Guidance medSAFE, the first Tracker is always the Master and the other Trackers are always Slaves.

The bit definitions of the bytes are:

Byte 1	FBB Configuration Mode
0	STANDALONE
1	ONE TRANSMITTER/MULTIPLE SENSORS

Bytes 2, 3      FBB Devices

BIT 15	0
BIT 14	If 1, device at address 14 is running If 0, device at address 14 is not running
	A Tracker is RUNNING when the fly switch is on, it has been AUTO-CONFIGed and it is AWAKE. A device is not running when the fly switch is on and it has not been AUTO-CONFIGed or it has been AUTO-CONFIGed and it is ASLEEP.
BIT 13	If 1, device at address 13 is running If 0, device at address 13 is not running
.	.
.	.
BIT 1	If 1, device at address 1 is running If 0, device at address 1 is not running
BIT 0	0

## SENSOR OFFSET

When PARAMETERnumber = 71 during EXAMINE, the tracker will return 3 words (6 bytes) of data corresponding to the X, Y and Z offsets used in the OFFSET command.

See OFFSET command for an explanation.

To CHANGE the offsets send 6 bytes of PARAMETERdata after the 2 command bytes.

#### BOOT LOADER FIRMWARE REVISION

When PARAMETERnumber = 130 during EXAMINE, the tracker will return 2 bytes indicating the revision number of the Boot Loader firmware. E.g., if the first byte returned is 1 and the second byte is 2, the firmware revision number is 1.2.

#### MDSP FIRMWARE REVISION

When PARAMETERnumber = 131 during EXAMINE, the system will return 2 bytes indicating the revision number of the MDSP firmware. E.g., if the first byte returned is 1 and the second byte is 2, the firmware revision number is 1.2.

#### NON DIPOLE POSERVER FIRMWARE REVISION

When PARAMETERnumber = 133 during EXAMINE, the tracker will return 2 bytes indicating the revision number of the NonDipole POServer firmware. E.g., if the first byte returned is 1 and the second byte is 2, the firmware revision number is 1.2.

#### FIVE DOF FIRMWARE REVISION

When PARAMETERnumber = 135 during EXAMINE, the tracker will return 2 bytes indicating the revision number of the 5DOF firmware. E.g., if the first byte returned is 1 and the second byte is 2, the firmware revision number is 1.2.

#### SIX DOF FIRMWARE REVISION

When PARAMETERnumber = 136 during EXAMINE, the tracker will return 2 bytes indicating the revision number of the 6DOF firmware. E.g., if the first byte returned is 1 and the second byte is 2, the firmware revision number is 1.2.

#### DIPOLE POSERVER FIRMWARE REVISION

When PARAMETERnumber = 137 during EXAMINE, the system will return 2 bytes indicating the revision number of the Dipole POServer firmware. E.g., if the first byte returned is 1 and the second byte is 2, the firmware revision number is 1.2.

## HEMISPHERE

	ASCII	HEX	DECIMAL	BINARY
Command Byte	L	4C	76	01001100

Command Data	HEMI_AXIS	HEMI_SIGN
--------------	-----------	-----------

The shape of the magnetic field transmitted by the tracker is symmetrical about each of the axes of the transmitter. This symmetry leads to an ambiguity in determining the sensor's X, Y, Z position. The amplitudes will always be correct, but the signs ( $\pm$ ) may all be wrong, depending upon the hemisphere of operation. In many applications, this will not be relevant, but if you desire an unambiguous measure of position, operation must be either confined to a defined hemisphere or your host computer must 'track' the location of the sensor.

There is no ambiguity in the sensor's orientation angles as output by the ANGLES command, or in the rotation matrix as output by the MATRIX command.

The HEMISPHERE command is used to tell the tracker in which hemisphere, centered about the transmitter, the sensor will be operating. There are six hemispheres from which you may choose: the forward, aft (rear), upper, lower, left, and the right. If no HEMISPHERE command is issued, the forward is used by default.

The two Command Data bytes, sent immediately after the HEMISPHERE command, are to be selected from these:

Hemisphere	HEMI_AXIS		HEMI_SIGN	
ASCII	HEX	ASCII	HEX	
Forward	nul	00	nul	00
Aft (Rear)	nul	00	soh	01
Upper	ff	0C	soh	01
Lower	ff	0C	nul	00
Left	ack	06	soh	01
Right	ack	06	nul	00

The ambiguity in position determination can be eliminated if your host computer's software continuously 'tracks' the sensor location. In order to implement tracking, you must understand the behavior of the signs ( $\pm$ ) of the X, Y, and Z position outputs when the sensor crosses a hemisphere boundary. When you select a given hemisphere of operation, the sign on the position axes that defines the hemisphere direction is forced to positive, even when the sensor moves into another hemisphere. For example, the power-up default hemisphere is the forward hemisphere. This forces X position outputs to always be positive. The signs on Y and Z will vary between plus and minus depending on where you are within this hemisphere. If you had selected the lower hemisphere, the sign of Z would always be positive and the signs on X and Y will vary between plus and minus. If you had selected the left hemisphere, the sign of Y would always be negative, etc.

Regarding the default forward hemisphere, if the sensor moved into the aft hemisphere, the signs on Y and Z would instantaneously change to opposite polarities while the sign on X remained positive. To 'track' the sensor, your host software, on detecting this sign change, would reverse the signs on The Tracker's X, Y, and Z outputs. In order to 'track' correctly: You must start 'tracking' in the selected hemisphere so that the signs on the outputs are initially correct, and you must guard against the case where the sensor legally crossed the  $Y = 0$ ,  $Z = 0$  axes simultaneously without having crossed the  $X = 0$  axes into the other hemisphere.

## MATRIX

	ASCII	HEX	DECIMAL	BINARY
Command Byte	X	58	88	01011000

The MATRIX mode outputs the 9 elements of the rotation matrix that define the orientation of the sensor's X, Y, and Z axes with respect to the transmitter's X, Y, and Z axes. If you want a three-dimensional image to follow the rotation of the sensor, you must multiply your image coordinates by this output matrix.

The nine elements of the output matrix are defined generically by:

$M(1,1)$	$M(1,2)$	$M(1,3)$
$M(2,1)$	$M(2,2)$	$M(2,3)$
$M(3,1)$	$M(3,2)$	$M(3,3)$

Or in terms of the rotation angles about each axis  
where Z = Zang, Y = Yang and X = Xang:

$\cos(Y) * \cos(Z)$	$\cos(Y) * \sin(Z)$	$-\sin(Y)$
$-\cos(X) * \sin(Z)$	$\cos(X) * \cos(Z)$	
$+\sin(X) * \sin(Y) * \cos(Z)$	$+\sin(X) * \sin(Y) * \sin(Z)$	$\sin(X) * \cos(Y)$
$\sin(X) * \sin(Z)$	$-\sin(X) * \cos(Z)$	
$+\cos(X) * \sin(Y) * \cos(Z)$	$+\cos(X) * \sin(Y) * \sin(Z)$	$\cos(X) * \cos(Y)$

Or in Euler angle notation, where R = Roll, E = Elevation, A = Azimuth:

$\cos(E) * \cos(A)$	$\cos(E) * \sin(A)$	$-\sin(E)$
$-\cos(R) * \sin(A)$	$\cos(R) * \cos(A)$	
$+\sin(R) * \sin(E) * \cos(A)$	$+\sin(R) * \sin(E) * \sin(A)$	$\sin(R) * \cos(E)$
$\sin(R) * \sin(A)$	$-\sin(R) * \cos(A)$	
$+\cos(R) * \sin(E) * \cos(A)$	$+\cos(R) * \sin(E) * \sin(A)$	$\cos(R) * \cos(E)$

The output record is in the following format for the eighteen transmitted bytes:

MSB							LSB		
7	6	5	4	3	2	1	0	BYTE #	
1	M8	M7	M6	M5	M4	M3	M2	#1	LSbyte M(1,1)
0	M15	M14	M13	M12	M11	M10	M9	#2	MSbyte M(1,1)
0	M8	M7	M6	M5	M4	M3	M2	#3	LSbyte M(2,1)
0	M15	M14	M13	M12	M11	M10	M9	#4	MSbyte M(2,1)
0	M8	M7	M6	M5	M4	M3	M2	#5	LSbyte M(3,1)
0	M15	M14	M13	M12	M11	M10	M9	#6	MSbyte M(3,1)
0	M8	M7	M6	M5	M4	M3	M2	#7	LSbyte M(1,2)
0	M15	M14	M13	M12	M11	M10	M9	#8	MSbyte M(1,2)
0	M8	M7	M6	M5	M4	M3	M2	#9	LSbyte M(2,2)
0	M15	M14	M13	M12	M11	M10	M9	#10	MSbyte M(2,2)
0	M8	M7	M6	M5	M4	M3	M2	#11	LSbyte M(3,2)
0	M15	M14	M13	M12	M11	M10	M9	#12	MSbyte M(3,2)
0	M8	M7	M6	M5	M4	M3	M2	#13	LSbyte M(1,3)
0	M15	M14	M13	M12	M11	M10	M9	#14	MSbyte M(1,3)
0	M8	M7	M6	M5	M4	M3	M2	#15	LSbyte M(2,3)
0	M15	M14	M13	M12	M11	M10	M9	#16	MSbyte M(2,3)
0	M8	M7	M6	M5	M4	M3	M2	#17	LSbyte M(3,3)
0	M15	M14	M13	M12	M11	M10	M9	#18	MSbyte M(3,3)

The matrix elements take values between the binary equivalents of +.99996 and -1.0.  
Element scaling is +.99996 = 7FFF Hex, 0 = 0 Hex, and -1.0 = 8000 Hex.

## METAL

	ASCII	HEX	DECIMAL	BINARY
Command Byte	s	73	115	01110011

Command Data	METALflag	METALdata
--------------	-----------	-----------

When the METAL mode command is given, all subsequent tracker data requests will have a METAL error byte added to the end of the data stream. If the BUTTON byte is also being output, the BUTTON byte precedes the METAL byte. The METAL error byte is a number between 0 and 127 base 10 that indicates the degree to which the position and angle measurements are in error due to 'bad' metals located near the transmitter and sensor or due to tracker 'system' errors. 'Bad' metals are metals with high electrical conductivity such as aluminum, or high magnetic permeability such as steel. 'Good' metals have low conductivity and low permeability such as 300- series stainless steel, or titanium. The METAL error byte also reflects tracker 'system' errors resulting from accuracy degradations in the transmitter, sensor, or other electronic components. The METAL error byte also responds to accuracy degradation resulting from movement of the sensor or environmental noise. A METAL error byte = 0 indicates no or minimal position and angle errors depending on how sensitive you have set the error indicator. A METAL error byte = 127 indicates maximum error for the sensitivity level selected.

The metal detector is sensitive to the introduction of metals in an environment where no metals were initially present. This metal detector can fool you, however, if there are some metals initially present and you introduce new metals. It is possible for the new metal to cause a distortion in the magnetic field that reduces the existing distortion at the sensor. When this occurs you'll see the METALerror value initially decreases, indicating less error, and then finally start increasing again as the new metal causes more distortion. **Important Note: You need to evaluate your application for suitability of this metal detector.**

Because the tracker is used in many different applications and environments, the METAL error indicator needs to be sensitive to this broad range of environments. Some users may want the METAL error indicator to be sensitive to very small amounts of metal in the environment while other applications may only want the error indicator sensitive to large amounts of metal. To accommodate this range of detection sensitivity, the METAL command allows the user to set a Sensitivity that is appropriate to their application.

The METAL error byte will always show there is some error in the system even when there are no metals present. This error indication usually increases as the distance between the transmitter and sensor increases and is due to the fact tracker components cannot be made or calibrated perfectly. To minimize the amount of this inherent error in the METAL error value, a linear curve fit, defined by a slope and offset, is made to this inherent error and stored in each individual sensor's memory since the error depends primarily on the size of the sensor being used (25mm, 8mm, or 5 mm). The METAL command allows the user to eliminate or change these values. For example, maybe the user's standard environment has large errors and he or she wants to look at variations from this standard environment. To do this he or she would adjust the slope and offset to minimize the METAL error values.



On power up initialization of the system or whenever the user wants to change the METAL values the user must send to the tracker the following three byte sequence:

Command Byte    METALflag    METALdata

Where the Command Byte is the equivalent of an ASCII s (lower case) and the METALflag and METALdata are:

METALflag	METALdata	
0	0	Turn off metal detection.
1	0	Turn on metal detection using system default METALdata
2	Sensitivity	Turn on metal detection and change the Sensitivity
3	Offset	Turn on metal detection and change the Offset
4	Slope	Turn on metal detection and change the Slope
5	Alpha	Turn on metal detection and change the filter's alpha

METALflag=0. This is the default power up configuration. No METAL error byte is output at the end of the tracker's data stream. A zero value, zero decimal or zero hex or zero binary must be sent as the METALdata if you are turning off METAL detection.

METALflag=1. Turns on METAL detection using the system default sensitivity, offset, slope and alpha values. When METAL detection is turned on an additional byte is output at the end of the tracker's output data. If you have BUTTON MODE enabled then the METAL error value will be output after the BUTTON value byte is output.

METALflag=2. Turns on METAL detection and changes the sensitivity of the measurement to metals. The Offset, Slope and Alpha values are unchanged from their previous setting. The METALError value that is output is computed from:

$$\text{METALError} = \text{Sensitivity} \times (\text{METALErrorSYSTEM} - (\text{Offset} + \text{Slope} \times \text{Range}))$$
Where range is the distance between the transmitter and sensor. The user supplies a Sensitivity byte as an integer between 0 and 127 depending on how little or how much he or she wants METALError to reflect errors. The default value is 32.

METALflag=3. Turns on METAL detection and changes the Offset value defined in the equation above. The Offset byte value must be an integer value between plus or minus 127. If you are trying to minimize the base errors in the system by adjusting the Offset you could set the Sensitivity =1, and the Slope=0 and read the Offset directly as the METALError value.

METALflag=4. Turns on METAL detection and changes the Slope value defined in the equation above. The Slope byte value must be an integer between plus or minus 127. You can determine the slope by setting the Sensitivity=1 and looking at the change in the METALError value as you translate the sensor from range=0 to range max for the system, i.e. 36 inches for the tracker.. Since its difficult to go from range =0 to max, you might just translate over say half the distance and double the METALError value change you measure.

METALflag=5. Turns on METAL detection and changes the filter's Alpha value. The METALError value is filtered before output to the user to minimize noise jitter. The Alpha value determines how much filtering is applied to METALError. Alpha varies from 0 to 127. A zero value is an infinite amount of filtering, whereas a 127 value is no filtering. The system default is 12. As Alpha gets smaller the time lag between the insertion of metal in the environment and it being reported in the METALError value increases.

## OFFSET

	ASCII	HEX	DECIMAL	BINARY
Command Byte	K	4B	75	01001011
Command Data	X, Y, Z    OFFSET DISTANCES FROM SENSOR			

Normally the position outputs from the tracker represent the x, y, z position of the center of the sensor with respect to the origin of the transmitter. The OFFSET command allows the user to specify a location that is offset from the center of the sensor. The x, y, z offset distances you supply with this command are measured in the reference frame attached to the sensor and are measured from the sensor center to the desired position. After the command is executed, all subsequent positional outputs from the Tracker will be x, y, z desired.

With the command you send to the tracker three words of data, the Xoffset, Yoffset, and Zoffset coordinates. The scaling of these coordinates is the same as the POSITION command coordinates. For example, assume you were using a tracker in its default maximum range mode of 36 inches full scale. Also assume the Xoffset, Yoffset, and Zoffset values where 5.4 inches, - 2.1 inches, and 1.3 inches. You would then output three integer or their hex equivalents to the tracker equal to:

$Xoffset = 4915 = 5.4 * 32768 / 36.$   
 $Yoffset = 63625 = 65536 - 1911$   
 $Zoffset = 1183$

## POINT

	ASCII	HEX	DECIMAL	BINARY
Command Byte	B	42	66	01000010

In the POINT mode, the tracker sends one data record each time it receives the B Command Byte.

## POSITION

	ASCII	HEX	DECIMAL	BINARY
Command Byte	V	56	86	01010110

In the POSITION mode, the tracker outputs the X, Y, and Z positional coordinates of the sensor with respect to the transmitter. The output record is in the following format for the six transmitted bytes:

	MSB 7	6	5	4	3	2	1	LSB 0	BYTE #
1	X8	X7	X6	X5	X4	X3	X2	X1	#1 LSbyte X
0	X15	X14	X13	X12	X11	X10	X9	X8	#2 MSbyte X
0	Y8	Y7	Y6	Y5	Y4	Y3	Y2	Y1	#3 LSbyte Y
0	Y15	Y14	Y13	Y12	Y11	Y10	Y9	Y8	#4 MSbyte Y
0	Z8	Z7	Z6	Z5	Z4	Z3	Z2	Z1	#5 LSbyte Z
0	Z15	Z14	Z13	Z12	Z11	Z10	Z9	Z8	#6 MSbyte Z

The X, Y, and Z values vary between the binary equivalent of  $\pm$  MAX inches. Where MAX = 36" or 72". The default positive X, Y, and Z directions are shown in Default Reference Frames on page 24

Scaling of each position coordinate is full scale = MAX inches. That is, +MAX = 7FFF Hex, 0 = 0 Hex, -MAX = 8000 Hex. Since the maximum range (Range = square root( $X^2+Y^2+Z^2$ )) from the Transmitter to the sensor is limited to MAX inches, only one of the X, Y, or Z coordinates may reach its full scale value. Once a full scale value is reached, the positional coordinates no longer reflect the correct position of the sensor.

**To convert the numbers into inches** first cast it into a signed integer. This will give you a number from +/- 32767. Second multiply by 36 or 72. Finally divide the number by 32768 to get the position *in inches*. The equation should look something like this:

$$\begin{aligned} & (\text{signed int}(\text{Hex \#}) * 36) / 32768 \\ \text{Or: } & (\text{signed int}(\text{Hex \#}) * 72) / 32768 \end{aligned}$$

## POSITION/ANGLES

	ASCII	HEX	DECIMAL	BINARY
Command Byte	Y	59	89	01011001

In the POSITION/ANGLES mode, the outputs from the POSITION and ANGLES modes are combined into one record containing the following twelve bytes:

MSB							LSB		
7	6	5	4	3	2	1	0	BYTE #	
1	X8	X7	X6	X5	X4	X3	X2	#1	LSbyte X
0	X15	X14	X13	X12	X11	X10	X9	#2	MSbyte X
0	Y8	Y7	Y6	Y5	Y4	Y3	Y2	#3	LSbyte Y
0	Y15	Y14	Y13	Y12	Y11	Y10	Y9	#4	MSbyte Y
0	Z8	Z7	Z6	Z5	Z4	Z3	Z2	#5	LSbyte Z
0	Z15	Z14	Z13	Z12	Z11	Z10	Z9	#6	MSbyte Z
0	Z8	Z7	Z6	Z5	Z4	Z3	Z2	#7	LSbyte Zang
0	Z15	Z14	Z13	Z12	Z11	Z10	Z9	#8	MSbyte Zang
0	Y8	Y7	Y6	Y5	Y4	Y3	Y2	#9	LSbyte Yang
0	Y15	Y14	Y13	Y12	Y11	Y10	Y9	#10	MSbyte Yang
0	X8	X7	X6	X5	X4	X3	X2	#11	LSbyte Xang
0	X15	X14	X13	X12	X11	X10	X9	#12	MSbyte Xang

See POSITION mode and ANGLE mode for number ranges and scaling.

## POSITION/MATRIX

	ASCII	HEX	DECIMAL	BINARY
Command Byte	Z	5A	90	01011010

In the POSITION/MATRIX mode, the outputs from the POSITION and MATRIX modes are combined into one record containing the following twenty four bytes:

MSB							LSB		
7	6	5	4	3	2	1	0	BYTE #	
1	X8	X7	X6	X5	X4	X3	X2	#1	LSbyte X
0	X15	X14	X13	X12	X11	X10	X9	#2	MSbyte X
0	Y8	Y7	Y6	Y5	Y4	Y3	Y2	#3	LSbyte Y
0	Y15	Y14	Y13	Y12	Y11	Y10	Y9	#4	MSbyte Y
0	Z8	Z7	Z6	Z5	Z4	Z3	Z2	#5	LSbyte Z
0	Z15	Z14	Z13	Z12	Z11	Z10	Z9	#6	MSbyte Z
0	M8	M7	M6	M5	M4	M3	M2	#7	LSbyte M(1,1)
0	M15	M14	M13	M12	M11	M10	M9	#8	MSbyte M(1,1)
0	M8	M7	M6	M5	M4	M3	M2	#9	LSbyte M(2,1)
0	M15	M14	M13	M12	M11	M10	M9	#10	MSbyte M(2,1)
0	M8	M7	M6	M5	M4	M3	M2	#11	LSbyte M(3,1)
0	M15	M14	M13	M12	M11	M10	M9	#12	MSbyte M(3,1)
0	M8	M7	M6	M5	M4	M3	M2	#13	LSbyte M(1,2)
0	M15	M14	M13	M12	M11	M10	M9	#14	MSbyte M(1,2)
0	M8	M7	M6	M5	M4	M3	M2	#15	LSbyte M(2,2)
0	M15	M14	M13	M12	M11	M10	M9	#16	MSbyte M(2,2)
0	M8	M7	M6	M5	M4	M3	M2	#17	LSbyte M(3,2)
0	M15	M14	M13	M12	M11	M10	M9	#18	MSbyte M(3,2)
0	M8	M7	M6	M5	M4	M3	M2	#19	LSbyte M(1,3)
0	M15	M14	M13	M12	M11	M10	M9	#20	MSbyte M(1,3)
0	M8	M7	M6	M5	M4	M3	M2	#21	LSbyte M(2,3)
0	M15	M14	M13	M12	M11	M10	M9	#22	MSbyte M(2,3)
0	M8	M7	M6	M5	M4	M3	M2	#23	LSbyte M(3,3)
0	M15	M14	M13	M12	M11	M10	M9	#24	MSbyte M(3,3)

See POSITION mode and MATRIX mode for number ranges and scaling.

## POSITION/QUATERNION

	ASCII	HEX	DECIMAL	BINARY
Command Byte	J	5D	93	01011101

In the POSITION/QUATERNION mode, the tracker outputs the X, Y, and Z position and the four quaternion parameters,  $q_0$ ,  $q_1$ ,  $q_2$ , and  $q_3$  that describe the orientation of the sensor with respect to the Transmitter. The output record is in the following format for the fourteen transmitted bytes:

MSB	7	6	5	4	3	2	1	LSB	0	BYTE #	
1	X8	X7	X6	X5	X4	X3	X2			#1	Lsbyte X
0	X15	X14	X13	X12	X11	X10	X9			#2	MSbyte X
0	Y8	Y7	Y6	Y5	Y4	Y3	Y2			#3	Lsbyte Y
0	Y15	Y14	Y13	Y12	Y11	Y10	Y9			#4	MSbyte Y
0	Z8	Z7	Z6	Z5	Z4	Z3	Z2			#5	Lsbyte Z
0	Z15	Z14	Z13	Z12	Z11	Z10	Z9			#6	MSbyte Z
0	B8	B7	B6	B5	B4	B3	B2			#7	Lsbyte $q_0$
0	B15	B14	B13	B12	B11	B10	B9			#8	MSbyte $q_0$
0	B8	B7	B6	B5	B4	B3	B2			#9	Lsbyte $q_1$
0	B15	B14	B13	B12	B11	B10	B9			#10	MSbyte $q_1$
0	B8	B7	B6	B5	B4	B3	B2			#11	Lsbyte $q_2$
0	B15	B14	B13	B12	B11	B10	B9			#12	MSbyte $q_2$
0	B8	B7	B6	B5	B4	B3	B2			#13	Lsbyte $q_3$
0	B15	B14	B13	B12	B11	B10	B9			#14	MSbyte $q_3$

See POSITION mode and QUATERNION mode for number ranges and scaling.

## QUATERNION

	ASCII	HEX	DECIMAL	BINARY
Command Byte	\	5C	92	01011100

In the QUATERNION mode, the tracker outputs the four quaternion parameters that describe the orientation of the sensor with respect to the transmitter. The quaternions,  $q_0$ ,  $q_1$ ,  $q_2$ , and  $q_3$  where  $q_0$  is the scalar component, have been extracted from the MATRIX output using the algorithm described in "Quaternion from Rotation Matrix" by Stanley W. Shepperd, *Journal of Guidance and Control*, Vol. 1, May-June 1978, pp. 223-4. The output record is in the following format for the eight transmitted bytes:

MSB							LSB	
7	6	5	4	3	2	1	0	BYTE #
1	B8	B7	B6	B5	B4	B3	B2	#1 Lsbyte $q_0$
0	B15	B14	B13	B12	B11	B10	B9	#2 MSbyte $q_0$
0	B8	B7	B6	B5	B4	B3	B2	#3 Lsbyte $q_1$
0	B15	B14	B13	B12	B11	B10	B9	#4 MSbyte $q_1$
0	B8	B7	B6	B5	B4	B3	B2	#5 Lsbyte $q_2$
0	B15	B14	B13	B12	B11	B10	B9	#6 MSbyte $q_2$
0	B8	B7	B6	B5	B4	B3	B2	#7 Lsbyte $q_3$
0	B15	B14	B13	B12	B11	B10	B9	#8 MSbyte $q_3$

Scaling of the quaternions is full scale = +.99996 = 7FFF Hex, 0 = 0 Hex, and -1.0 = 8000 Hex.



## READ\_VPD

	ASCII	HEX	DECIMAL	BINARY
Command Byte	o	6F	111	01101111

The READ\_VPD command allows the user to read from the 128-byte Vital Product Data (VPD) sections in the board, transmitter, sensor or preamp.

The command sequence consists of the command byte followed by a high-order address byte, a low-order address byte and an EEPROM-selector byte. The two-byte address is intended to support possible future expansion of the VPD area. The EEPROM-selector byte can be one of four values:

Selector:	EEPROM:
0	Board
1	Sensor
2	Transmitter
3	Preamplifier

Upon receiving the READ\_VPD command, the tracker will output the single byte of data read from the selected VPD section

Reading VPD data is not allowed while the transmitter is running. The tracker will return a 0 byte.

## REFERENCE FRAME

	ASCII	HEX	DECIMAL	BINARY
Command Byte	r	72	114	01110010
Command Data	A, E, R			

By default, the tracker's reference frame is defined by the transmitter's physical X, Y, and Z axes (see [Default Reference Frames](#)). In some applications, it may be desirable to have the orientation measured with respect to another reference frame. The REFERENCE FRAME command permits you to define a new reference frame by inputting the angles required to align the physical axes of the Transmitter to the X, Y, and Z axes of the new reference frame. The alignment angles are defined as rotations about the Z, Y, and X axes of the transmitter. These angles are called the, Azimuth, Elevation, and Roll angles.

The command sequence consists of a Command Byte and 6 Command Data bytes. The Command Data consists of the alignment angles Azimuth (A), Elevation (E), and Roll (R).

If you immediately follow the REFERENCE FRAME command with a POINT or STREAM mode data request you may not see the effect of this command in the data returned. It may take one measurement period before you see the effect of the command.

The Command Byte and Command Data must be transmitted to the tracker in the following seven-byte format:

MSB				LSB				BYTE #
7	6	5	4	3	2	1	0	
0	1	1	1	0	0	1	0	#1 Command Byte
B7	B6	B5	B4	B3	B2	B1	B0	#2 Lsbyte A
B15	B14	B13	B12	B11	B10	B9	B8	#3 MSbyte A
B7	B6	B5	B4	B3	B2	B1	B0	#4 Lsbyte E
B15	B14	B13	B12	B11	B10	B9	B8	#5 MSbyte E
B7	B6	B5	B4	B3	B2	B1	B0	#6 Lsbyte R
B15	B14	B13	B12	B11	B10	B9	B8	#7 MSbyte R

See the ANGLES command for the format and scaling of the angle values sent.

## REPORT RATE

Measurement Rate Divisor Command	ASCII	HEX	DECIMAL	BINARY
1	Q	51	81	01010001
2	R	52	82	01010010
8	S	53	83	01010011
32	T	54	84	01010100

If you do not want a tracker data record output to your host computer every measurement cycle when in STREAM mode then use the REPORT RATE command to change the output rate to every other cycle (R), every eight cycles (S) or every thirty-two cycles (T). If no REPORT RATE command is issued, transmission proceeds at the default measurement rate.

**Note:**

For alternate Report Rate settings, use the CHANGE/EXAMINE command for this function. See [REPORT RATE](#)

## RESET

	ASCII	HEX	DECIMAL	BINARY
Command Byte	b	62	98	01100010

The RESET command is issued to the tracker to restart the system. RESET does reinitialize the system from the flash memory, so any configuration or alignment data entered before the system was reset, will revert back to power-up settings stored in the Flash.

**Note:**

Command not implemented at time of manual writing.

## RS232 TO FBB

	ASCII	HEX	DECIMAL	BINARY	
Command Byte	≡	F0	240	11110000	+ FBB (SENSOR) ADDR

The RS232 TO FBB pass through command is a command that was developed for first generation single sensor products, to allow the host computer to communicate with any specified sensors via a single RS232 interface. However, this command is equally relevant for the 3DGuidance systems, which support multiple sensors. The pass through command permits selection and configuration of sensors beyond the first sensor.

The command is a preface to each of the RS232 commands. This command is 1 Byte long:

Command Byte = 0xF0 + destination sensor address (in Hex)  
 i.e. Sensor address 1 (0x1 hex) would be: 0xF1  
 Sensor address 4 (0x4 hex) would be: 0xF4

Example 1: There are two Tracked sensors connected to the system. One at Address 1 and the other at Address 2. (By default the Tracked sensor connected to the first port is at Address 1 and the Tracked sensor connected to the second port is at Address 2)

To get Position/Angle data **from Sensor 1**, the host would either send:

- A 2 byte command consisting of:
  - The RS232 TO FBB command, (0xF1 hex)
  - Followed by the POINT command (0x42 hex)
- Or the 1 byte:
  - POINT command (0x42hex)

To get Position/Angle data **from Sensor 2**, the host would send:

- A 2 byte command consisting of:
  - The RS232 TO FBB command, (0xF2 hex)
  - Followed by the POINT command (0x42 hex)

Notes:

1) To use STREAM mode with multiple sensors, first send the GROUP MODE command, followed by STREAM command.

**Note:**

No jumpers or FBB cables required for multi-sensor operation in the 3D Guidance systems.

## RUN

	ASCII	HEX	DECIMAL	BINARY
Command Byte	F	46	70	01000110

The RUN command is issued to the tracker to restart normal system operation after it has been put to sleep with the SLEEP command. RUN does not reinitialize the system RAM memory, so any configuration or alignment data entered before the system went to SLEEP will be retained.

## SLEEP

	ASCII	HEX	DECIMAL	BINARY
Command Byte	G	47	71	01000111

The SLEEP command turns the transmitter off, and halts the system. While asleep, the tracker will respond to data requests and mode changes but the data output will not change. To resume normal system operation, issue the RUN command.

**Tip:**

To maximize the life of your system, issue the SLEEP command when you are not using the tracker, or configure the [SleepOnReset](#) setting in the Configuration Utility.

## STREAM

	ASCII	HEX	DECIMAL	BINARY
Command Byte	@	40	64	01000000

In the STREAM mode, the tracker starts sending continuous data records to the host computer as soon as new data is available - at selected measurement rate. Data records will continue to be sent until the host sends the STREAM STOP command or the POINT command, or any format command such as POSITION to stop the stream.

Some computers and/or high-level software languages may not be able to keep up with the constant STREAM of data in this mode. Bytes received by your RS232 port may overrun one another or your input buffer may overflow if tracker data is not retrieved fast enough. This condition will cause lost bytes, hence if your high-level application software requests say 12 bytes from the RS232 input buffer, it may hang because one or more bytes were lost. To eliminate this possibility, read one byte at a time looking for the phasing bit that marks the first byte of the data record.

See [REPORT RATE](#) to change the rate at which records are transmitted during STREAM.



## STREAM STOP

	ASCII	HEX	DECIMAL	BINARY
Command Byte	?	3F	63	00111111

STREAM STOP turns STREAM mode off, stopping any data that was STREAMing from the tracker.

This is an alternative to stopping the stream using a POINT command. NOTE: The record in progress when the tracker receives the command will still be output in its entirety to the host computer. To ensure that you have cleared your input port before executing any new commands, send STREAM STOP, delay and then discard any data in your serial port buffer.

## WRITE\_VPD

	ASCII	HEX	DECIMAL	BINARY
Command Byte	p	70	112	01110000

The WRITE\_VPD command allows the user to write to the 128-byte Vital Product Data (VPD) sections in the board, transmitter, sensor or preamp.

The command sequence consists of the command byte followed by a high-order address byte, a low-order address byte, an EEPROM-selector byte and the data byte to be written. The two-byte address is intended to support possible future expansion of the VPD area. The EEPROM-selector byte can be one of four values:

Selector:	EEPROM:
0	Board
1	Sensor
2	Transmitter
3	Preamplifier

After writing the data to the selected VPD section, the WRITE\_VPD command rereads the same byte from EEPROM and outputs it to the user.

Reading VPD data is not allowed while the transmitter is running. The tracker will return a 0 byte.

## Error Reporting

3Dguidance medSAFE continuously monitors system activities and reports particular conditions to you through error codes. These codes may be generated as a result of the power-up diagnostics, or from regular error detection during normal operation. All error codes are 1 byte in length, and are reported to the SYSTEM ERROR buffer. This buffer holds up to 16 bytes of error information (16 codes).

### Notification

The user can choose to be notified that an error has been generated through any of the following methods:

#### *Error Flag*

Monitor the ERROR bit (**B13**) in the **two byte** BIRD STATUS register.

1. Send EXAMINE VALUE command with PARAMETERnumber = 0

When an error is detected this bit is set to a '1', and the generated error code is sent to the SYSTEM ERROR buffer.

To retrieve the code after the flag has been set:

2. Send the EXAMINE VALUE command with PARAMETERnumber = 10

This returns the earliest Error sent to the buffer and clears it.

Note: If there is only one error in the buffer, reading the ERROR bit (B13) in the BIRD STATUS word, will reset the bit to '0' indicating all errors have been read and cleared from the buffer. If the bit remains a '1', then additional errors remain and should be read.

#### **SYSTEM ERROR**

Alternatively, you can query the SYSTEM ERROR buffer directly.

1. Send the EXAMINE VALUE command with PARAMETERnumber = 10

This returns the earliest Error sent to the buffer. and clears it.

2. Additional queries will return and clear the next Error code in the buffer, until the buffer. is empty. When the buffer is empty, the query will return '0'.

Note: If the query returns '45', then the 16-byte(16 error codes) buffer has overflowed, and the newest codes generated will be lost.

## Error Code Listing

<u>CODE</u>	<u>ERROR DESCRIPTION</u>
0	No Error Cause: SYSTEM ERROR register empty
3	Electronic Unit Configuration Data Corrupt Cause: The system was not able to read the Electronics unit's configuration data Action: Reset the system
5	Component Configuration Data Corrupt Cause: The system was not able to read the component EEPROM configuration data, or the components are not plugged in. Action: Insure that the components are present, calibrate the components
6	Invalid RS232 Command Cause: The system has received an invalid RS232 command, which can occur if the user sends down a command character that is not defined or if the data for a command does not make sense (i.e., change value commands with an unknown parameter number). Action: Only send valid RS232 commands to the tracker..
11	RS232 Receive Overrun or Framing Error Cause: An overrun or framing error has been detected by the serial channel UART as it received characters from the user's host computer on the RS232 interface. Action: If an overrun error, the baud rate of your host computer and the tracker differ. This may be due to incorrect baud selection, inaccuracy of the baud rate generator, or the RS232 cable is too long for the selected baud rate. If a framing error, the host software may be sending characters to its own UART before the UART finishes outputting the previous character.
18	Illegal Baud Rate Error Cause: If the baud rate setting is in an 'invalid' baud rate setting then this error will occur. Action: Set up baud rate with a valid setting.
37	DSP POST error Cause: Can't download code to the acquisition DSP(s). Action: Contact Ascension
44	Algorithm Overflow Cause: Computational error, possibly due to noise in the environment Action: Check environment, and sensor data using utility
45	Error Buffer Overflow Cause: Too many errors detected and sent to the SYSTEM ERROR register. Not all errors will be reported Action: Read register to clear errors

- 46      Flash Checksum error  
Cause: Section of the Electronics Unit's Flash memory corrupted.  
Action: Reload the Flash using the utility, but not more than 5000 times.

# 7

## Chapter 7: Troubleshooting

Most installation and tracking problems are easy to fix. Consult our troubleshooting table for common problems and their solutions. If you continue to experience problems, contact us for technical support.

Symptom	Possible Causes	Solution
No front panel LED illumination	No power	-Check AC connections to power supply -Reset hardware by cycling AC power
Demo Utility doesn't run	No serial communication  Software installation unsuccessful	-Check the suggestions outlined in 'Not able to communicate' below  -Re-initialize the HOST PC and run the installation again
Power-up defaults did not change after configuring with the utility	Configuration download interrupted  Tracker did not reset(restart) after burning the Flash settings	-Re-start the tracker and the utility and set the defaults again. Be sure to click APPLY to send the settings to the Electronics  -Cycle power to the tracker, and re-check the settings
Not able to communicate with the system using USB	Re-initialize USB  Driver not installed	-Unplug and replug USB cable on tracker. -Cycle power on the tracker  -Check installation/status of tracker USB driver in Windows Device Manager. Re-install if necessary

Symptom	Possible Causes	Solution
Can communicate with the system, but data not changing	<p>Sensor is saturated</p> <p>Transmitter is OFF or disconnected</p> <p>Component connections faulty</p>	<p>-Check the error codes for a sensor a saturation condition. Move the sensor farther away from the Transmitter.</p> <p>-Check status of the Transmitter. Turn ON using <a href="#">SELECT_TRANSMITTER</a> parameter (3D Guidance API) or <a href="#">RUN</a> command (Flock protocol).</p> <p>-Check that Sensor/Transmitter connectors are correctly installed. Inspect pins for wear or damage.</p> <p>-Contact Ascension for assistance</p>
Data is too noisy	<p>Filters OFF</p> <p>Low Signal</p> <p>External noise in environment</p> <p>Changing Hemisphere</p> <p>Line frequency value set incorrectly.</p>	<p>-Check FILTER STATUS for present state of filter configuration.</p> <p>-Decrease distance from sensor to Transmitter.</p> <p>-Be sure that the sensor is not located near the Tracker's power supply or other electronic devices or cables. See section on <a href="#">Reducing Noise</a> in Chapter 3</p> <p>-If the signs of the X, Y or Z position outputs suddenly change you may have crossed a hemisphere boundary. Use the HEMISPHERE command to rectify.</p> <p>-Determine correct frequency (50 Hz in Europe, 60 Hz in North America) and set to correct value.</p>
Poor accuracy	<p>Metal in tracking environment.</p> <p>Damaged equipment.</p> <p>Sensor or transmitter connector not properly inserted.</p> <p>A software application error.</p>	<p>-Check all around the transmitter to the furthest distance from the center of the transmitter to the maximum distance the sensor is used. Move or replace metal, or reposition the tracker system.</p> <p>-Check for damage.</p> <p>-Correct by unplugging and plugging the components back into the board.</p> <p>-Verify formulas and scale factors.</p>

## Error Codes

The 3D Guidance API Reference provides a complete listing of all [3D Guidance API error codes](#) via the USB interface.

# 8

## Chapter 8: Maintenance, Repair and Disposal

Taking care of your tracker is simple and straightforward. For years of accurate operation, be sure to treat the components as delicate electronic components.

The parts of the tracker that are physically handled are subject to the most wear. With proper handling and care the electronics unit, sensor and transmitter should indefinitely -- well beyond our warranty period.

### User Maintenance

The tracker requires minimal maintenance. You should do the following to maintain good performance:

#### Maintenance Prior to Each Use

1. Check the transmitter and sensor cables for nicks and cuts in the insulation. If nicks or cuts are found, the component should be replaced after proper disposal.
2. Inspect component connectors and receptacles for bent or damaged pins or other obstructions.
3. Inspect the transmitter for cracks or exterior damage. If transmitter is cracked or interior of the transmitter is exposed in some way, the component should be replaced after proper disposal.

#### Periodic Maintenance (As needed)

1. Inspect USB and power connections to ensure positive contact.



2. Transmitter and sensors are properly mounted as recommended in [Mounting the Hardware](#).

### Cleaning and Disinfecting

Periodically, clean the equipment (electronics unit, transmitter, sensor, and cables) by wiping down with a cloth dampened in a cleaning solution such as mild soap and water, isopropyl alcohol or a similar acceptable cleaning solution. If the tracker's components come in contact with biological fluid or tissue, be sure to follow your organization's procedures for proper cleaning and disinfection. The electronics unit, transmitters and sensors are not designed to withstand autoclaving or gamma radiation. Sensors are ETO compatible. Do not immerse the electronics unit, transmitter, sensor, or cables in liquids. Components are not waterproof.

### Sensor Sterilization

3D Guidance medSAFE sensor materials are tolerant of both cold sterilant (Cidex or equivalent) and Ethylene Oxide (EtO) gas sterilization processes.

However, even if embedded in a medical instrument such as an endoscope or other non-disposable tool, the electronics portion of the sensors should never be subject to autoclaving or gamma radiation. The electronics portion of the sensors resides within the connector.

### BROAD GUIDELINES WHEN CONSIDERING THE CIDEX (GLUTARALDEHYDE) PROCESS

Warning: Never use this sterilization process without first consulting the manufacturer's instructions for proper and safe use. Ascension cannot determine appropriate minimum dosages since medSAFE components are always part of a larger medical device. Degree of sterilization is a function of the type of procedure undertaken and the device manufacturers' specifications. In all cases, institutional protocols should be strictly followed.

When considering the use of Cidex, you should:

- Use Cidex classified as "sterilant." A Cidex products classified as "disinfectants" are not adequate.
- Take into account the physical properties of the medical instrument being sterilized: It must be clean, relatively smooth, impervious to moisture, and be of a shape that permits all surfaces to be exposed to the sterilant.
- Ensure the medical instrument receives adequate exposure. All surfaces, both interior and exterior, should be exposed to the sterilant. Tubing must be completely filled and the

materials to be sterilized must be clean and arranged in the sterilant to assure total immersion.

- Use fresh solutions. The sterilant solution should be clean and fresh. Most sterilants come in solutions consisting of two parts to form an "activated" solution. The shelf life of activated solutions is indicated in the instructions for commercial products. Generally, this is from one to four weeks.
- Rinse chemically sterilized items. Instruments, implants, and tubing (both inside and out) must be rinsed with sterile saline or sterile water prior to use to avoid tissue damage.

#### BROAD GUIDELINES WHEN CONSIDERING THE ETO STERILIZATION PROCESS

Warning: Never use this sterilization process without first consulting the manufacturer's instructions for proper and safe use. Ascension cannot determine appropriate minimum dosages since medSAFE components are always part of a larger medical device that is sterilized. Degree of sterilization is a function of the type of procedures undertaken and the device manufacturers' specifications. In all cases, institutional protocols should be strictly followed.

EtO is a long-established and widely used hospital method of sterilization. Its low - temperature environment is compatible with electronic devices, such as the medSAFE sensor assembly. Gas sterilization with ethylene oxide requires the use of an approved gas sterilizer and appropriate monitoring systems to assure sterility and personnel safety. Ethylene gas is irritating to tissue; all materials require appropriate airing time.

Cycle protocols should be implemented in accordance with EN 550 /ISO 11135, "Sterilization of Healthcare Products." This document describes Ethylene oxide and requirements for development, validation, and routine control of a sterilization process for medical devices.

### Software and Firmware Updates

As new features or updates become available for the tracker, you may find it necessary to update the firmware stored in the electronic unit's memory. The *Medical Utility* included on your CD-ROM allows you to do this without opening the electronics or returning it to Ascension. Instructions for this procedure are included with updates.

## Repair

There are no user level repairs that can be made on the electronics unit, transmitters, or sensors. If you have a problem with any part of your tracker, please contact Ascension Technical Support.

Technical Support contact points are as follows:

**World Wide Web:** <http://www.ascension-tech.com/technical/>

**E-mail:** [support@ascension-tech.com](mailto:support@ascension-tech.com)

**Telephone:** Call (802) 893-6657, 9 AM -- 5 PM U.S. Eastern Standard Time, Monday through Friday.

**Fax:** (802) 893-6659.

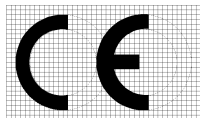
## Warranty

Ascension warrants that its products are free from defects in material and workmanship for a period of one (1) year from date of delivery, providing they are not subject to misuse, neglect, accident, incorrect installation, or improper care. If any Ascension products fail due to no fault of the buyer, Ascension will (at its option) either repair the defective product and restore it to normal operation without charge for parts and labor or provide a replacement in exchange for the defective product. Repair work shall be warranted for the remainder of the unexpired warranty period or for a period of 60 days whichever is longer. This warranty is the exclusive warranty given in lieu of any other express or implied warranty. Ascension disclaims any implied warranties of merchantability and fitness to a particular purpose. Warranties are voided if the buyer utilizes a power supply that does not strictly adhere to Ascension's electrical power requirements, changes the configuration of a tracker, (such as, adding extensions to cables or modifying boards), or mishandles sensors or cables. To avoid warranty issues, customers should carefully adhere to all product advisories. Transmitter and sensor cables and connectors are sensitive electronic components and should be treated with care. Do not drop, pull, twist, or mishandle cables.

## Disposal

The European Union has issued a directive, known as the WEEE (Waste Electrical and Electronic Equipment) Directive, to protect the quality of our environment by reducing the amount of electrical equipment waste buried in landfills. WEEE focuses on the recycling and reuse of "equipment that depends on an electronic current or an electromagnetic field to operate and as equipment for the generation, transfer and measurement of such currents and fields." Although none of the tracker's components are hazardous materials, proper disposal is important, especially, in the European Union where these components cannot be consigned to a landfill. Wherever available, tracker components should be brought to centralized recycling and collection points. Please contact Ascension Technical Support for further instructions on the correct disposal procedures in your country. If you have biologically contaminated components, please refer to your organizational procedures for disposing of biologically contaminated material.

## Chapter 9: Regulatory Information and Specifications



In accordance with EN60601-1 (Medical electrical equipment – general requirements for safety), this equipment is classified as follows:

Class I

Type CF Applied Part /Defib Proof

Not AP/APG

**Class I:** Non-invasive electric/electronic equipment without a monitoring function, which has a reliable ground and thus provides the type of protection against electric shock as defined by EN60601-1.



**Type CF Applied Part:** An applied part (sensors) complying with the specified requirements of EN 60601-1 to provide protection against electric shock, particularly regarding allowable leakage current. Type CF specifies the degree of electric shock protection provided by the unit.

**Defib Proof** signifies the sensor's capability to withstand the energy discharged from a defibrillator, as specified in the waveform detailed in IEC 60601-1, Amendment 2, Clause 17h

Note that the Defibrillation-Proof Type CF symbol (IEC symbol 60417-5336) is marked on the front panel of the electronics unit, next to the sensor input ports.

**Not AP/APG** means unsuitable for use in the presence of flammable gases.

Modification or use of the equipment in any way that is not specified by Ascension Technology Corporation may impair the protection and accuracy provided by the equipment.



The lightning flash arrow symbol within an equilateral triangle is intended to alert the user to the presence of uninsulated dangerous voltage within the product's enclosure. That voltage may constitute a risk of electric shock to persons.



The exclamation point within an equilateral triangle is intended to alert the user to the presence of important operating and maintenance (servicing) instructions in the appliance literature.



**Warning: This tracker does not have approval from the FDA for patient contact applications**

**APPLIED PART CAN COME IN CONTACT WITH PATIENT!!! PROVIDED ITS USER COMPLIES WITH ALL PERTINENT FDA/CE/IRB REQUIREMENTS.**

## EC Declaration of Conformity

Issued by

ASCENSION TECHNOLOGY CORPORATION  
PO Box 527  
Burlington, VT 05402 USA  
802-893-6657

**Equipment Description:** 3D Guidance medSAFE  
(Applies to tracking system utilizing ATC electronics unit PCB and  
ATC electronics unit enclosure.)

**Applicable Directive:** 93/42/EEC, Medical Devices

**Applicable Standards:** IEC 60601-1 Ed. 2 1997  
Medical Electrical Equipment: Part 1: General  
Requirements for Safety  
  
IEC 60601-1-2 Ed. 2 2001  
Medical Electrical Equipment: Part 1: General Requirements  
for Safety –2. Collateral Standard: Electromagnetic Compatibility-  
Requirements and Test

**Authorized by:**

**Date:**

Ernie Blood  
President/Chief Technology Officer  
Ascension Technology Corporation

## FCC Compliance Statement

### Radio and television interference

**Warning:** Changes or modifications to this unit not expressly approved by the party responsible for compliance could void the user's authority to operate the equipment.

**Note:** This equipment has been tested and found to comply with the limits for a Class B digital device, pursuant to Part 15 of the FCC rules. These limits are designed to provide reasonable protection against harmful interference in a residential installation. This equipment generates, uses, and can radiate radio frequency energy and, if not installed and used in accordance with the instructions, may cause harmful interference to radio communications. However, there is no guarantee that interference will not occur in a particular installation. If this equipment does cause harmful interference to radio or television reception, which can be determined by turning the equipment off and on, the user is encouraged to try and correct the interference by one or more of the following measures:

- Reorient or relocate the receiving antenna.
- Increase the separation between the equipment and receiver.
- Connect the equipment into an outlet on a circuit different from that to which the receiver is connected.
- Consult the dealer or an experienced radio/TV technician for help.

#### Declaration of Conformity

**Model Number:** 3D Guidance medSAFE  
**Trade Name:** Ascension  
**Responsible Party:** Ascension Technology Corporation  
**Address:** P.O. Box 527  
 Burlington, Vermont 05402  
**Telephone Number:** (802) 893-6657

This device complies with Part 15 of the FCC rules.

Operation is subject to the following two conditions:

1. This device may not cause harmful interference, and
2. This device must accept any interference received, including interference that may cause undesired operation.

## Product Specifications

### Performance

<b>Degrees of freedom:</b>	<b>Short-, Mid-Range and 4-Axis-Flat Transmitter:</b> Six (position and orientation)
<b>Translation range:</b>	<b>Short-Range Transmitter:</b> ±46 cm in any direction  <b>Mid-Range Transmitter:</b> ±76 cm in any direction  <b>Flat Transmitter:</b> +46 cm in Z direction
<b>Angular range:</b>	All attitude: ±180 deg azimuth and roll, ±90 deg elevation
<b>Static accuracy: (see note below)</b>	1.4mm RMS position 0.5 degree RMS orientation
<b>Update rate:</b>	<b>Short- and Mid-Range Transmitter:</b> Up to 375 updates/second. Calibrated at 80.0Hz measurement rate (240 updates/second).  <b>4-Axis-Flat Transmitter:</b> Up to 200 updates per second Calibrated at 40.5 Hz measurement rate (162 updates/second).  <b>9-Axis-Flat Transmitter:</b> Up to 198 updates per second Calibrated at 22 Hz measurement rate (198 updates/second).
<b>Position:</b>	Because of symmetries in the transmitted field for short and mid-range transmitters, operation of the sensor with these transmitters shall be confined to one of six hemispheres of operation, selectable by the user at runtime. In order to meet accuracy specifications, the system must operate in the forward (positive X) hemisphere.
<b>Outputs:</b>	X, Y, Z positional coordinates, orientation angles, orientation matrix and quaternion.
<b>Interface:</b>	USB / RS232

## Physical

### Size:

#### **Mid-Range Transmitter:**

3.75" (9.6cm) cube with 10' (3.3m) cable

#### **Short-Range Transmitter:**

2.47" (6.27cm) x 1.81" (4.6cm) x 2.05" (5.2cm)

#### **4-Axis Flat Transmitter:**

22" (56cm) x 22" (56cm) x 1.1" (2.8cm)

#### **9-Axis Flat Transmitter:**

22" (56cm) x 22" (56cm) x 1.1" (2.8cm)

#### **Model 800 Sensor:**

Sensor max OD 8.0mm

Sensor max length 20mm

Cable max OD 3.8mm

Cable length: 2 meters

#### **Model 180 Sensor:**

Sensor max OD 2.0mm

Sensor max length 9.9mm

Cable max OD 1.2mm

Cable length: 2 meters

#### **Model 130 Sensor:**

Sensor max OD 1.5mm

Sensor max length 7.6mm

Cable max OD 1.2mm

Cable length: 2 meters

#### **Model 90 Sensor:**

Sensor max OD 0.87mm

Sensor max length 8.8mm

Cable OD 0.6mm

Cable length: 2 meters

#### **180,130,and 90 Sensor Materials:**

Ascension Medi-Mag cable,

USP class 6 cable jacket material.

USP class 6 sensor housing material..

USP class 6 epoxy potting in tip.

Sensor assembly and cable materials are EtO and cold sterilant

tolerant. Do not use Gamma Radiation or autoclaving on sensor

assemblies. Semiconductor devices in the connector are not gamma

shielded and may be damaged or erased if exposed to gamma radiation

and/or heat of autoclaving.

Connectors are not sealed and must not be subjected to immersion in

liquids of any type.

Do not subject cable to an axial pull greater than 200 grams.



**Pre-amp unit**

Module max OD (minus threaded cap): 23.5mm  
Module max length (including sensor conn.) 163mm.  
Module length (excluding sensor connector) 144mm  
Cable max OD 5mm  
Cable length 3 meters

NOTE: Current pre-amplifiers are designed with consumer grade materials and construction. Specifically, they are not designed to be resistant to sterilization methods of any kind and are IPA wipe down only. Immersion in liquids will render them inoperable.

**Electronics Unit**

Dimensions (L x W x H): 27.1cm x 28.1cm x 7.0cm  
Weight: 2.06 Kg

<b>Power:</b>	The unit's internal supplies will operate from 100 to 240V at 50/60 Hz. Power consumption is 60 VA.
<b>Operating temperature:</b>	59°F to 95°F (15°C to 35°C), 90% non-condensing humidity.
<b>Warm up:</b>	System shall meet accuracy specifications after 5 mins.
<b>Note on static accuracy:</b>	Accuracy is defined as the RMS position error of the magnetic center of a single sensor with respect to the magnetic center of a single transmitter over the Performance Motion Box. Accuracy will be degraded if there are interfering electromagnetic noise sources or metal in the operating environment.

# Appendix I: medSAFE Utility

## Running medSAFE Utility

### Setup

Before changing settings or beginning an upgrade, setup the tracker with either the RS232 or USB interface.

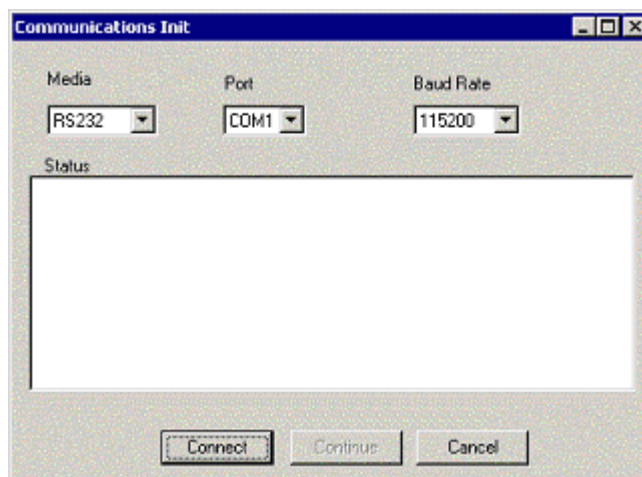
1. Connect the RS232 or USB cable and the Power cable to the rear panel of the tracker.
2. Connect the other end of the RS232 or USB cable to an open COM port or USB port on the host PC (PC that will run the utility)
3. Power on the Tracker.

### Run the Utility

4. Start the utility by running the 'medSAFEUtility.exe' file.  
This will open the interface configuration window of the Utility.
5. If using the RS232 interface, select the COM port from the pull down menu, and click Connect.

**Note:**

The Utility can be run using either the RS232 (COM port capable of 115.2Kbaud) or USB interfaces.

**Note:**

Current operation supports 115200 baud only.

If using the USB interface, select 'USB' from the 'Media' pull down menu and click Connect.

This will establish communication with the tracker, and initiate a reading of the current configuration. Progression of the reading is shown in the Status window.

**medSAFEUtility- (Rev. 2.0)**

NonDipole Settings | Flash Maintenance | Product Info | Data

**System Settings**

IP Address: 192.168.200.51 Mask: 255.255.0.0 Port: 6000

Baud Rate: 115200

Measurement Rate: 40.5

Hemisphere: FRONT

Scale: 36.0

Data Format: POS/AN

Report Rate: 1

Line Freq: 60

Sensor Offsets (inches): X: 0, Y: 0, Z: 0

Angle Align (degrees): Az: 0, El: 0, Rl: 0

Reference Frame (degrees): 0, 0, 0

☒ Sleep on reset

**Filters**

☒ AC Wide ON Alpha: 0.02, 0.02, 0.02, 0.02, 0.02, 0.02, 0.02, 0.02, 0.02, 0.02, 0.02, 0.02

☐ AC Narrow ON Alpha: 0.9, 0.9, 0.9, 0.9, 0.9, 0.9, 0.9, 0.9, 0.9, 0.9, 0.9, 0.9

☐ Adaptive filter VM: 60, 50, 40, 40, 2, 2, 2, 2, 2, 2, 2, 2

☐ Noise Reduction

**Kalman Filter**

Position Process Noise Velocity Model: .1

Position Process Noise Static: .1

Noise Filter Parameter: 0.5

Gain Q Static: 1e-005

Kalman Model: Both

Starting X (mm): 0

Starting Y (mm): 0

Starting Z (mm): 0

Orientation Process Noise Velocity Model: 10

Orientation Process Noise Static: 10

Starting Sensor Gain: 0.15

Gain Q Dynamic: 1e-005

Starting Position and Orientation: Auto

Starting Azimuth (deg): 0


Starting Elevation (deg): 0

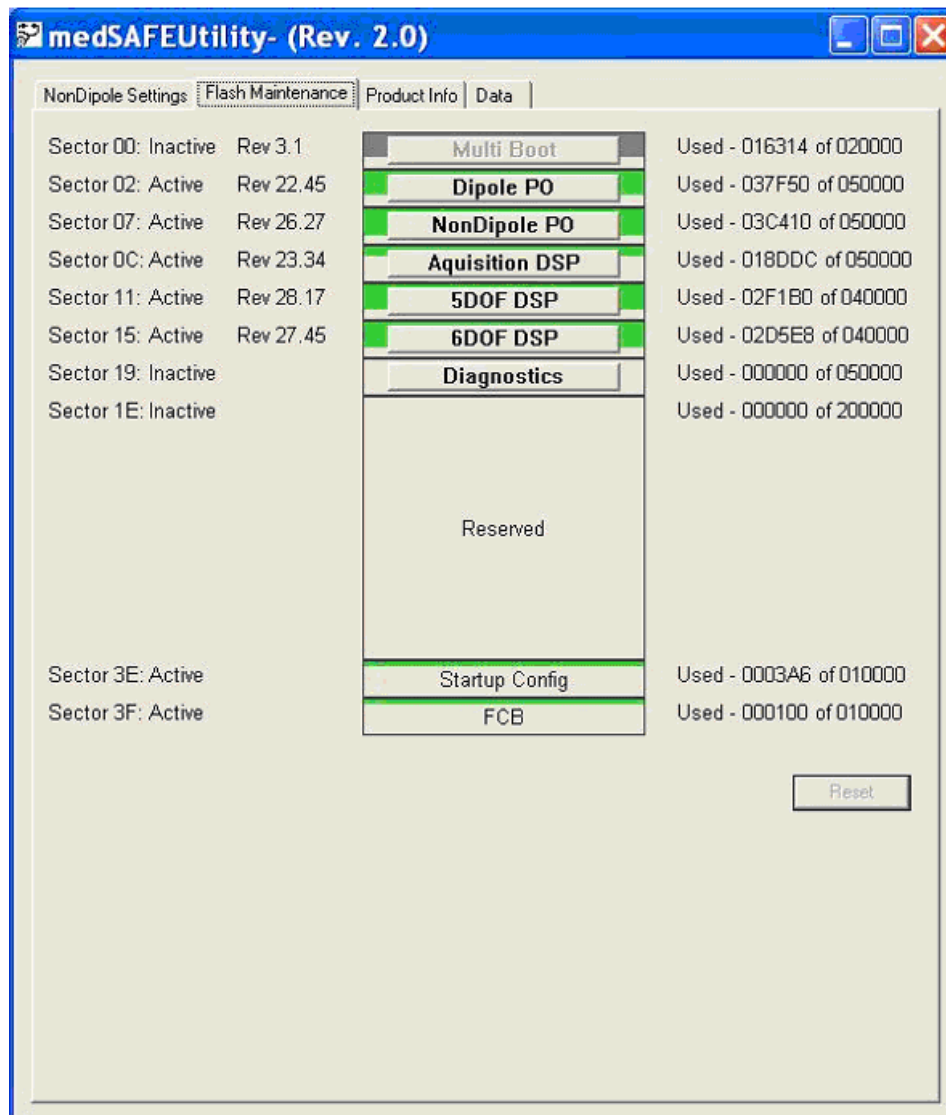
Starting Roll (deg): 0

Set To Defaults | Refresh | Apply | Reset

When the reading has completed, select 'Continue'. This will open the Utility to the main display and the 'Settings' tab:

6. If you want to change any of the power-up default settings, enter them here, and click 'Apply' to send them to the Flash memory.
7. To upgrade Flash Sectors, click the 'Flash Maintenance' tab to show the contents of the flash memory device.
8. Click the flash sector to be upgraded. Select the new loader file and click 'Open'. This will begin the upgrade of the flash sector. A progress bar will indicate status. When the upgrade of the sector is complete, the new contents of the Flash will be displayed in the 'Rev' field next to each sector.

 **Note:**  
Changes to power-up setting apply to non-dipole transmitters only.



9. Close the Utility ('X' in title bar) and cycle the power on the tracker.

**Note:**

First sector (Sector 00) should contain Rev 2.0 or greater of the Boot loader. If not, contact Tech Support.

## Appendix II: Application Notes

### Computing Stylus Tip Coordinates

In many applications that require a sensor mounted on a tool or pointing device, the position of the tip must be known and tracked. . This type of pointing device is generically referred to as a stylus or styllet.

The sensor position and orientation values are presented with respect to the center of the sensor. The corresponding X, Y, Z coordinates at the tip of the stylus may be easily calculated knowing the tip offset from the sensor center.

The stylus coordinates can be computed from the following:

$$X_S = X_B + X_O * M(1,1) + Y_O * M(2,1) + Z_O * M(3,1)$$

$$Y_S = Y_B + X_O * M(1,2) + Y_O * M(2,2) + Z_O * M(3,2)$$

$$Z_S = Z_B + X_O * M(1,3) + Y_O * M(2,3) + Z_O * M(3,3)$$

Where:  $X_B, Y_B, Z_B$  are the X, Y, Z position outputs from the 3DGuidance sensor with respect to the transmitter's center.

$X_O, Y_O, Z_O$  are the offset distances from the sensor's center to the tip of the stylus.

$X_S, Y_S, Z_S$  are the coordinates of the stylus's tip with respect to the transmitter's center.

$M(i, j)$  are the elements of the rotation matrix.

Often the values of  $X_O, Y_O, Z_O$  are not known ahead of time and must be calculated. This may be done by placing the tip of the stylus at a set location, collecting data of the stylus being repositioned with the tip fixed, and solving for  $X_O, Y_O, Z_O$ . Since the tip location ( $X_S, Y_S, Z_S$ ) is fixed, and the 3DGuidance sensor position ( $X_B, Y_B, Z_B$ ) and orientation ( $M(i, j)$ ) are reported by the system, solving for  $X_O, Y_O, Z_O$  may be solved.

Collect many measurement points over a large range of angles and rotations for maximum accuracy.