



***Ascension***  
*Technology Corporation*  
an **NDI** company

## **3D Guidance API Guide**

Revision 2  
November 2014

## Revision Status

Revision Number	Date	Description
1	June 2014	Initial issue
2	November 2014	Update addresses

Part Number: IL-1070236

Copyright 2014 Northern Digital Inc. All Rights Reserved.

---

Published by:

Northern Digital Inc.  
103 Randall Dr.  
Waterloo, Ontario, Canada N2V 1C5

Telephone: + (519) 884-5142  
Toll Free: + (877) 634-6340  
Global: + (800) 634 634 00  
Facsimile: + (519) 884-5184  
Website: [www.ndigital.com](http://www.ndigital.com)

Copyright 2013, Northern Digital Inc.

All rights reserved. No part of this document may be reproduced, transcribed, transmitted, distributed, modified, merged or translated into any language in any form by any means - graphic, electronic, or mechanical, including but not limited to photocopying, recording, taping or information storage and retrieval systems - without the prior written consent of Northern Digital Inc. Certain copying of the software included herein is unlawful. Refer to your software license agreement for information respecting permitted copying.

## **DISCLAIMER OF WARRANTIES AND LIMITATION OF LIABILITIES**

Northern Digital Inc. has taken due care in preparing this document and the programs and data on the electronic media accompanying this document including research, development, and testing.

This document describes the state of Northern Digital Inc.'s knowledge respecting the subject matter herein at the time of its publication, and may not reflect its state of knowledge at all times in the future. Northern Digital Inc. has carefully reviewed this document for technical accuracy. If errors are suspected, the user should consult with Northern Digital Inc. prior to proceeding. Northern Digital Inc. makes no expressed or implied warranty of any kind with regard to this document or the programs and data on the electronic media accompanying this document.

Northern Digital Inc. makes no representation, condition or warranty to the user or any other party with respect to the adequacy of this document or accompanying media for any particular purpose or with respect to its adequacy to produce a particular result. The user's right to recover damages caused by fault or negligence on the part of Northern Digital Inc. shall be limited to the amount paid by the user to Northern Digital Inc. for the provision of this document. In no event shall Northern Digital Inc. be liable for special, collateral, incidental, direct, indirect or consequential damages, losses, costs, charges, claims, demands, or claim for lost profits, data, fees or expenses of any nature or kind.

Product names listed are trademarks of their respective manufacturers. Company names listed are trademarks or trade names of their respective companies.

## **TRADEMARKS**

driveBAY and trakSTAR are trademarks of Ascension Technology Corporation

All other products mentioned in this guide are trademarks or registered trademarks of their respective companies.



# Table of Contents

Read Me First .....	vii
Contact Information .....	vii
 1 3D Guidance API Reference .....	 1
1.1 3D Guidance API Overview .....	1
1.2 3D Guidance API files .....	2
1.3 Sample Programs .....	3
1.4 Quick Reference .....	4
1.5 Pre-Initialization Setup .....	10
1.6 System Initialization .....	11
1.7 System Setup .....	12
1.8 Sensor Setup .....	14
1.9 Transmitter Setup .....	15
1.10 Acquiring Position and Orientation Data .....	16
1.11 Error Handling .....	17
1.12 3D Guidance API .....	18
 2 3D Guidance API Functions .....	 19
InitializeBIRDSystem .....	20
GetBIRDSystemConfiguration .....	22
GetTransmitterConfiguration .....	23
GetSensorConfiguration .....	25
GetBoardConfiguration .....	27
GetSystemParameter .....	29
GetSensorParameter .....	31
GetTransmitterParameter .....	33
GetBoardParameter .....	35
SetSystemParameter .....	37
SetSensorParameter .....	39
SetTransmitterParameter .....	41
SetBoardParameter .....	43
GetAsynchronousRecord .....	45
GetSynchronousRecord .....	47

GetBIRDError .....	50
GetErrorText .....	52
GetSensorStatus .....	54
GetTransmitterStatus .....	56
GetBoardStatus .....	58
GetSystemStatus .....	60
SaveSystemConfiguration .....	62
RestoreSystemConfiguration .....	64
CloseBIRDSystem .....	66
<b>3 3D Guidance API Structures .....</b>	<b>67</b>
SYSTEM_CONFIGURATION .....	68
TRANSMITTER_CONFIGURATION .....	70
SENSOR_CONFIGURATION .....	71
BOARD_CONFIGURATION .....	72
ADAPTIVE_PARAMETERS .....	74
QUALITY_PARAMETERS .....	75
VPD_COMMAND_PARAMETER .....	76
POST_ERROR_PARAMETER .....	77
DIAGNOSTIC_TEST_PARAMETERS .....	78
COMMUNICATIONS_MEDIA_PARAMETERS .....	81
BOARD_REVISIONS .....	83
SHORT_POSITION_RECORD .....	85
SHORT_ANGLES_RECORD .....	87
SHORT_MATRIX_RECORD .....	89
SHORT_QUATERNIONS_RECORD .....	91
SHORT_POSITION_ANGLES_RECORD .....	92
SHORT_POSITION_MATRIX_RECORD .....	94
SHORT_POSITION_QUATERNION_RECORD .....	95
DOUBLE_POSITION_RECORD .....	96
DOUBLE_ANGLES_RECORD .....	98
DOUBLE_MATRIX_RECORD .....	99
DOUBLE_QUATERNIONS_RECORD .....	101
DOUBLE_POSITION_ANGLES_RECORD .....	102
DOUBLE_POSITION_MATRIX_RECORD .....	104
DOUBLE_POSITION_QUATERNION_RECORD .....	105

---

DOUBLE_POSITION_TIME_STAMP_RECORD .....	106
DOUBLE_ANGLES_TIME_STAMP_RECORD .....	107
DOUBLE_MATRIX_TIME_STAMP_RECORD .....	108
DOUBLE_QUATERNIONS_TIME_STAMP_RECORD .....	109
DOUBLE_POSITION_ANGLES_TIME_STAMP_RECORD .....	110
DOUBLE_POSITION_MATRIX_TIME_STAMP_RECORD .....	112
DOUBLE_POSITION_QUATERNION_TIME_STAMP_RECORD .....	113
DOUBLE_POSITION_TIME_Q_RECORD .....	114
DOUBLE_ANGLES_TIME_Q_RECORD .....	116
DOUBLE_MATRIX_TIME_Q_RECORD .....	117
DOUBLE_QUATERNIONS_TIME_Q_RECORD .....	118
DOUBLE_POSITION_ANGLES_TIME_Q_RECORD .....	119
DOUBLE_POSITION_MATRIX_TIME_Q_RECORD .....	121
DOUBLE_POSITION_QUATERNION_TIME_Q_RECORD .....	123
SHORT_ALL_RECORD .....	125
DOUBLE_ALL_RECORD .....	127
DOUBLE_ALL_TIME_STAMP_RECORD .....	129
DOUBLE_ALL_TIME_STAMP_Q_RECORD .....	131
DOUBLE_ALL_TIME_STAMP_Q_RAW_RECORD .....	133
DOUBLE_POSITION_ANGLES_TIME_Q_BUTTON_RECORD .....	135
DOUBLE_POSITION_MATRIX_TIME_Q_BUTTON_RECORD .....	137
DOUBLE_POSITION_QUATERNION_TIME_Q_BUTTON_RECORD .....	139
 4 3D Guidance Enumeration Types.....	 141
BIRD_ERROR_CODES .....	142
SENSOR_PARAMETER_TYPE .....	147
MESSAGE_TYPE .....	154
TRANSMITTER_PARAMETER_TYPE .....	155
BOARD_PARAMETER_TYPE .....	157
SYSTEM_PARAMETER_TYPE .....	158
HEMISPHERE_TYPE .....	160
AGC_MODE_TYPE .....	161
DATA_FORMAT_TYPE .....	162
BOARD_TYPES .....	164
DEVICE_TYPES .....	165
COMMUNICATIONS_MEDIA_TYPES .....	167

PORT_CONFIGURATION_TYPE .....	168
AUXILIARY_PORT_TYPE .....	169
 5 3D Guidance API Status/Error Bit Definitions .....	 170
ERRORCODE .....	171
DEVICE_STATUS .....	172
 6 3D Guidance Initialization Files .....	 173
6.1 3D Guidance Initialization File Format Reference .....	173
 7 Ascension RS232 Interface Reference .....	 178
7.1 RS232 Signal Description .....	178
7.2 RS232 Commands .....	179
ANGLES .....	185
ANGLE ALIGN .....	186
BORESIGHT .....	187
BORESIGHT REMOVE .....	188
BUTTON MODE .....	189
BUTTON READ .....	190
CHANGE VALUE .....	191
EXAMINE VALUE .....	192
HEMISPHERE .....	206
MATRIX .....	208
METAL .....	210
OFFSET .....	212
POINT .....	214
POSITION .....	215
POSITION/ANGLES .....	216
POSITION/MATRIX .....	217
POSITION/QUATERNION .....	218
QUATERNION .....	219
REFERENCE FRAME .....	220
REPORT RATE .....	221
RESET .....	222
RS232 TO FBB .....	223



RUN .....	224
SLEEP .....	225
STREAM .....	226
STREAM STOP .....	227
7.3 Error Reporting .....	228

# List of Figures

Figure 3-1	Measurement Reference Frame (Mid-Range Transmitter) . . . . .	86
Figure 3-2	Measurement Reference Frame (Mid-Range Transmitter) . . . . .	96
Figure 4-1	Measurement Reference Frame (Standard Transmitter) . . . . .	148
Figure 4-2	Receiver Zero Orientation (8mm Sensor) . . . . .	148
Figure 4-3	Sensor Offset . . . . .	152
Figure 4-4	Measurement Reference Frame (Standard Transmitter) . . . . .	155
Figure 4-5	Receiver Zero Orientation (8mm Sensor) . . . . .	156
Figure 7-1	Rear View - 9 Pin D-Sub Connector . . . . .	178
Figure 7-2	Using the OFFSET command . . . . .	213

# Read Me First

This guide describes the 3D Guidance Application Program Interface (API).

Before sending any commands to the system, read the manual that accompanied your system to ensure that you have a full understanding of the functionality.

---

**Note** Throughout this guide, the term 3D Guidance refers to both the driveBAY and trakSTAR systems.

---

## Contact Information

If you have any questions regarding the content of this guide or the operation of this product, please contact us:



NDI INTERNATIONAL  
HEADQUARTERS

CANADA  
Tel: + 1 (877) 634-6340  
Email: [info@ndigital.com](mailto:info@ndigital.com)  
Website: [www.ndigital.com](http://www.ndigital.com)

ASCENSION TECHNOLOGY  
CORPORATION (an NDI company)

USA  
Tel: + 1 (802) 985-1114  
Email: [info@ndigital.com](mailto:info@ndigital.com)  
Website: [www.ascension-tech.com](http://www.ascension-tech.com)

NDI EUROPE GmbH

GERMANY  
Tel: + 49 (77 32) 8234-0  
Email: [info@ndieurope.com](mailto:info@ndieurope.com)  
Website: [www.ndieurope.com](http://www.ndieurope.com)

NDI ASIA PACIFIC

HONG KONG  
Tel: + (852) 2802 2205  
Email: [APinfo@ndigital.com](mailto:APinfo@ndigital.com)  
Website: [www.ndigital.com](http://www.ndigital.com)

[www.ndigital.com](http://www.ndigital.com)

[www.ascension-tech.com](http://www.ascension-tech.com)



# 1 3D Guidance API Reference

This guide details how to write an application that will access the tracker using the 3D Guidance API contained in the ATC3DG.dll. It also describes the setup of the tracker and defines the operational parameters.

## 1.1 3D Guidance API Overview

Communication with the tracker is achieved indirectly using the 3D Guidance API. It is the simplest method to communicate with the tracker. This interface is standard across Ascension's family of products and has been continued for the new generation of USB tracking devices. If you have previously written applications software using the pciBIRD/microBIRD driver for PCI bus-based trackers, you will find integration of the 3D Guidance tracker virtually seamless.

If you are new to the 3D Guidance family of trackers and the 3D Guidance API, you will find several tools on the CD-ROM for experimenting with the tracker's capabilities and quickly creating custom code. These tools include three demo applications, CUBES and APITest (formerly pciBIRDTalk), and two sample programs containing C++ project files. The tables below describe both the tools available and where they can be found following software installation.

## 1.2 3D Guidance API files

API Components	Description	Location (after software installation)
ATC3DG.h	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	Program Files(x86)\Ascension\3D GuidanceXXX\3DG API Developer\3DG API
ATC3DG.lib ATC3DG64.lib	Library file required during compiling of any code that makes calls to API (ATC3DG64.lib is used when building 64 bit applications.)	
ATC3DG.DLL ATC3DG64.DLL	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. (ATC3DG64.DLL is used when building 64 bit applications.)	
Applications	Description	Location
Cubes	<p>A Demo Windows application that displays tracking data (both test and graphical representation) for all sensors connected to the system. 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>NOTE: Cubes is a demo program and should NOT be used when fast data collection is required.</p>	Program Files(x86)\Ascension\3D Guidance XXXXX\Applications
APITest	<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 it in their original 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>	Program Files(x86)\Ascension\3D Guidance XXXXX\Applications

## 1.3 Sample Programs

Sample Code	Description	Location
Sample	<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>	Program Files(x86)\Ascension3D GuidanceXXXX\3DG API Developer\Samples\Sample\
Sample2	<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>	Program Files(x86)\Ascension3D GuidanceXXXX\3DG API Developer\Samples\Sample2\
GetSynchronous-Record Sample	<p>This C++ project is a version of the Sample project described above, which has been modified to collect as much data from the tracker as possible using the GetSynchronousRecord() call. 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), and sets the measurement rate to the maximum supported for the tracker.</li> <li>3. Turns on the transmitter</li> <li>4. If all above is valid, collects 10000 records (POS/ANG/TimeStamp format) from each of the attached sensors and streams them to a file. The status of the sensors is queried after each record is received.</li> <li>6. 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>7. 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>	Program Files(x86)\Ascension3D GuidanceXXXX\3DG API Developer\Samples\ GetSynchronous-RecordSample\

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

Quick Reference ([page 4](#))

Pre-Initialization Setup ([page 10](#))

System Initialization ([page 11](#))

System Setup ([page 12](#))

Sensor Setup ([page 14](#))

Transmitter Setup ([page 15](#))

Acquiring Position and Orientation Data ([page 16](#))

Error Handling ([page 17](#))

## 1.4 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 API defaults for short and mid-range transmitter configurations are as follows:

Transmitter:	No transmitter selected
Power Line Frequency:	60.0 (Hz)
Automatic Gain Control (AGC) Mode (not yet implemented):	Transmitter and Sensor AGC
Measurement Rate:	80.0 Hz
Position Scaling:	36.0 (inches, maximum range)
Metric:	False (floating point output representation is in inches)
Report rate:	1 (no decimation for the <code>GetSynchronousRecord()</code> function call)
Communications:	USB
Reset on initialization:	True (resets tracker to a known state)
Logging:	False (log file not generated)
Auto Configuration:	4 (4 sensors)

### SELECT\_TRANSMITTER

This command allows us to turn on next transmitter or turn off the current transmitter in the system. Refer to `SELECT_TRANSMITTER` ([page 158](#)) for a full description of the operation.



## POWER\_LINE\_FREQUENCY

This parameter represents the frequency of the AC power source used by the system. You need to set it for proper operation of the AC Wide Notch Filter ([page 7](#)).

## AGC\_MODE

---

**Note** Transmitter AGC mode is not currently implemented.

---

The 3DGuidance tracking systems have one mode of operation, and thus will operate the same way regardless of the setting. This mode is best described by the ‘Sensor’ section of the AGC (see [page 5](#)). 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 sample rate of the system. The tracker’s measurement rate is nominally set at 80.0 measurements/second. You can increase the tracker’s measurement rate to a maximum of 255 measurements/second for mid and short range transmitters. The downside of selecting rates faster than ~150 measurements/sec is that the errors introduced by nearby metals may increase. You can decrease the tracker’s measurement rate to no less than 20 measurements/sec. Decreasing the measurement rate is useful if you need 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, speed changes will have minimal effect. In such a case the metal is not inducing errors into the tracker’s measurements.

## REPORT\_RATE

The report rate is the decimation factor used when streaming data records from the tracking device to the user’s application. Stream mode is automatically set when using the [GetSynchronousRecord](#) command. The tracking device can compute a new P&O solution at 3 times the measurement rate, so at 80hz, an application will actually receive streaming data records at 240hz. The REPORT RATE provides a mechanism to decimate the streaming data records. A report rate of 2 would send every other data record, a report of 3 would send every 3<sup>rd</sup>, etc.

## Position Scaling

This represents the scale factor for position data returned as signed binary integers. The position scaling can be set to either 36, 72, or 144 (inches). We refer to it in our documentation as the MAXIMUM\_RANGE parameter. The value set will be the maximum possible full-scale value returned by the tracker. Note that the selection of scale factors greater than the default of 36 will reduce the resolution provided by the binary integer value accordingly.

## METRIC Position Representation

There is a system option that allows the data formatted in double precision floating point format to be output 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 setup can be performed individually for each sensor. The parameters can be set and read by making calls to [SetSensorParameter](#) and [GetSensorParameter](#). The factory defaults for each sensor channel are as follows:

Data Format:	Double precision floating point Position/Angles
Angle Align:	0, 0, 0
Filters:	AC Wide Notch: Enabled AC Narrow Notch: Disabled DC ADAPTIVE: Enabled All values in <b>Alpha max</b> table = <b>0.9000</b> , all values in <b>alpha min</b> table = <b>0.0200</b> , <b>Vm</b> table values = <b>2, 4, 4, 4, 4, 4, 4</b>
Hemisphere:	Front hemisphere (in front of the ATC logo on the transmitter)
Metal Distortion:	Filter alpha = 12, Slope = 0, Offset = 0 and Sensitivity = 2
Sensor Offsets:	0,0,0
Vital Product Data Storage (sensor and preamp):	Empty

## 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\\_RECORD](#), [DOUBLE\\_POSITION\\_MATRIX\\_TIME\\_Q\\_BUTTON\\_RECORD](#),

---

## DOUBLE\_POSITION\_QUATERNION\_TIME\_Q\_BUTTON\_RECORD

---

**Note** driveBAY units do not currently support the switch input.

---

These data formats for each sensor can be set up using the [SetSensorParameter](#) command. See [page 6](#) for all available data format combinations.

### ANGLE ALIGN

It aligns sensor to the tracked object coordinate system. These parameters can be set up for each sensor using the [SetSensorParameter](#) command. The current setting of ANGLE ALIGN can be accessed using the [GetSensorParameter](#) command. See [ANGLE ALIGN](#) for a full description of its meaning and usage.

### FILTERS

**DC Filter:** This filter is an adaptive alpha filter. It is initialized to a default condition. Its operation can be modified by changing the values in three tables that are contained in the [ADAPTIVE\\_PARAMETERS](#). These parameters are changed through use of the [SetSensorParameter](#), function call. The current state of the alpha filter parameters may be observed by calling the [GetSensorParameter](#) function call. See [ADAPTIVE\\_PARAMETERS](#) for a complete description of their meaning and use.

**AC Narrow Notch Filter:** A 2 tap finite impulse response (FIR) notch filter 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. See [FILTER\\_AC\\_NARROW\\_NOTCH](#) for a full description.

**AC Wide Notch Filter:** A 8 tap finite impulse response (FIR) filter applied to 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 correctly initialized using the [SetSystemParameter](#) function call.

**Sudden Output Change Filter:** If you select this filter, it will lock the output data to the current position and orientation if a sudden large change in position or orientation is detected. See [FILTER\\_LARGE\\_CHANGE](#) for a full description of its meaning and use.

### HEMISPHERE

The HEMISPHERE command tells the tracker the desired hemisphere of operation. This parameter determines which of the six possible hemispheres of the transmitter the sensor is operating in. It can be set up for each individual sensor by using the [SetSensorParameter](#) command. The current setting of HEMISPHERE can be accessed using the [GetSensorParameter](#) command. See [HEMISPHERE](#) for a full description of its meaning and usage.

### METAL DISTORTION

This command outputs an accuracy degradation indicator. It is also known as the "quality" number. The metal distortion value is output in certain 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 using the [SetSensorParameter](#) command. The current setting of The METAL DISTORTION parameters can be accessed using the [GetSensorParameter](#) command. See [QUALITY\\_PARAMETERS](#) for a description of the meaning and usage of the METAL DISTORTION parameters.

## SERIAL NUMBER\_

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

## SENSOR OFFSETS\_

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

## VITAL PRODUCT DATA STORAGE\_

User application data that is tied to a particular tracking sensor can be stored in the Vital Product Data (VPD) storage area on the individual sensor (or preamp – if applicable). The Vital Product Data parameter provides a mechanism for reading or writing individual bytes to this storage area. The VPD section comprises 112 bytes of user modifiable data storage. It is the user’s responsibility to define the contents and structure and to maintain that structure. See the Sensor Parameter type [VITAL\\_PRODUCT\\_DATA\\_RX\\_](#)

## MODEL STRING\_

The sensor’s model string can be obtained by calling [GetSensorParameter](#).

## PART NUMBER\_

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

## POINT

One data record is output from the selected sensor for each command issued. This command is performed when the [GetAsynchronousRecord](#) function call is issued.

Note that 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.

## STREAM

This is the mode the tracker is placed into when the [GetSynchronousRecord](#) function call is issued. Its usage and formatting is identical to the [GetAsynchronousRecord](#) function. Unlike the POINT command, however, STREAM mode commands the tracker to begin sending continuous data

records to the host PC (DLL) 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. If the host application does not keep up with the constant stream of data being provided in this mode, 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 (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.

The rate at which records are transmitted when using the `GetSynchronousRecord` can be changed through use of 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  $80 * 3 = 240$  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.

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.

## BOARD

Information regarding the printed circuit board (PCB) hardware is available. Apart from this information there are no operations necessary or available for interacting directly with the PCB.

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

**SOFTWARE REVISION NUMBER** The tracker's firmware version number is stored as a two-byte revision number. Use the [GetBoardConfiguration](#) command to access if for a specified board. Revisions of secondary firmware loads can be queried using the `GetBoardParameter` call and the **BOARD\_SOFTWARE\_REVISIONS** parameter type.

**POST ERRORS**, Selecting the `POST_ERROR_PCB` board parameter type with the [GetBoardParameter](#) call immediately after power-up, will return the results of the Power ON Self Test (POST). See the parameter structure [POST\\_ERROR\\_PARAMETER](#) for details

**VITAL PRODUCT DATA STORAGE**, User application data that is tied to a particular tracking board (electronics unit) can be stored in the Vital Product Data (VPD) storage area on the individual board. The Vital Product Data parameter provides a mechanism for reading or writing individual bytes to this storage area.. The VPD section comprises 112 bytes of user modifiable data storage. It is the user's responsibility to define the contents and structure and to maintain that structure. See the Board Parameter type [VITAL\\_PRODUCT\\_DATA\\_PCB](#).

**MODEL STRING**, The board's model string can be obtained by calling [GetBoardParameter](#).

**PART NUMBER**, The board's part number can be obtained by calling [GetBoardParameter](#).

## TRANSMITTER

The following operations apply only to transmitters.

**SELECT\_TRANSMITTER.** This command allows us to turn on next transmitter or turn off the current transmitter in the 3DGuidance™ system. A full description of the operation is found at [SELECT\\_TRANSMITTER](#).

**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. The factory defaults initialized the Reference Frame to 0,0,0 and the XYZ Reference Frame is disabled.

---

Note No transmitter is selected at power up.

---

**REFERENCE\_FRAME** Defines a new measurement reference frame. The new reference frame is provided as three angles describing the azimuth, elevation and roll angles. There is no offset component, and the reference frame remains centred on the transmitter. This parameter is changed or examined using the [SetTransmitterParameter](#) and [GetTransmitterParameter](#) function calls. See [TRANSMITTER\\_PARAMETER\\_TYPE](#)

**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. Note that 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](#).

**SERIAL\_NUMBER** The transmitter's serial number can be found by using [GetTransmitterParameter](#).

**VITAL\_PRODUCT\_DATA\_STORAGE.** User application data that is tied to a particular tracking transmitter can be stored in the Vital Product Data (VPD) storage area on the individual transmitter. The Vital Product Data parameter provides a mechanism for reading or writing individual bytes to this storage area. The VPD section comprises 112 bytes of user modifiable data storage. It is the user's responsibility to define the contents and structure and to maintain that structure. See the Transmitter Parameter type [VITAL\\_PRODUCT\\_DATA\\_TX](#)

**MODEL\_STRING.** The transmitter's model string can be obtained by calling [GetTransmitterParameter](#).

**PART\_NUMBER.** The transmitter's part number can be obtained by calling [GetTransmitterParameter](#).

## 1.5 Pre-Initialization Setup

There are some specific operations that are available PRIOR to initializing the 3DGuidance tracking system. These operations utilize parameters passed with the [GetSystemParameter](#) and [SetSystemParameter](#) calls, and provide API access to those items stored in the system configuration file (i.e. ATC3DG.ini). NOTE: If when using these 'Pre-Init' SetSystem parameters you receive the error 'BIRD\_ERROR\_UNABLE\_TO\_OPEN\_FILE', check to be sure current user has Windows modify privileges.

---

Note Changes to the *ATC3DG.ini* file via these commands require, ADMIN (modify) privileges.

---

## COMMUNICATIONS MEDIA

This parameter is used to configure the communications media that will be used for sending and receiving data with the tracking device. Current supported media types are USB and RS232. See the parameter structure: [COMMUNICATIONS\\_MEDIA\\_PARAMETERS](#) for additional details.

---

**Note** This parameter can be used prior to a `InitBIRDSystem()` function call. The Media Type is set to USB by default.

---

## LOGGING

Setting the LOGGING parameter to TRUE (using the SetSystem Parameter) will enable logging of the communications traffic between the API and the tracking device. Useful for reporting and trouble-shooting errors with the Ascension tracking system.

---

**Note** This parameter can be used prior to a `InitBIRDSystem()` function call. It is set to FALSE by default.

---

## RESET

This parameter is used to enable/disable the automatic reset of the tracking device that occurs with the InitializeBIRDSystem API call. Disabling reset will make the InitBIRDSystem function call perform much faster and may be useful in the development phase of a project, but this should be used with caution, as a reset places the tracking device in a known state and disabling this may cause undetermined side effects.

---

**Note** This parameter can be used prior to a `InitBIRDSystem()` function call. It is set to TRUE by default.

---

## AUTOCONFIG

This parameter can be used to specify the number of sensors to configure for the tracking device. This parameter is useful for systems that support the use of multiple 5DOF sensors in lieu of a single 6DOF sensor. 4 and 12 are the valid values for this parameter.

---

**Note** This parameter can be used prior to a `InitBIRDSystem()` function call. It is set to 4 by default.

---

## 1.6 System Initialization

With the exception of the 'Pre-init' commands, initialization must be performed before the tracker can be used. This is performed by calling the function `InitializeBIRDSystem()`. 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 indicate either a condition that has prevented the system from initializing or a prevailing condition that disallows the system from completing the initialization.

For example, the error code `BIRD_ERROR_COMMAND_TIME_OUT` typically 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 tracker cannot be used.

Initialization is an all-inclusive operation. Internally, the first task it performs is to enumerate the tracker connected to the system. Secondly, each board is queried concerning its 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 "ATC3DG.h"

.
.
int errorCode;

.
.

errorCode = InitializeBIRDSystem();

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

In order to use any 3D Guidance API calls, it is necessary to include the header file *ATC3DG.h*. The returned value *errorCode* must be declared as a variable of type *int*.

An error handler should be called that will display or log the error reported. The application should terminate since 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.)

## 1.7 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 "ATC3DG.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 initialize the system is to use the `RestoreSystemConfiguration()` call. 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.

---

**Note** Most tracker power-up settings can also be pre-configured using the Configuration Utility. See Power-Up Settings for details.

---

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

The system searches initially for the .inf 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.

## 1.8 Sensor Setup

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

---

**Note** The ALL\_SENSORS Sensor ID can only be used in the GetXXRecord() calls.

---

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

```

## 1.9 Transmitter Setup

The transmitter setup consists setting up a number of transmitter parameters. 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.

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. Note that 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](#).

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
```

## 1.10 Acquiring Position and Orientation Data

Data is acquired by making calls to [GetAsynchronousRecord](#) or [GetSynchronousRecord](#). 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.

---

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

---

```
/////////////////////////////////////  
//  
// 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);  
}
```

## 1.11 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.

Usually an error code is 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.

Even though all error codes indicate a fatal error condition, the software can recover from some failures. For example, if the system has not been initialized, the error code `BIRD_ERROR_SYSTEM_UNINITIALIZED` will be returned. The software can recover by calling `InitializeBIRDSystem()` before doing anything else, but this error is usually an indication of a software “bug”. Other errors, such as `BIRD_ERROR_NO_SENSOR_ATTACHED`, are recoverable by displaying a message to the user suggesting that they attach a sensor to the system.

### Status Codes

The status code returned by the `GetXXXStatus()` commands gives a bit-mapped indication of abnormal conditions that have been detected by the tracker for the device selected. This status code can be utilized effectively by the host application to track down hardware errors and conditions, as well as to help validate the position and orientation data records that have been returned from the sensor. For example, calling the `GetSensorStatus()` immediately after the `GetSynchronousRecord()` will allow the host application to determine if the ‘0’s received in the sensor data record were due to the sensor being saturated (`DEVICE STATUS` bit 2 set) or due to the position/orientation algorithm still initializing (`DEVICE STATUS` bit 13 set).

---

**Note** Call `GetSensorStatus()` after each data request to help validate the data returned from the sensor.

---

The use of the `Get...Status()` function call(s) has the advantage, from the host application stand point, that no additional tracking communications take place during the course of this API call. This makes the use of these calls very fast and efficient. Note that the status code is only updated with a call to either `GetAsynchronous` or `GetSynchronousRecord()`. Therefore the status returned is always the status associated with the last data record requested.

If the status code returned = 0, then the device or component is fully operational. Any status other than 0 indicates an error/abnormal condition, which may prevent successful operation of the device or component. 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. See the [DEVICE\\_STATUS](#) definition table for additional details.

## 1.12 3D Guidance API

The following elements define the API used by the 3D Guidance trackers.

3D Guidance API Functions ([page 19](#))

3D Guidance API Structures ([page 67](#))

3D Guidance API Enumeration Types ([page 141](#))

3D Guidance API Status/Error Bit Definitions ([page 170](#))

3D Guidance Initialization Files ([page 173](#))

## 2 3D Guidance API Functions

The following functions are used with 3D Guidance Systems:

- [“InitializeBIRDSystem” on page 20](#)
- [“GetBIRDSystemConfiguration” on page 22](#)
- [“GetTransmitterConfiguration” on page 23](#)
- [“GetSensorConfiguration” on page 25](#)
- [“GetBoardConfiguration” on page 27](#)
- [“GetSystemParameter” on page 29](#)
- [“GetSensorParameter” on page 31](#)
- [“GetTransmitterParameter” on page 33](#)
- [“GetBoardParameter” on page 35](#)
- [“SetSystemParameter” on page 37](#)
- [“SetSensorParameter” on page 39](#)
- [“SetTransmitterParameter” on page 41](#)
- [“SetBoardParameter” on page 43](#)
- [“GetAsynchronousRecord” on page 45](#)
- [“GetSynchronousRecord” on page 47](#)
- [“GetBIRDError” on page 50](#)
- [“GetErrorText” on page 52](#)
- [“GetSensorStatus” on page 54](#)
- [“GetTransmitterStatus” on page 56](#)
- [“GetBoardStatus” on page 58](#)
- [“GetSystemStatus” on page 60](#)
- [“SaveSystemConfiguration” on page 62](#)
- [“RestoreSystemConfiguration” on page 64](#)
- [“CloseBIRDSystem” on page 66](#)

## InitializeBIRDSystem

The **InitializeBIRDSystem** function resets and initializes the 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_INVALID_DEVICE_ID	A device has been found that is not supported by this API dll. Verify 3D Guidance model installed.
BIRD_ERROR_FAILED_LOCKING_DEVICE	This error is for a PCIBIRD system only. Driver could not lock PCIBIRD resources. Check that there is not another application using the hardware.
BIRD_ERROR_BOARD_MISSING_ITEMS	This error is for a PCIBIRD system only. The required resources were not found defined in the configuration registers. Possible corrupt configuration. If error is repeatable there is an unrecoverable hardware failure.
BIRD_ERROR_INCORRECT_PLD	The PLD version on the hardware is incompatible with this version of the API dll. Verify model installed.
BIRD_ERROR_COMMAND_TIME_OUT	The 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	The 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 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 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. 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.



BIRD_ERROR_INVALID_CONFIGURATION,	The system has detected an invalid Multi-Unit Sync (MUS) configuration. This may be caused by an invalid Unit ID setting or more than one transmitter connected to the system. See the MUS Installation Addendum for assistance with MUS configuration setup.
-----------------------------------	---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

## Remarks

If the [RESET](#) system parameter is enabled, this function call will first reset all 3DGuidance units connected to the system. 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.

---

**Note** This function must be called first, before other commands can be sent.

---

## Requirements

**Header:** Declared in ATC3DG.h

**Library:** Use ATC3DG.lib (ATC3DG64.lib for 64 bit applications)

See Also

[CloseBIRDSystem](#),  
[RestoreSystemConfiguration](#)

## 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 3DGuidance 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

**Header:** Declared in ATC3DG.h

**Library:** Use ATC3DG.lib (ATC3DG64.lib for 64 bit applications)

See Also

[GetSystemParameter](#),  
[SetSystemParameter](#),  
[GetSystemStatus](#),  
[SaveSystemConfiguration](#)

## 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 3DGuidance hardware and system has not been initialized yet. The <a href="#">InitializeBIRDSysyem</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. Note however that the DEVICE\_TYPES enumerated type returned with this call WILL NOT support future transmitters. The MODEL\_STRING and PART\_NUMBER parameter types have replaced this functionality.

## Requirements

**Header:** Declared in ATC3DG.h

**Library:** Use ATC3DG.lib (ATC3DG64.lib for 64 bit applications)

See Also

[GetTransmitterConfiguration](#),  
[SetTransmitterParameter](#),  
[GetTransmitterStatus](#)

## 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 3DGuidance hardware and system has not been initialized yet. The <a href="#">InitializeBIRDSysyem</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. Note however that the DEVICE\_TYPES enumerated type returned with this call WILL NOT support future transmitters. The MODEL\_STRING and PART\_NUMBER parameter types have replaced this functionality.

## Requirements

**Header:** Declared in ATC3DG.h

**Library:** Use ATC3DG.lib (ATC3DG64.lib for 64 bit applications)

See Also

[GetSensorParameter](#),  
[SetSystemParameter](#),  
[GetSensorStatus](#)

## 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 3DGuidance hardware and system has not been initialized yet. The <a href="#">InitializeBIRDSys</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

**Header:** Declared in ATC3DG.h

**Library:** Use ATC3DG.lib (ATC3DG64.lib for 64 bit applications)

See Also

[GetBoardParameter](#),  
[SetBoardParameter](#),  
[GetBoardStatus](#),  
[GetSystemStatus](#)



## 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.

---

**Note** 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 3DGuidance 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.

### Remarks

The [GetSystemParameter](#) and [SetSystemParameter](#) commands 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

**Header:** Declared in ATC3DG.h

**Library:** Use ATC3DG.lib (ATC3DG64.lib for 64 bit applications)

See Also

[SetSystemParameter](#),  
[GetSystemStatus](#)

## 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 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 3DGuidance 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 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

**Header:** Declared in `ATC3DG.h`

**Library:** Use `ATC3DG.lib` (`ATC3DG64.lib` for 64 bit applications)

See Also

[SetSensorParameter](#),  
[GetSensorConfiguration](#),  
[GetSensorStatus](#)

## 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 3DGuidance 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.

## 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

**Header:** Declared in `ATC3DG.h`

**Library:** Use `ATC3DG.lib` (`ATC3DG64.lib` for 64 bit applications)

See Also

[SetTransmitterParameter](#),  
[GetTransmitterConfiguration](#),  
[GetTransmitterStatus](#)

## 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 3DGuidance 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

**Header:** Declared in ATC3DG.h

**Library:** Use ATC3DG.lib (ATC3DG64.lib for 64 bit applications)

See Also

[GetBoardParameter](#),  
[GetBoardConfiguration](#),  
[GetBoardStatus](#)



## SetSystemParameter

The **SetSystemParameter** function allows an application to change basic 3DGuidance 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 3DGuidance hardware and system has not been initialized yet. The <a href="#">InitializeBIRDSys</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: Turn off the currently running transmitter and there is no transmitter attached to the system Turn on the transmitter with the selected ID and there is no transmitter attached at that ID.

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. 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. See SYSTEM\_PARAMETER\_TYPE for a description of the individual parameters

## Requirements

**Header:** Declared in ATC3DG.h

**Library:** Use ATC3DG.lib (ATC3DG64.lib for 64 bit applications)

See Also

[GetSystemParameter](#),  
[GetSystemStatus](#),  
[GetBIRDSystemConfiguration](#)

## SetSensorParameter

The **SetSensorParameter** function allows an application to select and change specific characteristics of individual sensors in a 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 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 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 3DGuidance hardware and system has not been initialized yet. The <a href="#">InitializeBIRDSys</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 <i>sensorID</i> 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. 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

**Header:** Declared in ATC3DG.h

**Library:** Use ATC3DG.lib (ATC3DG64.lib for 64 bit applications)

See Also

[GetSensorParameter](#),  
[GetSensorConfiguration](#),  
[GetSensorStatus](#)

## 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 3DGuidance 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	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. 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

**Header:** Declared in ATC3DG.h

**Library:** Use ATC3DG.lib (ATC3DG64.lib for 64 bit applications)

See Also

[GetTransmitterParameter](#),  
[GetTransmitterConfiguration](#),  
[GetTransmitterStatus](#)

## 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 3DGuidance 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 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

**Header:** Declared in ATC3DG.h

**Library:** Use ATC3DG.lib (ATC3DG64.lib for 64 bit applications)

See Also

[GetBoardParameter](#),  
[GetBoardConfiguration](#),  
[GetBoardStatus](#)



## GetAsynchronousRecord

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

```
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 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 overall record, see *pRecord* above.

### 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="#">InitializeBIRDSysyem</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 GetAsynchronousRecord 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 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(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.

See the [Status Codes](#) reference section for recommendations on using the GetSensorStatus() call immediately after the GetXXXXRecord request. This can be an effective means of validating the position and orientation data records returned from the sensors.

## Requirements

**Header:** Declared in ATC3DG.h

**Library:** Use ATC3DG.lib (ATC3DG64.lib for 64 bit applications)

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 overall record, see *PRecord* above.

### 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="#">InitializeBIRDSsystem</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 Report Rate divisor. 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 system 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 Report Rate system parameter type, and 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 `GetXXXXConfiguration`). 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.

See the [Status Codes](#) reference section for recommendations on using the GetSensorStatus() call immediately after the GetXXXXRecord request. This can be an effective means of validating the position and orientation data records returned from the sensors.

## Requirements

**Header:** Declared in ATC3DG.h

**Library:** Use ATC3DG.lib (ATC3DG64.lib for 64 bit applications)

See Also

[GetAsynchronousRecord](#)

## 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 3DGuidance hardware and system has not been initialized yet. The <a href="#">InitializeBIRDSystem</a> function must be called first.
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 `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 `GetXXXRecord`. For example: In a system with two boards there will exist eight sensor channels. Calling `GetBIRDError` will acquire the current status for all eight 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

**Header:** Declared in ATC3DG.h

**Library:** Use ATC3DG.lib (ATC3DG64.lib for 64 bit applications)

See Also

[GetErrorText](#)

## 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
<code>SIMPLE_MESSAGE</code>	A single line text string will be returned with a terse description of the error.
<code>VERBOSE_MESSAGE</code>	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
<code>BIRD_ERROR_SUCCESS</code>	No errors occurred. Call completed successfully
<code>BIRD_ERROR_SYSTEM_UNINITIALIZED</code>	The 3DGuidance 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 BIRD_ERROR_CODES was passed in parameter <i>errorCode</i> .
--------------------------------------	-------------------------------------------------------------------------------------------------

## Remarks

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

## Requirements

**Header:** Declared in ATC3DG.h

**Library:** Use ATC3DG.lib (ATC3DG64.lib for 64 bit applications)

See Also

[GetBIRDError](#)

## 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	S	B	R	T
0	GLOBAL_ERROR	Global error bit. If any 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_RUNNING	An attempt was made to call <a href="#">GetAsynchronousRecord</a> when no transmitter was running.	x	x	x	
8	BAD_12V	N/A for the 3DG systems				
9	CPU_TIMEOUT	N/A for the 3DG systems				
10	INVALID_DEVICE	N/A for the 3DG systems				
11	NO_TRANSMITTER_ATTACHED	A transmitter is not attached to the tracking system.	x	x	x	x
12	OUT_OF_MOTIONBOX	The sensor has exceeded the maximum range and the position has been clamped to the maximum range			X	
13	ALGORITHM_INITIALIZING	The sensor has not acquired enough raw magnetic data to compute an accurate P&O solution.			x	
14 - 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.

### 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.

The use of the Get...Status() function call(s) has the advantage, from the host application stand point, that no additional tracking communications take place during the course of this API call. This makes the use of these calls very fast and efficient, lending to use of the status value as an effective means for the host application to validate the position and orientation data records from the sensor.

Note that the status code is only updated with a call to either GetAsynchronous or GetSynchronousRecord( ). Therefore the status returned is always the status associated with the last data record requested. Determining if the sensor is operational can be done by simply testing to ensure that the status word = 0.

## Requirements

**Header:** Declared in ATC3DG.h

**Library:** Use ATC3DG.lib (ATC3DG64.lib for 64 bit applications)

See Also

[GetSensorConfiguration](#),  
[GetSensorParameter](#),  
[SetSensorParameter\\_](#)

## 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	S	B	R	T
0	GLOBAL_ERROR	Global error bit. If any 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="#">InitializeBIRDSys</a>	x	x	x	x
7	NO_TRANSMITTER_RUNNING	An attempt was made to call <a href="#">GetAsynchronousRecord</a> when no transmitter was running.	x	x	x	
8	BAD_12V	N/A for the 3DG systems				
9	CPU_TIMEOUT	N/A for the 3DG systems				
10	INVALID_DEVICE	N/A for the 3DG systems				
11	NO_TRANSMITTER_ATTACHED	A transmitter is not attached to the tracking system.	x	x	x	x
12	OUT_OF_MOTION-BOX	The sensor has exceeded the maximum range and the position has been clamped to the maximum range			X	
13	ALGORITHM_INITIALIZING	The sensor has not acquired enough raw magnetic data to compute an accurate P&O solution.			x	
14 - 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.

## 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 InitializeBIRDSYSTEM 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 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) transmitter ID then the “Non-Existent” bit will be set.

Determining if the transmitter is operational can be done by simply testing to ensure that the status word = 0.

## Requirements

**Header:** Declared in ATC3DG.h

**Library:** Use ATC3DG.lib (ATC3DG64.lib for 64 bit applications)

See Also

[GetTransmitterConfiguration](#),  
[GetTransmitterParameter](#),  
[SetTransmitterParameter](#)

## GetBoardStatus

The **GetBoardStatus** will return the status of the selected 3D guidance card.

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

### Parameters

*boardID*

[in] The boardID is in the range 0..(n-1) where n is the number of units connected to the system.

### Return Values

The function returns a value of type *DEVICE\_STATUS*. The returned value contains the status of the selected 3Dguidance card. The bits in the status word have the following meanings:

Bit	Name	Meaning	S	B	R	T
0	GLOBAL_ERROR	Global error bit. If any 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="#">InitializeBIRDSsystem</a>	x	x	x	x
7	NO_TRANSMITTER_RUNNING	An attempt was made to call <a href="#">GetAsynchronousRecord</a> when no transmitter was running.	x	x	x	
8	BAD_12V	N/A for the 3DG systems				
9	CPU_TIMEOUT	N/A for the 3DG systems				
10	INVALID_DEVICE	N/A for the 3DG systems				
11	NO_TRANSMITTER_ATTACHED	A transmitter is not attached to the tracking system.	x	x	x	x
12	OUT_OF_MOTIONBOX	The sensor has exceeded the maximum range and the position has been clamped to the maximum range			X	
13	ALGORITHM_INITIALIZING	The sensor has not acquired enough raw magnetic data to compute an accurate P&O solution.			x	
14 - 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.

### Remarks

This function takes as its only parameter an index to a selected 3Dguidance 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 InitializeBIRDSystem 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 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) board ID then the “Non-Existent” bit will be set.

Determining if the board is operational can be done by simply testing to ensure that the status word = 0.

## Requirements

**Header:** Declared in ATC3DG.h

**Library:** Use ATC3DG.lib (ATC3DG64.lib for 64 bit applications)

See Also

[GetBoardConfiguration](#),  
[GetBoardParameter](#),  
[SetBoardParameter](#)

## GetSystemStatus

The **GetSystemStatus** will return the status of the 3DGuidance 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 3DGuidance system. The bits in the status word have the following meanings:

Bit	Name	Meaning	S	B	R	T
0	GLOBAL_ERROR	Global error bit. If any 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="#">InitializeBIRDSys</a>	x	x	x	x
7	NO_TRANSMITTER_RUNNING	An attempt was made to call <a href="#">GetAsynchronousRecord</a> when no transmitter was running.	x	x	x	
8	BAD_12V	N/A for the 3DG systems				
9	CPU_TIMEOUT	N/A for the 3DG systems				
10	INVALID_DEVICE	N/A for the 3DG systems				
11	NO_TRANSMITTER_ATTACHED	A transmitter is not attached to the tracking system.	x	x	x	x
12	OUT_OF_MOTIONBOX	The sensor has exceeded the maximum range and the position has been clamped to the maximum range			X	
13	ALGORITHM_INITIALIZING	The sensor has not acquired enough raw magnetic data to compute an accurate P&O solution.			x	
14 - 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.

### 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 [InitializeBIRDSys](#) has been called. If the function [InitializeBIRDSys](#) has not been called then the “UnInitialized” bit will be set.



Determining if the system is operational can be done by simply testing the status word. The system is only operational when the status word = 0.

## Requirements

**Header:** Declared in ATC3DG.h

**Library:** Use ATC3DG.lib (ATC3DG64.lib for 64 bit applications)

See Also

[GetBIRDSystemConfiguration](#),  
[GetSystemParameter](#),  
[SetSystemParameter](#)

## 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 3DGuidance hardware and system has not been initialized yet. The <a href="#">InitializeBIRDSysyem</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");
```

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 which 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 [3D Guidance Initialization File Format](#) section.

## Requirements

**Header:** Declared in ATC3DG.h

**Library:** Use ATC3DG.lib (ATC3DG64.lib for 64 bit applications)

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 3DGuidance 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 <a href="#">SaveSystemConfiguration()</a> 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.

### 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");
```

if the full pathname specification is not provided then the default search path is in the working 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 [3D Guidance Initialization File Format Reference](#)

## Requirements

**Header:** Declared in `ATC3DG.h`

**Library:** Use `ATC3DG.lib` (`ATC3DG64.lib` for 64 bit applications)

See Also

[SaveSystemConfiguration](#)

## CloseBIRDSystem

The **CloseBIRDSystem** function closes the 3DGuidance system down.

```
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 3DGuidance system 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 3DGuidance system in order to prevent memory and/or resource leaks.

### Requirements

**Header:** Declared in ATC3DG.h

**Library:** Use ATC3DG.lib (ATC3DG64.lib for 64 bit applications)

See Also

[InitializeBIRDSystem](#)

## 3 3D Guidance API Structures

The following structures are used with the 3DGuidance system.

*SYSTEM\_CONFIGURATION*  
*SENSOR\_CONFIGURATION*  
*TRANSMITTER\_CONFIGURATION*  
*BOARD\_CONFIGURATION*  
*ADAPTIVE\_PARAMETERS*  
*QUALITY\_PARAMETERS*  
*VPD\_COMMAND\_PARAMETER*  
*POST\_ERROR\_PARAMETER*  
*DIAGNOSTIC\_TEST\_PARAMETERS*  
*COMMUNICATIONS\_MEDIA\_PARAMETERS*  
*BOARD\_REVISIONS*

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\_RECORD*  
*DOUBLE\_POSITION\_MATRIX\_TIME\_Q\_BUTTON\_RECORD*  
*DOUBLE\_POSITION\_QUATERNION\_TIME\_Q\_BUTTON\_RECORD*

## 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. Default is 80.0 Hz.

#### **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 3DGuidance cards connected/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

**Header:** Declared in ATC3DG.h

**Library:** Use ATC3DG.lib (ATC3DG64.lib for 64 bit applications)

## 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](#). Note however that the `DEVICE_TYPES` enumerated type **WILL NOT** support future transmitters. The `MODEL_STRING` and `PART_NUMBER` parameter types have replaced this functionality.

#### **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

**Header:** Declared in `ATC3DG.h`

**Library:** Use `ATC3DG.lib` (`ATC3DG64.lib` for 64 bit applications)

## 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](#). Note however that the `DEVICE_TYPES` enumerated type WILL NOT support future sensors. The `MODEL_STRING` and `PART_NUMBER` parameter types have replaced this functionality.

#### **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

**Header:** Declared in `ATC3DG.h`

**Library:** Use `ATC3DG.lib` (`ATC3DG64.lib` for 64 bit applications)

## BOARD\_CONFIGURATION

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

```
typedef struct tagBOARD_CONFIGURATION{
    ULONG          serialNumber;
    BOARD_TYPEType;
    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](#). Note however that the BOARD\_TYPES enumerated type WILL NOT support future boards. The MODEL\_STRING and PART\_NUMBER parameter types have replaced this functionality.

#### **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

**Header:** Declared in ATC3DG.h

**Library:** Use ATC3DG.lib (ATC3DG64.lib for 64 bit applications)

## 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

**Header:** Declared in ATC3DG.h

**Library:** Use ATC3DG.lib (ATC3DG64.lib for 64 bit applications)

## QUALITY\_PARAMETERS

The **QUALITY\_PARAMETERS** structure contains the parameters used to setup the distortion detection algorithm for an individual sensor channel. For further information see [“QUALITY” on page 150](#).

```
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

**Header:** Declared in ATC3DG.h

**Library:** Use ATC3DG.lib (ATC3DG64.lib for 64 bit applications)

## VPD\_COMMAND\_PARAMETER

The **VPD\_COMMAND\_PARAMETER** structure contains the parameters used during reading and writing to the Vital Product Data (VPD) storage area. The VPD is a 512-byte storage area located in EEPROM on the main electronic unit (EU). Using the **SetSystemParameter()** and **GetSystemParameter()** 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 address of a byte location within the VPD that is the target for either a read or a write operation.

#### **value**

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

---

Note The VITAL\_PRODUCT\_DATA\_PCB ([BOARD\\_PARAMETER\\_TYPE](#)) has replaced this functionality.

---

### Requirements

**Header:** Declared in ATC3DG.h

**Library:** Use ATC3DG.lib (ATC3DG64.lib for 64 bit applications)



## POST\_ERROR\_PARAMETER

The **POST\_ERROR\_PARAMETER** structure contains the parameters used to report errors generated during the POST (Power on Self-Test) sequence. Note that when used with Multi-Unit Synch configurations, passing this parameter with the `GetSystemParameter()` call will only access those POST errors generated on the first unit (Multi-Unit ID =0). The `POST_ERROR_PCB` board parameter type should be used with the `GetBoardParameter()` call, for accessing these errors across multiple units.

```
typedef struct tag POST_ERROR_PARAMETER{
USHORT error;
  UCHAR      channel;
  UCHAR      fatal;
  UCHAR      moreErrors;
} POST_ERROR_PARAMETER;
```

### Members

#### **error**

Contains a 32-bit value representing the error code parameter that was reported from the POST sequence.

This value takes the form of a standard [ERRORCODE](#), indicating success or failure. The enumerated error code field contained within the 32-bit [ERRORCODE](#) will be of the type [BIRD\\_ERROR\\_CODES](#). A message string describing the nature of the error code can be obtained by passing the error to the `GetErrorText` function.

#### **channel**

This value contains the ID number of the sensor channel in which the POST error was detected.

#### **fatal**

This value indicates whether or not the error detected and reported by the POST sequence is a fatal error or just a warning.

#### **moreErrors**

The `moreErrors` value is used to indicate if additional POST errors are in the Error buffer, waiting to be read.

### Requirements

**Header:** Declared in `ATC3DG.h`

**Library:** Use `ATC3DG.lib` (`ATC3DG64.lib` for 64 bit applications)

## DIAGNOSTIC\_TEST\_PARAMETERS

The **DIAGNOSTIC\_TEST\_PARAMETERS** structure contains the parameters needed to perform various diagnostic test functions. It is used by the `GetSystemParameter()` and the `SetSystemParameter()` function calls. Note however that when used with Multi-Unit Synch configurations, the Get/Setsystem parameter will only access those diagnostic tests on the first unit (Multi-Unit ID =0). The **DIAGNOSTIC\_TEST\_PCB\_board** parameter type should be used with the `Get/SetBoardParameter()` calls, for accessing these tests across multiple units.

The diagnostic tests provided by a system are divided into suites of tests, where each suite typically provides a group of related tests, for example, sensor tests. By passing this structure with the parameters appropriately set the user can select one, an entire suite or the entire set of tests to be executed.

NOTE: Error terminology used within this command: There are 2 basic types of error. The first type is an error in the command invocation, namely, use of incorrect parameter values and/or sizes. This is called a **Call Error**. The second type of error is the error reported back from the tracking system as a consequence of having performed and failed a diagnostic test. This is called a **Diagnostic Error**.

```
typedef struct tagDIAGNOSTIC_TEST_PARAMETERS{
    UCHAR          suite;
    UCHAR          test;
    UCHAR          suites;
    UCHAR          tests;
    PCHAR *pTestName;
    USHORT testNameLen;
    USHORT *diagError;
} DIAGNOSTIC_TEST_PARAMETERS, *PDIAGNOSTIC_TEST_PARAMETERS;
```

### Members

#### suite

This parameter determines which test suite is being referenced. The test suites are numbered using a base of 1. If the suite number is set greater than the maximum number of suites available then a Call Error message will be returned. If the suite number is set to zero then the entire diagnostic set is being referenced.

#### test

This parameter is used to select the individual test within a test suite to reference. The tests are numbered using a base of 1. If the test number provided exceeds the number of tests in the suite a Call Error will be returned. If the test number passed is zero then the entire set of tests in the selected suite is being referenced.

#### suites

This parameter is a “don’t care” when passed to the function. Upon return it can have a number of different meanings depending upon the situation where it was used. See .

#### tests

This parameter is a “don’t care” when passed to the function. Upon return it can have a number of different meanings depending upon the situation where it was used. See .

**pTestName**

This parameter is a pointer provided to a buffer, which should be large enough to hold a string containing the name of the last test to be referenced by the selected diagnostic(s). If the diagnostics all run to completion successfully this name will be the name of the last test. The user should provide a buffer 64 bytes long. This buffer is long enough to contain 2 names each 32 bytes long. The first name is the title of the Test Suite and the second name if present is the title of the individual Diagnostic Test referenced.

**testNameLen**

This parameter is an unsigned short whose value is the length of the TestName buffer provided by the user whose pointer is provided by pTestName. It is essential that the length parameter match the actual length of the buffer supplied otherwise buffer overruns may occur with unpredictable consequences.

**diagError**

This parameter is a USHORT where the call will return an error code if the diagnostic fails otherwise the contents will be zero.

GetSystemParameter() and SetSystemParameter() Diagnostic Test Usage							
Call	Action	Input Param.		Output Parameter			
		Suite	Test	Suites	Tests	TestName	DiagError
GetSystemParameter()	Get number of test suites available.	0	0 (Don't care)	Total number of suites available. If no diagnostics are supported this value will be zero.	Don't care	Don't Care	Don't Care.
GetSystemParameter()	Get number of tests available in selected suite.	N	0	Suite number N is returned	Total number of tests available in the suite	32 character zero terminated string containing the <b>Suite Name</b>	Error* if N > Total number of available suites.
GetSystemParameter()	Get string name of individual test.	N	M	Suite number N is returned	Test number M is returned.	64 character zero terminated string containing the <b>Suite/Test Name</b> .	Error* if N > Total number of available suites. Error if M > Total number of available tests within this suite.
SetSystemParameter()	Execute entire set of available diagnostic tests.	0	0 (Don't Care)	<b>If successful</b> returns the number of the final suite. <b>If fails</b> stops at and returns the number of the suite	<b>If successful</b> returns the number of the final test. <b>If fails</b> stops at and returns the number of the test	64 character zero terminated string containing the <b>Suite/Test Name</b> .	<b>If successful</b> returns an error code of zero <b>If fails</b> returns an error code. *

SetSys- temPa- rameter()	Execute entire set of tests for the selected suite.	N	0	Suite number N is returned.	<b>If successful</b> returns the number of the final test. <b>If fails</b> stops at and returns the number of the test	64 character zero terminated string containing the <b>Suite/Test name.</b>	<b>If successful</b> returns an error code of zero <b>If fails</b> returns an error code. *
SetSys- temPa- rameter()	Execute an individ- ual diag- nostic test.	N	M	Suite number N is returned.	Test number M is returned.	64 character zero terminated string containing the <b>Suite/Test name.</b>	<b>If successful</b> returns an error code of zero <b>If fails</b> returns an error code. (See note below.)

---

**Note** A character string describing the error condition can be obtained by calling GetErrorText() and passing the error code.

---

## Requirements

**Header:** Declared in PCIBird3.h

**Library:** Use PCIBird3.lib

## COMMUNICATIONS\_MEDIA\_PARAMETERS

The COMMUNICATIONS\_MEDIA\_PARAMETERS structure contains the parameters needed to configure the various communication protocols. It can be used by the GetSystemParameter() and the SetSystemParameter() function calls prior to initialization, and provides API access to those items stored in the system configuration file (i.e. ATC3DG.ini). NOTE: Users without Windows Modify privileges (i.e. Admin) may receive the error 'BIRD\_ERROR\_UNABLE\_TO\_OPEN\_FILE' when calling the SetSystemParameter() call with these 'pre-init' parameters.

```
typedef struct tagCOMMUNICATIONS_MEDIA_PARAMETERS
{
    COMMUNICATIONS_MEDIA_TYPE mediaType;
    union
    {
        struct
        {
            CHAR    comport[64];
        } rs232;
        struct
        {
            USHORT port;
            CHAR    ipaddr[64];
        } tcpip;
    };
} COMMUNICATIONS_MEDIA_PARAMETERS;
```

### Members

#### mediaType

This parameter defines the communications media to use for subsequent calls to InitBIRDSystem(). See COMMUNICATIONS\_MEDIA\_TYPES for valid values.

#### rs232.comport

This parameter is used only for the RS232 media type. This parameter defines the name of the RS232 communications port. Valid string format are "COMX" where X is the RS232 communications port number.

#### tcpip.port

This parameter is used only for the TCPIP media type. This parameter defines the port number to use for establishing a TCP/IP session with the tracking device.

#### tcpip.ipaddr

This parameter is used only for the TCPIP media type. This parameter defines IP address to use for establishing a TCP/IP session with the tracking device. This parameter is required to be in dotted decimal internet addressing format.

## Requirements

**Header:** Declared in ATC3DG.h

**Library:** Use ATC3DG.lib (ATC3DG64.lib for 64 bit applications)

## 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 nondipole DSP.

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

Minor revision number for the nondipole 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

**Header:** Declared in ATC3DG.h

**Library:** Use ATC3DG.lib (ATC3DG64.lib for 64 bit applications)



## 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.

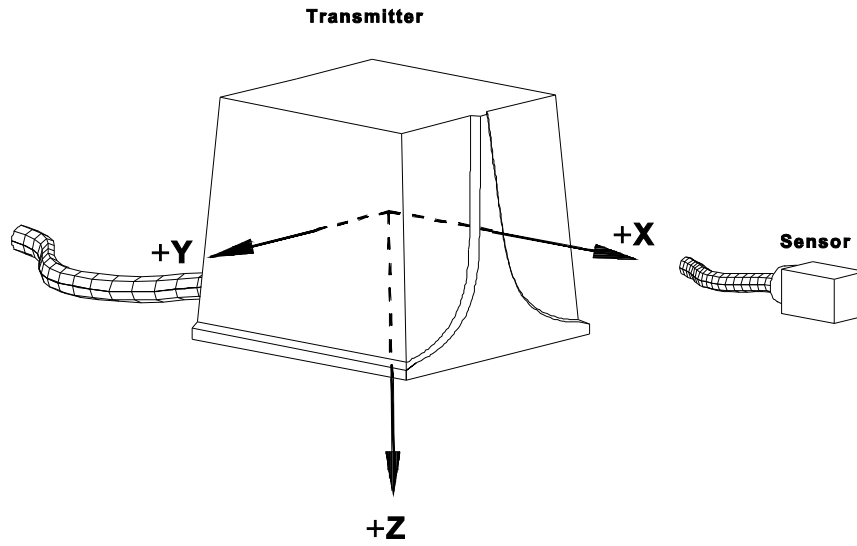


Figure 3-1 Measurement Reference Frame (Mid-Range Transmitter)

## Requirements

**Header:** Declared in ATC3DG.h

**Library:** Use ATC3DG.lib (ATC3DG64.lib for 64 bit applications)

## 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

**Header:** Declared in ATC3DG.h

**Library:** Use ATC3DG.lib (ATC3DG64.lib for 64 bit applications)

## 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**:

COS(Y)*COS(Z)	COS(Y)*SIN(Z)	-SIN(Y)
-(COS(X)*SIN(Z))	(COS(X)*COS(Z))	SIN(X)*COS(Y)
+(SIN(X)*SIN(Y)*COS(Z))	+(SIN(X)*SIN(Y)*SIN(Z))	
(SIN(X)*SIN(Z))	-(SIN(X)*COS(Z))	(COS(X)*COS(Y)
+(COS(X)*SIN(Y)*COS(Z))	+(COS(X)*SIN(Y)*SIN(Z))	

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

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

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

**Header:** Declared in ATC3DG.h

**Library:** Use ATC3DG.lib (ATC3DG64.lib for 64 bit applications)

## 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 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.

### Requirements

**Header:** Declared in ATC3DG.h

**Library:** Use ATC3DG.lib (ATC3DG64.lib for 64 bit applications)

## 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

**Header:** Declared in ATC3DG.h

**Library:** Use ATC3DG.lib (ATC3DG64.lib for 64 bit applications)

**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

**Header:** Declared in ATC3DG.h

**Library:** Use ATC3DG.lib (ATC3DG64.lib for 64 bit applications)

**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

**Header:** Declared in ATC3DG.h

**Library:** Use ATC3DG.lib (ATC3DG64.lib for 64 bit applications)

**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. The positive X, Y and Z directions are shown below.

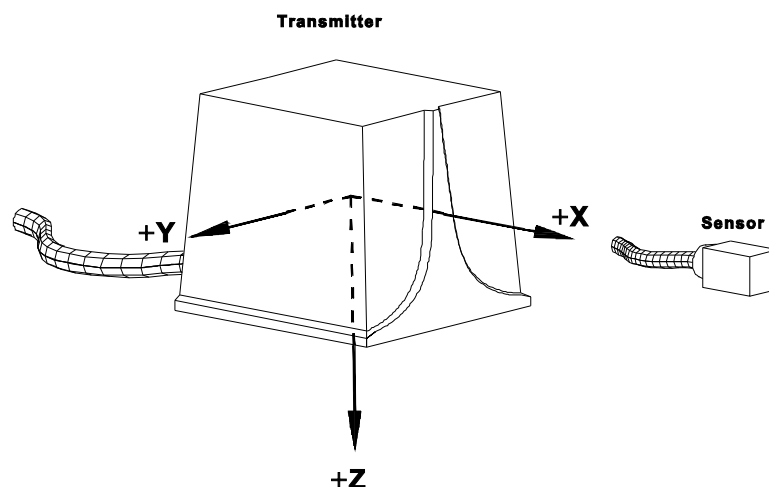


Figure 3-2 Measurement Reference Frame (Mid-Range Transmitter)

## Requirements

**Header:** Declared in ATC3DG.h

**Library:** Use ATC3DG.lib (ATC3DG64.lib for 64 bit applications)

## 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

**Header:** Declared in ATC3DG.h

**Library:** Use ATC3DG.lib (ATC3DG64.lib for 64 bit applications)

## 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**:

$\cos(Y)\cos(Z)$	$\cos(Y)\sin(Z)$	$-\sin(Y)$
$-(\cos(X)\sin(Z))$ $+(\sin(X)\sin(Y)\cos(Z))$	$(\cos(X)\cos(Z))$ $+(\sin(X)\sin(Y)\sin(Z))$	$\sin(X)\cos(Y)$
$(\sin(X)\sin(Z))$ $+(\cos(X)\sin(Y)\cos(Z))$	$-(\sin(X)\cos(Z))$ $+(\cos(X)\sin(Y)\sin(Z))$	$(\cos(X)\cos(Y))$

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

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

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

Element scaling is  $+0.99996 = 7FFF$  Hex,  $0 = 0$  Hex, and  $-1.0 = 8000$  Hex.

Matrix information is 0 when sensor saturation occurs.

## Requirements

**Header:** Declared in ATC3DG.h

**Library:** Use ATC3DG.lib (ATC3DG64.lib for 64 bit applications)



## 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 scaler component, have been extracted from the MATRIX output using the algorithm described in "Quaternion from Rotation Matrix" by Stanley W. Sheppard, Journal of Guidance and Control, Vol. 1, May-June 1978, pp. 223-4.

### Requirements

**Header:** Declared in ATC3DG.h

**Library:** Use ATC3DG.lib (ATC3DG64.lib for 64 bit applications)

## 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

**Header:** Declared in ATC3DG.h

**Library:** Use ATC3DG.lib (ATC3DG64.lib for 64 bit applications)

**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

**Header:** Declared in ATC3DG.h

**Library:** Use ATC3DG.lib (ATC3DG64.lib for 64 bit applications)

**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

**Header:** Declared in ATC3DG.h

**Library:** Use ATC3DG.lib (ATC3DG64.lib for 64 bit applications)

**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

**Header:** Declared in ATC3DG.h

**Library:** Use ATC3DG.lib (ATC3DG64.lib for 64 bit applications)

**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

**Header:** Declared in ATC3DG.h

**Library:** Use ATC3DG.lib (ATC3DG64.lib for 64 bit applications)

**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

**Header:** Declared in ATC3DG.h

**Library:** Use ATC3DG.lib (ATC3DG64.lib for 64 bit applications)

**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

**Header:** Declared in ATC3DG.h

**Library:** Use ATC3DG.lib (ATC3DG64.lib for 64 bit applications)

**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

**Header:** Declared in `ATC3DG.h`

**Library:** Use `ATC3DG.lib` (`ATC3DG64.lib` for 64 bit applications)

**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

**Header:** Declared in ATC3DG.h

**Library:** Use ATC3DG.lib (ATC3DG64.lib for 64 bit applications)

**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

**Header:** Declared in ATC3DG.h

**Library:** Use ATC3DG.lib (ATC3DG64.lib for 64 bit applications)

**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

**Header:** Declared in ATC3DG.h

**Library:** Use ATC3DG.lib (ATC3DG64.lib for 64 bit applications)

**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

**Header:** Declared in ATC3DG.h

**Library:** Use ATC3DG.lib (ATC3DG64.lib for 64 bit applications)

**See Also**

[DOUBLE\\_ANGLES\\_RECORD](#)



## DOUBLE\_MATRIX\_TIME\_Q\_RECORD

The **DOUBLE\_MATRIX\_TIME\_Q\_RECORD** structure contains a 3x3 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

**Header:** Declared in ATC3DG.h

**Library:** Use ATC3DG.lib (ATC3DG64.lib for 64 bit applications)

**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

**Header:** Declared in ATC3DG.h

**Library:** Use ATC3DG.lib (ATC3DG64.lib for 64 bit applications)

**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

**Header:** Declared in `ATC3DG.h`

**Library:** Use `ATC3DG.lib` (`ATC3DG64.lib` for 64 bit applications)

**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

**Header:** Declared in ATC3DG.h

**Library:** Use ATC3DG.lib (ATC3DG64.lib for 64 bit applications)

**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

**Header:** Declared in ATC3DG.h

**Library:** Use ATC3DG.lib (ATC3DG64.lib for 64 bit applications)

**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

**Header:** Declared in ATC3DG.h

**Library:** Use ATC3DG.lib (ATC3DG64.lib for 64 bit applications)

**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

**Header:** Declared in ATC3DG.h

**Library:** Use ATC3DG.lib (ATC3DG64.lib for 64 bit applications)

### 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**

**Header:** Declared in ATC3DG.h

**Library:** Use ATC3DG.lib (ATC3DG64.lib for 64 bit applications)

**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

**Header:** Declared in ATC3DG.h

**Library:** Use ATC3DG.lib (ATC3DG64.lib for 64 bit applications)

**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.

#### **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

**Header:** Declared in ATC3DG.h

**Library:** Use ATC3DG.lib (ATC3DG64.lib for 64 bit applications)

**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 the user guide 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

**Header:** Declared in `ATC3DG.h`

**Library:** Use `ATC3DG.lib` (`ATC3DG64.lib` for 64 bit applications)

**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 the user guide 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**

**Header:** Declared in ATC3DG.h

**Library:** Use ATC3DG.lib (ATC3DG64.lib for 64 bit applications)

**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 the user guide 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

**Header:** Declared in ATC3DG.h

**Library:** Use ATC3DG.lib (ATC3DG64.lib for 64 bit applications)

**See Also**

[DOUBLE\\_POSITION\\_RECORD](#),  
[DOUBLE\\_QUATERNIONS\\_RECORD](#)



## 4 3D Guidance Enumeration Types

The following enumeration types are used with the 3DGuidance 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](#)

[COMMUNICATIONS\\_MEDIA\\_TYPES](#)

[PORT\\_CONFIGURATION\\_TYPE](#)

[AUXILIARY\\_PORT\\_TYPE](#)

## 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,
BIRD_ERROR_MAXIMUM_VALUE
};

```

Enumerator Value	Meaning
BIRD_ERROR_SUCCESS=0	No errors occurred. Call completed successfully
BIRD_ERROR_PCB_HARDWARE_FAILURE	The 3DGuidance 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	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_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 3DGuidance hardware was not 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. *pciBIRD error only
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 3DGuidance 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. *pci-BIRD error only
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 3DGuidance 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	3DGuidance 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 3DGuidance hardware and system has not been initialized yet. The <a href="#">InitializeBIRDSsystem</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 3DGuidance hardware is incompatible with this version of the API dll. Verify 3DGuidance model installed.
BIRD_ERROR_NO_TRANSMITTER_ATTACHED	A request was made to do one of the following: Turn off the currently running transmitter and there is no transmitter attached to the system Turn on the transmitter with the selected ID and there is no transmitter attached at that ID.
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	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 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 – InitializeBIRDSystem() 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 or it may be caused by 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	The position and orientation algorithm used for the non-dipole transmitters has not correctly initialized.
BIRD_ERROR_LOST_CONNECTION	USB connection to the tracker has been lost. This may be due to usb cable removal, or removal of power to the electronics unit.
BIRD_ERROR_INVALID_CONFIGURATION	The system has queried the Multi-Unit ID settings for each of the attached electronics units, and found that the current configuration is not supported. Valid MUS configurations include: 0,1 (8 sensors), 0,1,2 (12 sensors), 0,1,2,3 (16 sensors). Further details can be found in the MUS Installation Guide Addendum.

BIRD_ERROR_TRANSMITTER_RUNNING	A request to read or write to the VPD component memory storage area was made while the transmitter was still running. De-select (turn off) the transmitter prior to a read/write to VPD memory.
BIRD_ERROR_MAXIMUM_VALUE	Final error code place holder

## Requirements

**Header:** Declared in ATC3DG.h

**Library:** Use ATC3DG.lib (ATC3DG64.lib for 64 bit applications)

## 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,
MODEL_STRING_RX,
PART_NUMBER_RX
MODEL_STRING_PREAMP,
PART_NUMBER_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 in (Figure 4-2), and are measured with respect to rotations about the physical X, Y and Z axes of the sensor (Figure 4-3). 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>

### DATA\_FORMAT

See the Data Format Structures section for details

### ANGLE\_ALIGN

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 in [Figure 4-1](#), and are measured with respect to rotations about the physical X, Y and Z axes of the sensor ([Figure 4-2](#)). 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.

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.

ANGLE\_ALIGN parameters are not meaningful for 5DOF sensors. ANGLE\_ALIGN parameters applied to a 5DOF sensor will be ignored.

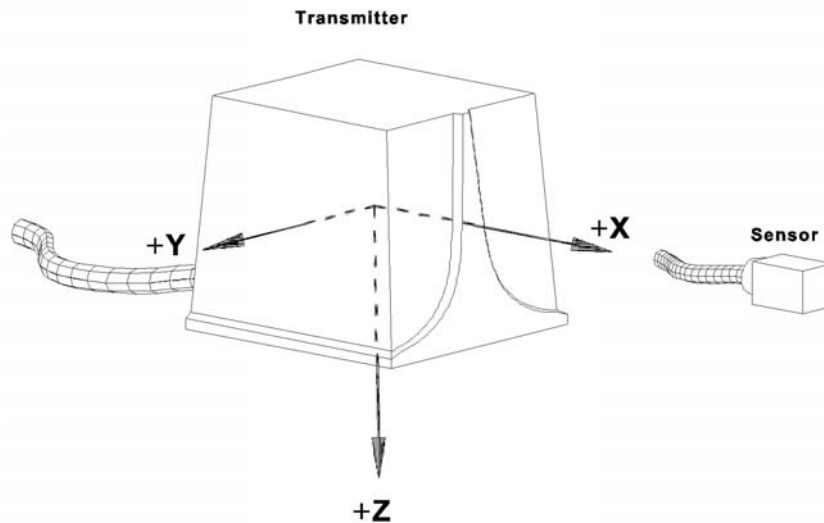


Figure 4-1 Measurement Reference Frame (Standard Transmitter)

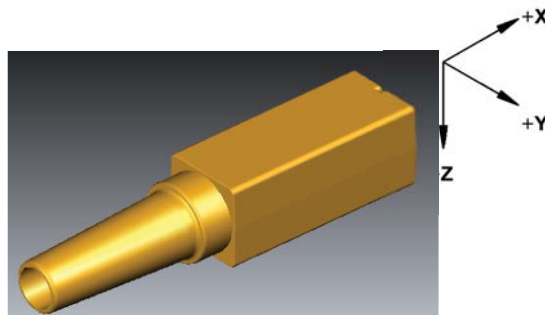


Figure 4-2 Receiver Zero Orientation (8mm Sensor)

## HEMISPHERE

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 in the ANGLES data formats, or in the rotation matrix as output in the MATRIX formats.

The HEMISPHERE parameter is used to tell the Tracker in which hemisphere, centred about 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.

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 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.

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.

#### **FILTER\_AC\_WIDE\_NOTCH**

The AC WIDE notch filter refers to a eight tap FIR notch filter that is applied to the sensor data to eliminate sinusoidal signals with a frequency between 30Hz and 72Hz. 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**

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.

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

### `FILTER_ALPHA_PARAMETERS`

To modify the filter characteristics, configure the elements of the structure [ADAPTIVE\\_PARAMETERS](#).

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 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`.

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.

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.

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.

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.

### `FILTER_LARGE_CHANGE`

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.

### `QUALITY`

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 QUALITYError 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 QUALITYError number indicates no or minimal position and angle errors depending on how sensitive you have set the error indicator. A large QUALITYError number indicates maximum error for the sensitivity level selected.

Users of the QUALITYError 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 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 decrease, indicating less error, and then finally start increasing again as the new metal causes more distortion.

---

**Note** You must evaluate your application for suitability of this metal detector.

---

Because the TRACKER is used in many different applications and environments, the QUALITYError 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 [SetSensorParameter](#) allows the user to set a QUALITY Sensitivity setting that is appropriate to their application.

The QUALITYError 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 **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 [QUALITY\\_PARAMETERS](#) structure (manipulated through the [SetSensorParameter](#) 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 QUALITYError values.

The QUALITYError number that is output is computed from the following equation:

$$\text{QUALITYError} = \text{Sensitivity} \times (\text{ErrorSYSTEM} - (\text{Offset} + \text{Slope} \times \text{Range}))$$

Where Range is the distance between the transmitter and sensor.

**Sensitivity** The user supplies a Sensitivity value based on how little or how much he or she wants the QUALITYError value to reflect errors.

**Offset** 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.

**Slope** 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 (ie 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.

**Alpha** 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 -> 0 Users should think of it as a signed fractional value with a range of 0.9999 -> 0 (negative numbers not allowed). A zero value is an infinite amount of 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\_OFFSET

Normally the position outputs from the 3DGuidanceTM represent the x, y, z position of the centre 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 centre of the sensor.

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 centre to the desired position on the tracked object. (These values must be supplied in inches.)

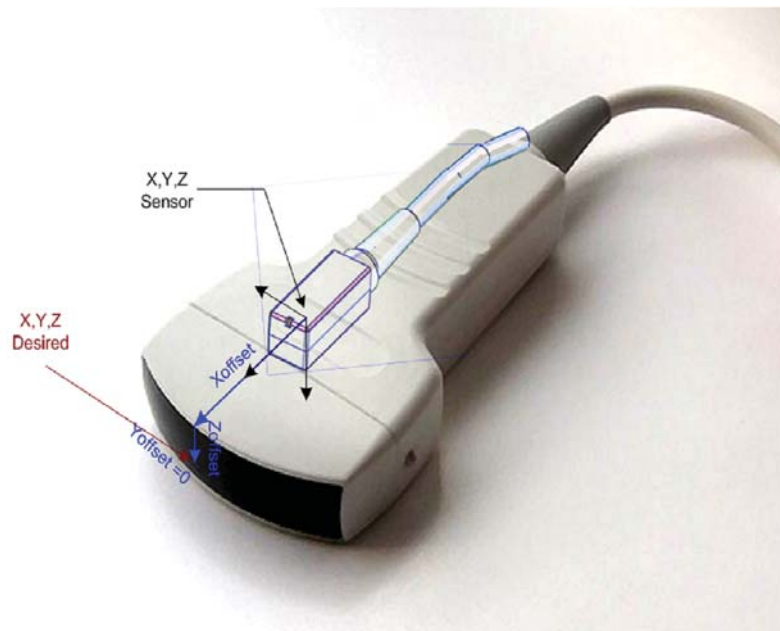


Figure 4-3 Sensor Offset

### VITAL\_PRODUCT\_DATA\_RX

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 [VPD\\_COMMAND\\_PARAMETER](#) which contains the address of the target byte and, in the case of the write command ([SetSensorParameter](#)) the value of the byte to be written.

In the case of the read command ([GetSystemParameter](#)), 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.

---

### VITAL\_PRODUCT\_DATA\_PREAMP

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 [VPD\\_COMMAND\\_PARAMETER](#) which contains the address of the target byte and, in the case of the write command ([SetSystemParameter](#)), the value of the byte to be written. In the case of the read command ([GetSensorParameter](#)), 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.

---

### MODEL\_STRING\_RX

Returns the model string of the sensor in an 11 byte NULL terminated character string.

### PART\_NUMBER\_RX

Returns the part number of the sensor in an 16 byte NULL terminated character string.

### MODEL\_STRING\_PREAMP

Returns the model string of the preamp in an 11 byte NULL terminated character string. (driveBAY and trakSTAR units do not have preamps, therefore this is an invalid parameter for those systems)

### PART\_NUMBER\_PREAMP

Returns the part number of the preamp in an 16 byte NULL terminated character string. (driveBAY and trakSTAR units do not have preamps, therefore this is an invalid parameter for those systems)

## Requirements

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

**Header:** Declared in ATC3DG.h

**Library:** Use ATC3DG.lib (ATC3DG64.lib for 64 bit applications)

## 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

**Header:** Declared in `ATC3DG.h`

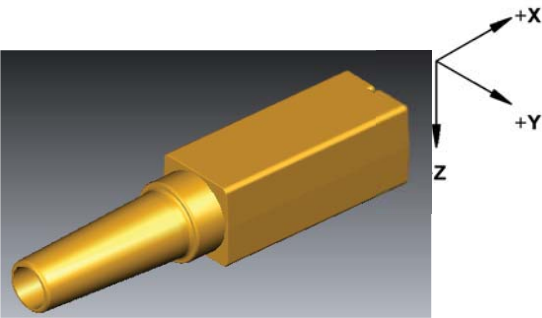
**Library:** Use `ATC3DG.lib` (`ATC3DG64.lib` for 64 bit applications)

## 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,
    MODEL_STRING_TX,
    PART_NUMBER_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 <a href="#">Figure 4-4</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="555 1171 1380 1690" data-label="Image"> <p>The diagram illustrates the measurement reference frame for a standard transmitter. It features a 3D coordinate system with three axes: +X (pointing to the right), +Y (pointing to the left), and +Z (pointing downwards). A transmitter is depicted as a 3D object with its physical axes aligned with this coordinate system. A cable connects the transmitter to a sensor, which is shown as a small rectangular block.</p> </div> <p>Figure 4-4 Measurement Reference Frame (Standard Transmitter)</p>

	<div></div> <p>Figure 4-5 Receiver Zero Orientation (8mm Sensor)</p>
XYZ_REFERENCE_FRAME	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.
VITAL_PRODUCT_DATA- _TX	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_PARAMETER</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. Note: Reading or writing VPD is not allowed when the transmitter is running.
MODEL_STRING_TX	Returns the model string of the transmitter in an 11 byte NULL terminated character string.
PART_NUMBER_TX	Returns the part number of the transmitter in an 16 byte NULL terminated character string.

Requirements

**Header:** Declared in ATC3DG.h

**Library:** Use ATC3DG.lib (ATC3DG64.lib for 64 bit applications)



## 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,
    MODEL_STRING_PCB,
    PART_NUMBER_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_REVISIONS</a> structure.
POST_ERROR_PCB	Used to examine results of the POST (Power ON Self Test). See Parameter Structure <a href="#">POST_ERROR_PARAMETER</a>
DIAGNOSTIC_TEST_PCB	Used to execute diagnostic tests. May also be used to acquire information on available diagnostic tests. See Parameter structure: <a href="#">DIAGNOSTIC_TEST_PARAMETERS</a>
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 112 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_PARAMETER</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.
MODEL_STRING_PCB	Returns the model string of the board in an 11 byte NULL terminated character string.
PART_NUMBER_PCB	Returns the part number of the board in an 16 byte NULL terminated character string.

### Requirements

**Header:** Declared in ATC3DG.h

**Library:** Use ATC3DG.lib (ATC3DG64.lib for 64 bit applications)

## 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,
COMMUNICATIONS_MEDIA,
LOGGING,
RESET,
AUTOCONFIG,
END_OF_LIST
};
```

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	Inform 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 AGC_MODE_TYPE.
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<rate<255.0
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 are three valid ranges, 36.0 inches, 72.0 inches and 144.0 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*	Used to read or write to individual bytes in the Vital Product Data (VPD) storage area on the main board. The VPD contains 512 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_PARAMETER</a> which contains the address of the target byte and in the case of the write command (SetSystemParameter) the value of the byte to be written. In the case of the read command (GetSystemParameter), the value of the byte read from the VPD is placed in the value location in the structure.
POST_ERROR*	Used to examine results of the POST (Power ON Self Test). See Parameter Structure <a href="#">POST_ERROR_PARAMETER</a>
DIAGNOSTIC_TEST*	Used to execute diagnostic tests. May also be used to acquire information on available diagnostic tests. See Parameter structure: <a href="#">DIAGNOSTIC_TEST_PARAMETERS</a>
REPORT_RATE	Used to set the decimation rate for streaming data records. The report rate is a single byte value 1 to 127, 0 is not valid.
COMMUNICATIONS_MEDIA	Used to configure the communications media for interaction with the tracking device. See Parameter structure: <a href="#">COMMUNICATIONS_MEDIA_PARAMETERS</a> . NOTE: This parameter can be used prior to a InitBIRDSystem() function call.
LOGGING	Used to enable/disable logging of the communications traffic between the API and the tracking device. Useful for reporting and trouble-shooting errors with the Ascension tracking system. NOTE: This parameter can be used prior to a InitBIRDSystem() function call.
RESET	Used to enable/disable the automatic tracking device reset on a InitBIRDSystem API call. Disabling reset will make the InitBIRDSystem function call perform much faster and may be useful in the development phase of a project, but this should be used with caution, as a reset places the tracking device in a known state and disabling it may cause undetermined side effects. NOTE: This parameter can be used prior to a InitBIRDSystem() function call.
AUTOCONFIG	Used to specify the number of sensors to configure the tracking device. This parameter is useful for systems that use multiple 5DOF sensors in lieu of a single 6DOF sensor. 4 and 12 are the valid values for this parameter. NOTE: This parameter can be used prior to a InitBIRDSystem() function call.
END_OF_LIST	Final system parameter type place holder

Note \* Use of the VITAL\_PRODUCT\_DATA, POST\_ERROR, and DIAGNOSTIC\_TEST system parameters with Multi-Unit System (MUS) configurations will only yield information from the first unit (MUS ID=0). To access this information/functionality across all units, see [GetBoardParameter](#).

## Requirements

**Header:** Declared in ATC3DG.h

**Library:** Use ATC3DG.lib (ATC3DG64.lib for 64 bit applications)

## 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

**Header:** Declared in ATC3DG.h

**Library:** Use ATC3DG.lib (ATC3DG64.lib for 64 bit applications)

## 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 Note: Transmitter AGC is not currently available.	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

**Header:** Declared in ATC3DG.h

**Library:** Use ATC3DG.lib (ATC3DG64.lib for 64 bit applications)

## 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,
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

Header: Declared in ATC3DG.h

Library: Use ATC3DG.lib (ATC3DG64.lib for 64 bit applications)

## 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`.

---

**Note** The `BOARD_TYPES` enumerated type WILL NOT support future boards. The `MODEL_STRING` and `PART_NUMBER` parameter types have replaced this functionality.

---

```
enum BOARD_TYPES{
    ATC3DG_MEDSAFE
    PCIBIRD_STD1
    PCIBIRD_STD2
    PCIBIRD_8mm1
    PCIBIRD_8mm2
    PCIBIRD_2mm1
    PCIBIRD_2mm2
    PCIBIRD_FLAT
    PCIBIRD_FLAT_MICRO1
    PCIBIRD_FLAT_MICRO2
    PCIBIRD_DSP4
    PCIBIRD_UNKNOWN
    ATC3DG_BB
};
```

Enumerator Value	Meaning
ATC3DG_MEDSAFE	medSAFE
PCIBIRD_STD1	Synchronized PCIBird – Single Standard Sensor
PCIBIRD_STD2	Synchronized PCIBird – Dual Standard Sensor
PCIBIRD_8mm1	Synchronized PCIBird – Single 8mm Sensor
PCIBIRD_8mm2	Synchronized PCIBird – Dual 8mm Sensor
PCIBIRD_2mm1	Single 2mm sensor - microsensor
PCIBIRD_2mm2	Dual 2mm sensor -microsensor
PCIBIRD_FLAT	Flat transmitter, 8mm
PCIBIRD_FLAT_MICRO1	Flat transmitter, single TEM sensor (all types)
PCIBIRD_FLAT_MICRO2	Flat transmitter, dual TEM sensor (all types)
PCIBIRD_DSP4	Standalone, DSP, 4 sensor
PCIBIRD_UNKNOWN	Default
ATC3DG_BB	DriveBAY/trakSTAR (formerly bayBIRD)

## Requirements

**Header:** Declared in `ATC3DG.h`

**Library:** Use `ATC3DG.lib` (`ATC3DG64.lib` for 64 bit applications)



## 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**.

**Note** The **DEVICE\_TYPES** enumerated type WILL NOT support future transmitters and sensors. The **MODEL\_STRING** and **PART\_NUMBER** parameter types have replaced this functionality.

```
enum DEVICE_TYPES{
    STANDARD_SENSOR,
    TYPE_800_SENSOR,
    STANDARD_TRANSMITTER,
    MINIBIRD_TRANSMITTER,
    SMALL_TRANSMITTER,
    TYPE_500_SENSOR,
    TYPE_180_SENSOR,
    TYPE_130_SENSOR,
    TYPE_TEM_SENSOR,
    UNKNOWN_SENSOR,
    UNKNOWN_TRANSMITTER,
    TYPE_800_BB_SENSOR,
    TYPE_800_BB_STD_TRANSMITTER,
    TYPE_800_BB_SMALL_TRANSMITTER,
    TYPE_090_BB_SENSOR
};
```

Enumerator Value	Meaning
STANDARD_SENSOR	Standard Flock sensor with miniDIN connector
TYPE_800_SENSOR	8mm sensor with miniDIN connector (miniBIRDII sensor)
STANDARD_TRANSMITTER	Standard Flock transmitter
MINIBIRD_TRANSMITTER	Standard MiniBird transmitter
SMALL_TRANSMITTER	Compact transmitter
TYPE_500_SENSOR	5mm sensor with miniDIN connector
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
TYPE_800_BB_SENSOR	DriveBAY/traksTAR sensor (bayBIRD)
TYPE_800_BB_STD_TRANSMITTER	DriveBYA/trakSTAR mid-range TX
TYPE_800_BB_SMALL_TRANSMITTER	DriveBYA/trakSTAR short-range TX
TYPE_090_BB_SENSOR	DriveBAY/traksTAR sensor (bayBIRD)

## Requirements

**Header:** Declared in ATC3DG.h

**Library:** Use ATC3DG.lib (ATC3DG64.lib for 64 bit applications)

## COMMUNICATIONS\_MEDIA\_TYPES

A value of the **COMMUNICATIONS\_MEDIA\_TYPES** enumeration type is set or returned in the *mediaType* field of the **COMMUNICATIONS\_MEDIA\_PARAMETERS** structure, which is used with a call to either `GetSystemParameter` or `SetSystemParameter`.

**Note** Once communication over a given media type has been initiated after power-up, the tracker will continue to expect communication over this media regardless of the setting of this enumerated type. The tracker will remain in this communication state until power to the unit has been reset, at which time the media type parameter may once again be utilized.

```
enum COMMUNICATIONS_MEDIA_TYPES{  
    USB,  
    RS232,  
    TCPIP,  
};
```

Enumerator Value	Meaning
USB	Selects the USB device driver that was selected during installation.
RS232	Uses the Window's COM port interface. IMPORTANT: There are some limitations on the use of this interface, the <code>GetSynchronous</code> API is not supported with this interface and <code>GetAsynchronous</code> API cannot be used with the <code>ALL_SENSORS</code> parameter.
TCPIP	Uses TCP/IP

### Requirements

**Header:** Declared in `ATC3DG.h`

**Library:** Use `ATC3DG.lib` (`ATC3DG64.lib` for 64 bit applications)

## PORT\_CONFIGURATION\_TYPE

The PORT\_CONFIGURATION\_TYPE enumeration type defines values that are used when getting the PORT\_CONFIGURATION with a GetSensorParameter call.

```
enum PORT_CONFIGURATION_TYPE {  
    DOF_NOT_ACTIVE,  
    DOF_6_XYZ_AER,  
    DOF_5_XYZ_AE,  
};
```

Enumerator Value	Meaning
DOF_NOT_ACTIVE	This sensor does not logically exist in the current configuration.
DOF_6_XYZ_AER	The sensor is a 6DOF sensor.
DOF_5_XYZ_AE	The sensor is a 5DOF sensor.

### Requirements

**Header:** Declared in ATC3DG.h

**Library:** Use ATC3DG.lib (ATC3DG64.lib for 64 bit applications)

## AUXILIARY\_PORT\_TYPE

The AUXILIARY\_PORT\_TYPE enumeration type defines values that are used when getting or setting the AUXILIARY\_PORT with a GetSystemParameter or SetSystemParameter call.

```
enum AUXILIARY_PORT_TYPE
{
    AUX_PORT_NOT_SUPPORTED,
    AUX_PORT_NOT_SELECTED,
    AUX_PORT_SELECTED,
};
```

Enumerator Value	Meaning
AUX_PORT_NOT_SUPPORTED	There is no auxiliary port associated with this sensor port.
AUX_PORT_NOT_SELECTED	The auxiliary port is not selected.
AUX_PORT_SELECTED	The auxiliary port is selected.

### Requirements

**Header:** Declared in ATC3DG.h

**Library:** Use ATC3DG.lib (ATC3DG64.lib for 64 bit applications)

## 5 3D Guidance API Status/Error Bit Definitions

The following bit definitions are used with the 3DGuidance tracker

[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 = 3DGuidance 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 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="#">InitializeBIRDSsystem</a>	x	x	x	x
7	NO_TRANSMITTER_RUNNING	An attempt was made to call <a href="#">GetAsynchronousRecord</a> when no transmitter was running.	x	x	x	
8	BAD_12V	N/A for the 3DG systems				
9	CPU_TIMEOUT	N/A for the 3DG systems				
10	INVALID_DEVICE	N/A for the 3DG systems				
11	NO_TRANSMITTER_ATTACHED	A transmitter is not attached to the tracking system.	x	x	x	x
12	OUT_OF_MOTIONBOX	The sensor has exceeded the maximum range and the position has been clamped to the maximum range			X	
13	ALGORITHM_INITIALIZING	The sensor has not acquired enough raw magnetic data to compute an accurate P&O solution.			x	
14 - 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.



## 6 3D Guidance Initialization Files

The Initialization File is used to set a 3DGuidance tracker to a predetermined state.

### 6.1 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 3DGuidance 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 3DGuidance system:

[System]

[ErrorHandling] Reserved for future enhancements)

[Sensorx] Where *x* is replaced with a decimal number representing the *id* of the sensor.)

[Transmitterx] 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 3DGuidance tracker hardware.

```
[System]
NumberOfBoards=number-boards
TransmitterIDRunning=Tx-ID
MeasurementRate=sample-rate
Metric=metric-switch
PowerLineFrequency=power
AGCMode=mode
MaximumRange=range
number-boards
```

This parameter is a decimal number and represents the number of 3DGuidance units connected to the PC. This number must match the current number of units 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 unsigned floating point value describing the measurement rate in Hz.

#### *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. The only valid values are 36 and 72 (inches).

The following example shows a typical **System** section:

```
[System]
NumberOfBoards=1
TransmitterIDRunning=0
MeasurementRate=103.3
Metric=YES
PowerLineFrequency=60
AGCMode=SENSOR_AGC_ONLY
MaximumRange=36
```

### [Sensor*x*]

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 the DATA\_FORMAT\_TYPE enumerated constant listed in the ATC3DG.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 ATC3DG.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 –127 and +127. (Default is 0) 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 2) and the alpha should have a value between 0 and 127. (The default is 12)

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
```



## 7 Ascension RS232 Interface Reference

This chapter details the method for communicating with 3D Guidance systems directly, using the RS232 interface protocol and command set.

### 7.1 RS232 Signal Description

A pinout and signal description of the RS-232C interface is detailed below.

**Note** The system requires connections only to pins 2, 3 and 5 of the 9-pin interface connector.

Table 7-1 9 Pin RS-232C connector pinout

Pin	RS232 Signal	Direction	Description
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** The above 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.

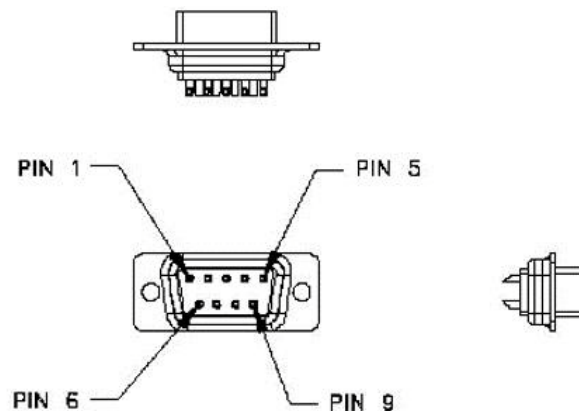


Figure 7-1 Rear View - 9 Pin D-Sub Connector

#### Using the 'reset on CTS' feature

The system 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 technical support for details.

---

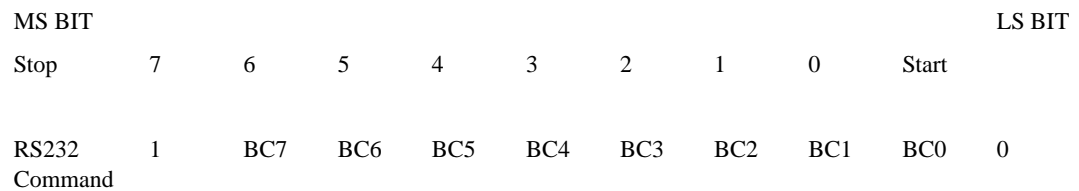
**Note** If you are running applications in a Windows® environment and want to enable this feature, you must ensure that command of this line (and of the serial port) is clearly asserted.

---

## 7.2 RS232 Commands

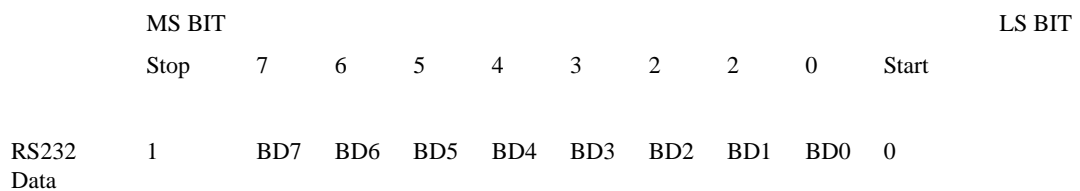
Each RS232 command consists of a single **command byte** followed by **N command data bytes**, where N depends upon the command. A command is an 8-bit value which the host transmits to the Tracker.

The RS232 **command format** is as follows:



where BC7-BC0 is the 8-bit command value (see [“RS232 Command Reference” on page 184](#)) 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:



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. For details on command use, see [“RS232 Command Reference” on page 184](#).

---

**Note** For a listing of valid system parameters to use with the CHANGE or EXAMINE VALUE commands, see [“CHANGE VALUE” on page 191](#).

---

Table 7-2 RS232 Command Summary

Command	Description
ANGLES	Data record contains 3 rotation angles.
ANGLE ALIGN	Aligns sensor to reference direction.
BORESIGHT (Not currently implemented)	Aligns sensor to the reference frame
BORESIGHT REMOVE (Not currently implemented)	Remove the sensor BORESIGHT
BUTTON MODE	Send to append the button (contact closure/switch) information to the data record
BUTTON READ	Send to read the state of the Button when Button mode is not enabled.
CHANGE VALUE	Changes the value of a selected Tracker system parameter.
EXAMINE VALUE	Reads and examines a selected Tracker system parameter.
MATRIX	Data record contains 9-element rotation matrix.
OFFSET	Configures positional outputs from the Tracker to specify a location that is offset from the centre of the Sensor.
POINT	One data record is output from the system for each B command issued.
POSITION	Data record contains X, Y, Z position of sensor.
POSITION/ANGLES	Data record contains POSITION and ANGLES.
POSITION/MATRIX	Data record contains POSITION and MATRIX.
POSITION/QUATERNION	Data record contains POSITION and QUATERNION.
QUATERNION	Data record contains QUATERNIONS.
REFERENCE FRAME	Defines new measurement reference frame.
REPORT RATE	Measurement rate divisor command. Reduces the number of records output during STREAM mode.
RESET	Performs a system reset
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 sensor. If GROUP mode is enabled then data records are output continuously from all sensors connected to the system unit.
STREAM STOP	Stops any data output that was started with the STREAM.

## Command Utilization

The host may tell what type of data to send when a data request is issued. The desired type of data is indicated by sending one of the following data record commands: ANGLES, MATRIX, POSITION, QUATERNION, POSITION/ANGLES, POSITION/MATRIX or POSITION/QUATERNION. These commands do not cause 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 tracker, use the REPORT RATE command. All commands can be issued in any order and at any time to change the system output characteristics.



The following is a hypothetical command sequence, issued after power-up, which illustrates the use of some of the commands.

Table 7-3 Hypothetical Command Sequence

Command	Action
<b>RUN</b>	Turn on the transmitter (needed if system configured to sleep on reset)
<b>ANGLES</b>	Output records will contain angles only.
<b>POINT</b>	Tracker outputs ANGLES data record.
<b>STREAM</b>	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.
<b>POINT</b>	An ANGLE data record is output and the streaming is stopped.

## Response Format

Two types of binary data are returned from the system:

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 '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 two least significant bits (bits B1 and B0) are not used by the tracker. 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	
7	6	5	4	3	2	1	0		Word #

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:

1. Shifts each data word right one bit.

MS	LS
00001000	10010001
00011001	10100010
00101010	10110011

2. Breaks each word into MSByte/LSByte pairs.

```

LSByte pairs
10010001 LS
00001000 MS
10100010 LS
00011001 MS
10110011 LS
00101010 MS

```

3. Shifts each LSByte right one more bit (Marks with “1” if first byte).

4. Transmits all bytes in stream bit (Marks with “1” if first byte

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”.)

01001000	LS
00001000	MS
01010001	LS
00011001	MS
01011001	LS
00101010	MS

2. Shifts each LSByte left one bit.

10010000	LS
00001000	MS
10100010	LS
00011001	MS
10110010	LS
00101010	MS

3. Combines each MSByte/LSByte pair into data words.
4. Shifts each word left one more bit, giving the correct original binary value. (The data words do not use the two least significant bits and can be ignored.).

MS	LS	MS	LS
00001000	10010000	00010001	00100000
00011001	10100010	00110011	01000100
00101010	10110010	01010101	01100100

## 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 system 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				LSB				
7	6	5	4	9	8	7	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 behaviour 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.

**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 (as shown in the user guide), 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 which differs from that of the actual sensor.

Note The ANGLE ALIGN command only affects the computation of orientation - it has no effect on position.

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 is ignored for 5DOF sensors.

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:

MSB							LSB	
7	6	5	4	3	2	1	0	Byte #
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

---

**Note** The BORESIGHT command is not currently implemented.

---

Sending the single byte BORESIGHT command to the 3DGuidance 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 BORESIGHTed 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 ALIGNs 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

---

**Note** The BORESIGHT REMOVE command is not currently implemented.

---

Sending the single byte BORESIGHT REMOVE command to the system 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 ALIGNED 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 (attached to the 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 rear panel of the 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 system 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 system 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.

		CHANGEable	Bytes Send	EXAMINE Bytes		Description
Parameter Dec	Hex			Send	Receive	
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
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

			CHANGE	EXAMINE		
				Bytes		
Parameter		CHANGEable	Bytes Send	Send	Receive	Description
20	14	Yes	3	2	1	Filter line frequency
22	16	Yes	4	2	2	Change/Examine Hemisphere
23	17	Yes	8	2	6	Change/Examine Angle Align2
24	18	Yes	8	2	6	Change/Examine 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 POSEServer 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 POSEServer Firmware Revision

The **CHANGE VALUE** command must be issued to The Tracker in the following N-byte sequence:

MSB				LSB				Byte #
7	6	5	4	9	8	7	0	
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 PARAMETERdata value, content and scaling depend 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:

---

**Note** To retrieve the error code(s) indicated by this 'flag' (B13), send '10' as the parameter. See ["ERROR CODE" on page 197](#).

---

Bit #	Description
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
B5	1 if The Tracker is in SLEEP mode. (Opposite of B12) 0 if The Tracker is in RUN mode
B4-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.)

## SOFTWARE REVISION NUMBER

When PARAMETERnumber = 1 during EXAMINE, The Tracker returns the two byte revision number of the software located in The Tracker's 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 01 and 03 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 3DGuidance™ 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. The average user of the Tracker should not have 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 #	Description
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 an eight tap FIR notch filter that is applied to the sensor data to eliminate sinusoidal signals with a frequency between 30 and 72 Hz. 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 PARAMETER data 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 PARAMETER number = 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::

Range (inches)	ALPHA MIN (Decimal)
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

To CHANGE ALPHA\_MIN, send The Tracker seven words of PARAMETER data 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.



## 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 to 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 The Tracker's 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:

Std. Range Xmtr (inches)	Vm (Integer)
0-17	2

17-22	4
22-27	4
27-34	4
34-42	4
42-54	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:

Std. Range Xmtr (Range)	ALPHA_MAX (fractional)
0-17	0.9 = 07333 Hex.
17-22	0.9
22-27	0.9
27-34	0.9
34-42	0.9
42-54	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 The Tracker's 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)
"6DBB4"	driveBAY/trakSTAR
"6DMC180-4"	3D Guidance (4 sensor 3DGuidance med-SAFE™)
"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. 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 Tracker Electronics Unit. 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 the Tracker's 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 the Tracker's transmitter. You can not 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

METALsensitivity

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 to the target Tracker sensor, 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 the Tracker 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 measurement cycles 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 Tracker returns to the host computer 14 bytes defining the physical configuration of each sensor attached to the unit. This command can be sent to the tracker either before or after the unit is running. The response has the following format, where one byte is returned for each possible sensor 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. If 0, device is not accessible. A device is accessible when it is attached and powered on. It may or may not be running
BIT 6	If 1, the device is running. If 0, device is not running. A device is running when the power switch to the unit is on, it has been AUTO-CONFIGed and it is AWAKE (in RUN mode). 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 (transmitter OFF).
BIT 7	If 1, device is accessible. If 0, device is not accessible. A device is accessible when it is attached and powered on. It may or may not be running.
BIT 6	If 1, the device is running. If 0, device is not running. A device is running when the power switch to the unit is on, it has been AUTO-CONFIGed and it is AWAKE (in RUN mode). 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 (transmitter OFF).
BIT 5	If 1, then there is a valid sensor at this address. If 0, then no sensor at this address was found.
BIT 4	If 1, then the transmitter is an ERT/WRT. If 0, transmitter is standard range

BIT 3	If 1, ERT/WRT #3 is present. If 0, not present
BIT 2	If 1, ERT/WRT #2 is present. If 0, not present
BIT 1	If 1, ERT/WRT #1 is present. If 0, not present
BIT 0	If 1, ERT/WRT #0 or standard range transmitter is present. If 0, not present.

## 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 trakSTAR and driveBAY, there are four Tracked sensors.

When PARAMETERnumber = 50 (0x32h), during an CHANGE VALUE command, the 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 Tracker sensors that should be used in the 1 transmitter/multiple sensor mode. The command sequence to AutoConfig for 4 sensors would be:

0x50 - Examine/Change  
0x32 –autoconfig- 1 transmitter/n sensors  
0x04. – 4 sensors

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** (treats unit as 1 sensor system) or **One Transmitter/Multiple Sensors** mode.

FBB DEVICES is used to tell which Trackers/Sensors on the FBB are running.

FBB DEPENDENTS is not used by the 3DGuidance systems.

The bit definitions of the bytes are:

Byte	Description	
1	FBB Configuration Mode: 0 - STANDALONE 1 - ONE TRANSMITTER/MULTIPLE SENSORS	
	FBB Devices:	
2,3	Bit	Definition
	15	0
	14	If 1, device at address 14 is running If 0, device at address 14 is not running
	A Tracker is RUNNING when the powers 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.	
	13	If 1, device at address 13 is running If 0, device at address 13 is not running
	1	If 1, device at address 1 is running If 0, device at address 1 is not running
	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 system will return 2 bytes indicating the revision number of the Boot Loader firmware. For example, 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. For example, if the first byte returned is 1 and the second byte is 2, the firmware revision number is 1.2.

## NONDIPOLE POSERVER FIRMWARE REVISION

When PARAMETERnumber = 133 during EXAMINE, the system will return 2 bytes indicating the revision number of the NonDipole POServer firmware. For example, if the first byte returned is 1 and the second byte is 2, the firmware revision number is 1.2.



## **FIVEDOF FIRMWARE REVISION**

When PARAMETERnumber = 135 during EXAMINE, the system will return 2 bytes indicating the revision number of the 5DOF firmware. For example, if the first byte returned is 1 and the second byte is 2, the firmware revision number is 1.2.

## **SIXDOF FIRMWARE REVISION**

When PARAMETERnumber = 136 during EXAMINE, the system will return 2 bytes indicating the revision number of the 6DOF firmware. For example, 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. For example, 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_SIGNS
--------------	-----------	------------

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, centred 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		
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 behaviour 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:

$$\begin{bmatrix} M(1, 1) & M(1, 2) & M(1, 3) \\ M(2, 1) & M(2, 2) & M(2, 3) \\ M(3, 1) & M(3, 2) & M(3, 3) \end{bmatrix}$$

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

$$\begin{bmatrix} \cos(Y)*\cos(Z) & \cos(Y)*\sin(Z) & -\sin(Y) \\ -\cos(X)*\sin(Z) & \cos(X)*\cos(Z) & \\ \sin(X)*\sin(Y)*\cos(Z) & \sin(X)*\sin(Y)*\sin(Z) & \sin(X)*\cos(Y) \\ \sin(X)*\sin(Z) & -\sin(X)*\cos(Z) & \\ \cos(X)*\sin(Y)*\cos(Z) & \cos(X)*\sin(Y)*\sin(Z) & \cos(X)*\cos(Y) \end{bmatrix}$$

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

$$\begin{bmatrix} \cos(E)*\cos(A) & \cos(E)*\sin(A) & -\sin(E) \\ -\cos(R)*\sin(A) & \cos(R)*\cos(A) & \\ \sin(R)*\sin(E)*\cos(A) & \sin(R)*\sin(E)*\sin(A) & \sin(R)*\cos(E) \\ \sin(R)*\sin(A) & -\sin(R)*\cos(A) & \\ \cos(R)*\sin(E)*\cos(A) & \cos(R)*\sin(E)*\sin(A) & \cos(R)*\cos(E) \end{bmatrix}$$

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 decrease, indicating less error, and then finally start increasing again as the new metal causes more distortion.

---

**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 that 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 METAL data 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:

$METALerror = Sensitivity \times (METALerrorSYSTEM - (Offset + Slope \times Range))$ . 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, ie 36" for a flock. 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 SENSORS

Normally the position outputs from the system represent the x, y, z position of the centre 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 centre of the sensor. Figure 6.1 shows an application of the offset command where the desired positional outputs from the Tracker differ from the normal x, y, z sensor outputs.

Referring to Figure 6.1, 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 centre to the desired position (**in inches**). 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:

$Xoffset = 4915 = 5.4 * 32768 / 36.$

$Yoffset = 63625 = 65536 - 1911$

$Zoffset = 1183$



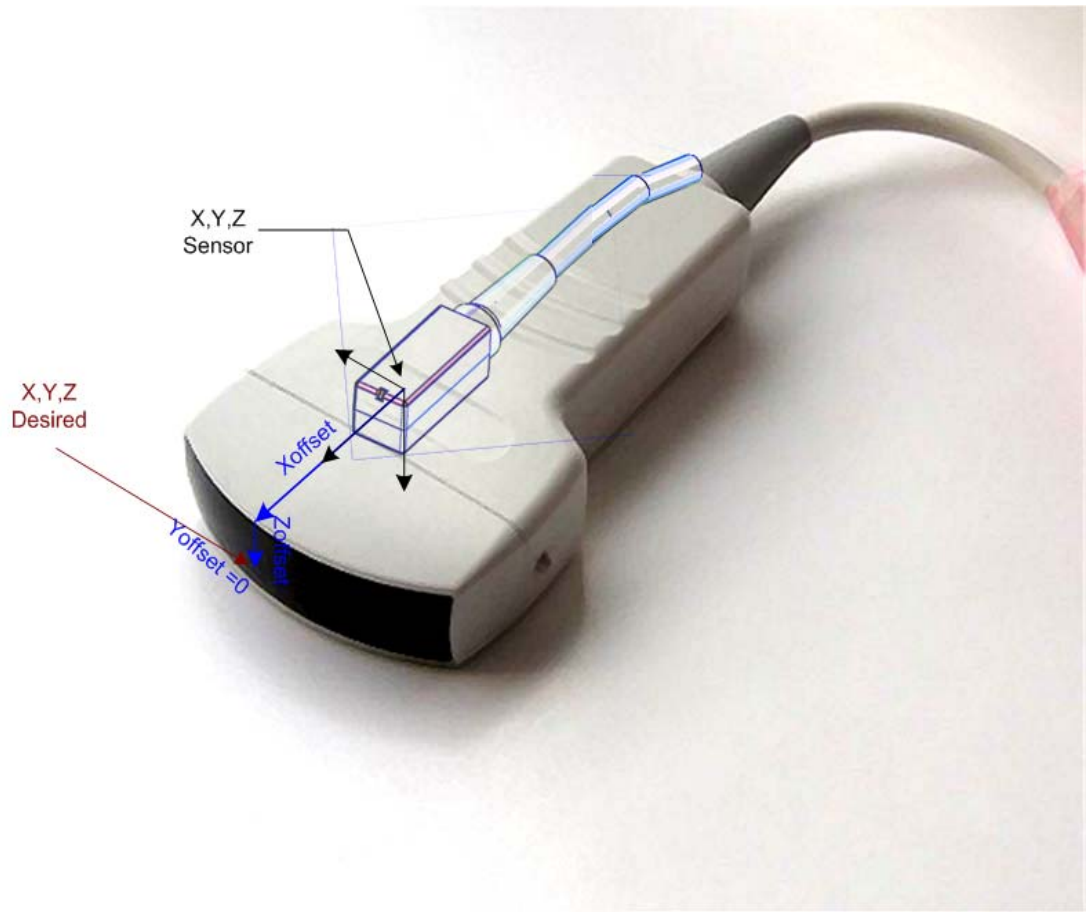


Figure 7-2 Using the OFFSET command

## 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				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

The X, Y, and Z values vary between the binary equivalent of  $\pm \text{MAX}$  inches, where  $\text{MAX} = 36''$  or  $72''$ .

Scaling of each position coordinate is full scale = MAX inches. That is,  $+\text{MAX} = 7\text{FFF Hex}$ ,  $0 = 0 \text{ Hex}$ ,  $-\text{MAX} = 8000 \text{ Hex}$ . Since the maximum range ( $\text{Range} = \text{square root}(X^{**2}+Y^{**2}+Z^{**2})$ ) 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  $\pm 32767$ . 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:

$$(\text{signed int}(\text{Hex \#}) * 36) / 32768$$

$$\text{Or: } (\text{signed int}(\text{Hex \#}) * 72) / 32768$$

## 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				Byte #
7	6	5	4	3	2	1	0	
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
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	]	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  $q_3$  which 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				LSB				Byte #
7	6	5	4	3	2	1	0	
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 M $q_0$
0	B15	B14	B13	B12	B11	B10	B9	#8 MSbyte M $q_0$
0	B8	B7	B6	B5	B4	B3	B2	#9 LSbyte M $q_1$
0	B15	B14	B13	B12	B11	B10	B9	#10 MSbyte M $q_1$
0	B8	B7	B6	B5	B4	B3	B2	#11 LSbyte M $q_2$
0	B15	B14	B13	B12	B11	B10	B9	#12 MSbyte M $q_2$
0	B8	B7	B6	B5	B4	B3	B2	#13 MSbyte $q_3$
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 scaler 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				Byte #
7	6	5	4	3	2	1	0	
1	B8	B7	B6	B5	B4	B3	B2	#1 LSbyte M $q_0$
0	B15	B14	B13	B12	B11	B10	B9	#2 MSbyte M $q_0$
0	B8	B7	B6	B5	B4	B3	B2	#3 LSbyte M $q_1$
0	B15	B14	B13	B12	B11	B10	B9	#4 MSbyte M $q_1$
0	B8	B7	B6	B5	B4	B3	B2	#5 LSbyte M $q_2$
0	B15	B14	B13	B12	B11	B10	B9	#6 MSbyte M $q_2$
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 =  $+0.99996 = 7FFF$  Hex, 0 = 0 Hex, and  $-1.0 = 8000$  Hex.

# 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. 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				
7	6	5	4	3	2	1	0	Byte #
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

---

**Note** **Note:** For alternate Report Rate settings, use the **CHANGE/ EXAMINE** command for this function. See [“REPORT RATE” on page 201.](#)

---

If you do not want a Tracker data record output to your host computer every Tracker 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.

## RESET

	ASCII	HEX	DECIMAL	BINARY
Command Byte	b	62	98	01100010

The RESET command is issued to the standalone Tracker to restart the entire 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.

## RS232 TO FBB

	ASCII	HEX	DECIMAL	BINARY
Command Byte	≡	F0	240	11110000 + FBB (SENSOR) ADDR

---

**Note** For legacy users: The 3D Guidance systems only support Normal Addressing Mode.

---

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 trackers via a single RS232 interface. However, this command is equally relevant for the 3D Guidance 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)

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)

---

**Note** No jumpers or FBB cables required for multi-sensor operation in the 3D Guidance **systems**.

---

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)

---

**Note** To use STREAM mode with multiple sensors, first send the [GROUP MODE](#) command, followed by [STREAM](#) command.

---

## RUN

	ASCII	HEX	DECIMAL	BINARY
Command Byte	F	46	70	01000110

The **RUN** command is issued to the standalone Tracker to restart normal system operation after the Tracker 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

---

**Note** To maximize the life of your system, issue the **SLEEP** command when you are not using the tracker, or configure the Sleep on Reset setting in the Configuration Utility.

---

The **SLEEP** command turns the transmitter off, and halts the system. While asleep, The Tracker will respond to data requests and **RUN** changes but the data output will not change. To resume normal system operation, issue the **RUN** command.

## 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 3DGuidance 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.

## 7.3 Error Reporting

The Tracker continuously monitors tracker activities and reports particular conditions 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 reported to an internal SYSTEM ERROR buffer. This buffer holds up to 32 error codes.

Note that there are two Examine commands that are used together to retrieve error codes from the tracker.

1. Examine Parameter 10 – retrieves single byte error code and removes it from the SYSTEM ERROR buffer.
2. Examine Parameter 16 - retrieves a two byte error code and does not remove it from the SYSTEM ERROR buffer.
  - Bit 16 - unused
  - Bit 15 - flag for a diagnostic error
  - Bit 14 – unused
  - Bit 13 – Fatal error
  - Bits 9-12 – Bird Address (Sensor address; 0 if not sensor related)
  - Bits 1-8 – Error Code

### Diagnostic Errors

Note that errors reported by the tracker immediately after power-up may be diagnostic messages that are reported by the tracker for use by the USB based API (ATC3DG). To confirm and/or see what information these messages contain, you can run and observe the ‘Communication Init’ window in the tracker’s Configuration Utility. Diagnostic messages are indicated by setting bit 15 in the 2-byte error code (returned with Examine Parameter 16).

```
diagError= (birdError >> 14) & 0x01; // Diag error in 15th bit
```

These diagnostic messages may be safely ignored, without affecting tracker operation. For a full detailed listing of these diagnostic codes, please contact Ascension.



## Notification

You can choose to be notified that an error has been generated through any of the following methods:

Error Notification	Description
Error Flag	<p>Monitor the ERROR bit (<b>B13</b>) in the <b>two byte</b> BIRD STATUS register.</p> <p>Send EXAMINE VALUE command with PARAMETERnumber = 0</p> <p>When an error is detected this bit is set to a '1', and the generated error code is sent to the SYSTEM ERROR buffer.</p> <p>To retrieve the code after the flag has been set:</p> <p>Send the EXAMINE VALUE command with PARAMETERnumber = 10</p> <p>This returns the earliest Error sent to the buffer and clears it.</p> <p><b>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.</b></p>
SYSTEM ERROR	<p>Alternatively, you can query the SYSTEM ERROR buffer directly.</p> <p>Send the EXAMINE VALUE command with PARAMETERnumber = 10</p> <p>This returns the earliest Error sent to the buffer. and clears it.</p> <p>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'.</p> <p><b>Note: If the query returns '45', then the 16-byte(16 error codes) buffer has overflowed, and the newest codes generated will be lost.</b></p>

## Error Code Listing

Code	Error Description
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 the user's 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.