



Ascension
Technology Corporation



3D Guidance trakSTAR 2TM

Installation and Operation Guide

www.ascension-tech.com

3D Guidance trakSTAR 2™

Installation and Operation Guide

REVISION NUMBER:

940052 Rev 2
6/6/2011

TRADEMARKS

Microsoft Windows XP®, Windows Vista®, and Windows 7®
are registered trademarks of Microsoft Corporation.
All other products mentioned in this manual are trademarks
or registered trademarks of their respective companies.

3DGuidance **trakSTAR 2** is a trademark of Ascension Technology Corporation
© 2010 Ascension Technology Corporation. All rights reserved.
P.O. Box 527
Burlington, VT 05402

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

Table of Contents

| | |
|--|-----------|
| Introduction | 1 |
| About This Guide | 1 |
| How This Guide is Organized | 1 |
| Guide Conventions | 2 |
| Getting Assistance | 3 |
| Preparing for Setup..... | 4 |
| System Requirements..... | 4 |
| Intended Use Statement..... | 4 |
| Software Requirements | 4 |
| Hardware Requirements | 4 |
| Unpacking the System..... | 5 |
| Package Checklist..... | 5 |
| Safe Performance and Handling Precautions | 8 |
| Environmental Conditions | 9 |
| Temperature | 9 |
| Humidity | 9 |
| Quick Start: Setup and Checkout..... | 10 |
| Install the Software | 10 |
| Windows Vista-32 Bit Installation..... | 13 |
| Cable Connections..... | 13 |
| Installing the Driver | 16 |
| System Checkout: Running the USB Demo Software | 17 |
| What you will see:..... | 18 |
| Configuration and Basic Operation | 21 |
| Default Configuration | 21 |
| Configurable Power-up Settings: | 24 |
| Default Reference Frame | 28 |
| Flat Transmitter..... | 29 |
| MAGnet Transmitter..... | 29 |
| Changing Your Settings | 30 |
| Mounting the Hardware | 31 |
| Mid-Range Transmitter Mounting..... | 31 |
| Short-Range Transmitter Mounting..... | 32 |
| Wide-Range Transmitter Mounting..... | 32 |
| Flat Transmitter Mounting..... | 33 |
| MAGnet Transmitter Mounting..... | 33 |

| | |
|--|-----------|
| Sensor Mounting..... | 34 |
| Electronics Unit Mounting | 34 |
| Rear Panel Connectors..... | 35 |
| Basic Operation..... | 37 |
| Dipole Transmitters..... | 37 |
| NonDipole Transmitters..... | 37 |
| 6DOF Sensor..... | 37 |
| 5DOF Sensor..... | 37 |
| Electronics Unit..... | 38 |
| Measurement Cycle..... | 38 |
| Calibration | 39 |
| Performance Factors..... | 39 |
| Electromagnetic and Other Interference in Tracking..... | 39 |
| Noise | 39 |
| Distortion..... | 40 |
| Tracker as the Cause of Interference | 41 |
| Factors in Tracker Accuracy | 41 |
| Warm-up..... | 41 |
| Measurement Rate | 41 |
| Equipment Alteration..... | 42 |
| Power Grid Magnetic Interference | 42 |
| Performance Motion Box..... | 42 |
| Flat Transmitter Motion Box..... | 44 |
| Software Operation - Tools for Successful Tracking..... | 45 |
| Software Overview | 45 |
| 3DGuidance Windows API | 45 |
| Ascension RS232 Interface..... | 48 |
| RS232 Windows Driver | 48 |
| Direct Communication: ATC RS232 Interface Protocol | 49 |
| 3DGuidance API Reference..... | 50 |
| Using 3DGuidance..... | 50 |
| Quick Reference..... | 51 |
| SYSTEM..... | 51 |
| SENSOR..... | 53 |
| BOARD | 58 |
| TRANSMITTER | 59 |
| Pre-Initialization Setup..... | 60 |
| System Initialization..... | 61 |
| System Setup..... | 62 |
| Sensor Setup..... | 64 |
| Transmitter Setup..... | 67 |
| Acquiring Tracking Data..... | 68 |
| 3DGuidance API | 71 |
| 3D Guidance API Functions | 72 |
| InitializeBIRDSystem | 73 |

| | |
|---|------------|
| GetBIRDSystemConfiguration..... | 75 |
| GetTransmitterConfiguration..... | 76 |
| GetSensorConfiguration..... | 77 |
| GetBoardConfiguration..... | 78 |
| GetSystemParameter..... | 79 |
| GetSensorParameter..... | 80 |
| GetTransmitterParameter..... | 82 |
| GetBoardParameter..... | 84 |
| SetSystemParameter..... | 86 |
| SetSensorParameter..... | 88 |
| SetTransmitterParameter..... | 90 |
| SetBoardParameter..... | 92 |
| GetAsynchronousRecord..... | 94 |
| GetSynchronousRecord..... | 96 |
| GetBIRDError..... | 99 |
| GetErrorText..... | 100 |
| GetSensorStatus..... | 102 |
| GetTransmitterStatus..... | 104 |
| GetBoardStatus..... | 106 |
| GetSystemStatus..... | 108 |
| SaveSystemConfiguration..... | 110 |
| RestoreSystemConfiguration..... | 112 |
| CloseBIRDSystem..... | 114 |
| 3D Guidance API Structures | 115 |
| SYSTEM_CONFIGURATION..... | 116 |
| TRANSMITTER_CONFIGURATION..... | 118 |
| SENSOR_CONFIGURATION..... | 119 |
| BOARD_CONFIGURATION..... | 120 |
| ADAPTIVE_PARAMETERS..... | 122 |
| QUALITY_PARAMETERS..... | 123 |
| VPD_COMMAND_PARAMETER..... | 124 |
| POST_ERROR_PARAMETER..... | 125 |
| DIAGNOSTIC_TEST_PARAMETERS..... | 126 |
| COMMUNICATIONS_MEDIA_PARAMETERS..... | 130 |
| BOARD_REVISIONS..... | 131 |
| SHORT_POSITION_RECORD..... | 133 |
| SHORT_ANGLES_RECORD..... | 135 |
| SHORT_MATRIX_RECORD..... | 137 |
| SHORT_QUATERNIONS_RECORD..... | 139 |
| SHORT_POSITION_ANGLES_RECORD..... | 140 |
| SHORT_POSITION_MATRIX_RECORD..... | 141 |
| SHORT_POSITION_QUATERNION_RECORD..... | 142 |
| DOUBLE_POSITION_RECORD..... | 143 |
| DOUBLE_ANGLES_RECORD..... | 145 |
| DOUBLE_MATRIX_RECORD..... | 146 |
| DOUBLE_QUATERNIONS_RECORD..... | 148 |
| DOUBLE_POSITION_ANGLES_RECORD..... | 149 |
| DOUBLE_POSITION_MATRIX_RECORD..... | 150 |

| | |
|--|------------|
| DOUBLE_POSITION_QUATERNION_RECORD..... | 151 |
| DOUBLE_POSITION_TIME_STAMP_RECORD..... | 152 |
| DOUBLE_ANGLES_TIME_STAMP_RECORD..... | 153 |
| DOUBLE_MATRIX_TIME_STAMP_RECORD..... | 154 |
| DOUBLE_QUATERNIONS_TIME_STAMP_RECORD..... | 155 |
| DOUBLE_POSITION_ANGLES_TIME_STAMP_RECORD..... | 156 |
| DOUBLE_POSITION_MATRIX_TIME_STAMP_RECORD..... | 157 |
| DOUBLE_POSITION_QUATERNION_TIME_STAMP_RECORD..... | 158 |
| DOUBLE_POSITION_TIME_Q_RECORD..... | 159 |
| DOUBLE_ANGLES_TIME_Q_RECORD..... | 160 |
| DOUBLE_MATRIX_TIME_Q_RECORD..... | 161 |
| DOUBLE_QUATERNIONS_TIME_Q_RECORD..... | 162 |
| DOUBLE_POSITION_ANGLES_TIME_Q_RECORD..... | 163 |
| DOUBLE_POSITION_MATRIX_TIME_Q_RECORD..... | 165 |
| DOUBLE_POSITION_QUATERNION_TIME_Q_RECORD..... | 166 |
| SHORT_ALL_RECORD..... | 167 |
| DOUBLE_ALL_RECORD..... | 169 |
| DOUBLE_ALL_TIME_STAMP_RECORD..... | 171 |
| DOUBLE_ALL_TIME_STAMP_Q_RECORD..... | 173 |
| DOUBLE_ALL_TIME_STAMP_Q_RAW_RECORD..... | 175 |
| DOUBLE_POSITION_ANGLES_TIME_Q_BUTTON_RECORD..... | 177 |
| DOUBLE_POSITION_MATRIX_TIME_Q_BUTTON_RECORD..... | 179 |
| DOUBLE_POSITION_QUATERNION_TIME_Q_BUTTON_RECORD..... | 181 |
| 3D Guidance Enumeration Types..... | 183 |
| BIRD_ERROR_CODES..... | 184 |
| SENSOR_PARAMETER_TYPE..... | 189 |
| MESSAGE_TYPE..... | 196 |
| TRANSMITTER_PARAMETER_TYPE..... | 197 |
| BOARD_PARAMETER_TYPE..... | 199 |
| SYSTEM_PARAMETER_TYPE..... | 200 |
| HEMISPHERE_TYPE..... | 202 |
| AGC_MODE_TYPE..... | 203 |
| DATA_FORMAT_TYPE..... | 204 |
| BOARD_TYPES..... | 206 |
| DEVICE_TYPES..... | 207 |
| COMMUNICATIONS_MEDIA_TYPES..... | 209 |
| 3D Guidance API Status/Error Bit Definitions..... | 210 |
| ERRORCODE..... | 211 |
| DEVICE_STATUS..... | 212 |
| 3D Guidance Initialization Files..... | 213 |
| 3D Guidance Initialization File Format Reference..... | 213 |
| [System]..... | 214 |
| [Sensorx]..... | 215 |
| [Transmitterx]..... | 217 |
| Ascension RS232 Interface Reference..... | 219 |
| RS232 Signal Description..... | 219 |

3D GUIDANCE TRAKSTAR 2™ INSTALLATION AND OPERATION GUIDE

| | |
|--|------------|
| Using the 'reset on CTS' feature | 220 |
| RS232 Commands | 220 |
| Command Summary..... | 221 |
| Command Utilization..... | 222 |
| Response Format..... | 223 |
| Position/Orientation Data Format | 223 |
| RS232 Command Reference | 225 |
| ANGLES..... | 226 |
| ANGLE ALIGN..... | 227 |
| BORESIGHT | 228 |
| BORESIGHT REMOVE | 229 |
| BUTTON MODE | 230 |
| BUTTON READ | 231 |
| CHANGE VALUE..... | 232 |
| EXAMINE VALUE..... | 232 |
| TRACKER STATUS..... | 236 |
| SOFTWARE REVISION NUMBER | 237 |
| TRACKER COMPUTER CRYSTAL SPEED..... | 237 |
| POSITION SCALING..... | 237 |
| FILTER ON/OFF STATUS | 237 |
| DC FILTER CONSTANT TABLE ALPHA_MIN | 239 |
| DISABLE/ENABLE DATA READY OUTPUT..... | 239 |
| SET DATA READY CHARACTER | 240 |
| ERROR CODE | 240 |
| DC FILTER TABLE V _m | 240 |
| DC FILTER CONSTANT TABLE ALPHA_MAX..... | 241 |
| SUDDEN OUTPUT CHANGE LOCK | 241 |
| SYSTEM MODEL IDENTIFICATION..... | 242 |
| XYZ REFERENCE FRAME..... | 242 |
| FILTER LINE FREQUENCY | 243 |
| HEMISPHERE..... | 243 |
| ANGLE ALIGN | 244 |
| REFERENCE FRAME | 244 |
| TRACKER SERIAL NUMBER..... | 244 |
| SENSOR SERIAL NUMBER | 244 |
| TRANSMITTER SERIAL NUMBER | 244 |
| METAL | 245 |
| REPORT RATE..... | 245 |
| GROUP MODE | 245 |
| SYSTEM STATUS | 246 |
| AUTOCONFIG..... | 247 |
| SENSOR OFFSET | 248 |
| BOOT LOADER FIRMWARE REVISION..... | 248 |
| MDSP FIRMWARE REVISION | 248 |
| NONDIPOLE POSERVER FIRMWARE REVISION | 248 |
| FIVEDOF FIRMWARE REVISION | 249 |
| SIXDOF FIRMWARE REVISION | 249 |
| DIPOLE POSERVER FIRMWARE REVISION | 249 |
| HEMISPHERE..... | 250 |
| MATRIX..... | 252 |
| METAL..... | 254 |

| | |
|---|------------|
| OFFSET | 257 |
| POINT | 259 |
| POSITION | 260 |
| POSITION/ANGLES | 261 |
| POSITION/MATRIX | 262 |
| POSITION/QUATERNION | 263 |
| QUATERNION | 264 |
| REFERENCE FRAME | 265 |
| REPORT RATE | 266 |
| RESET | 267 |
| RS232 TO FBB | 268 |
| RUN | 270 |
| SLEEP | 271 |
| STREAM | 272 |
| STREAM STOP | 273 |
| Error Reporting | 274 |
| Diagnostic Errors | 274 |
| Error Code Listing | 276 |
| Troubleshooting, Maintenance, and Repair | 278 |
| User Maintenance | 278 |
| Maintenance Prior to Each Use | 278 |
| Periodic Maintenance (As needed) | 278 |
| Proper Handling of Sensor and Cable | 279 |
| Cleaning and Disinfecting | 279 |
| Firmware Updates | 279 |
| Disposal | 280 |
| Troubleshooting | 281 |
| Error Codes | 282 |
| LED Definitions | 283 |
| Contacting Ascension Technology Corporation for Repair | 285 |
| Fuses | 285 |
| Other | 285 |
| Warranty | 285 |
| Regulatory Information and Specifications | 286 |
| Mid-Range, Short Range, Flat, and MAGnet Transmitter Configurations: | 286 |
| EC Declaration of Conformity | 288 |
| FCC Compliance Statement | 289 |
| Radio and television interference | 289 |
| Wide-Range Transmitter Configurations: | 290 |
| FCC Regulations | 290 |
| EC Declaration of Conformity | 291 |
| Product Specifications | 292 |
| Performance | 292 |

| | |
|--|------------|
| Physical | 293 |
| Appendix I: winBIRD | 295 |
| Running the RS232 Demo Utility | 295 |
| Appendix II: driveBAY 2 Utility | 299 |
| Running the driveBAY 2 Utility | 299 |
| Setup | 299 |
| Run the Utility | 299 |
| Appendix III: Application Notes | 302 |
| Computing Stylus Tip Coordinates | 302 |
| Explaining Measurement Rate vs. Update Rate..... | 303 |
| Older Systems: | 303 |
| 3DGuidance Systems | 303 |

List of Figures

| | |
|--|-----|
| Figure 1 trakSTAR 2 Electronics Unit | 5 |
| Figure 2 trakSTAR 2 Sensors..... | 5 |
| Figure 3 Transmitter Options Available..... | 6 |
| Figure 4 trakSTAR 2 Cables..... | 7 |
| Figure 5 3D Guidance trakSTAR 2 CD-ROM..... | 7 |
| Figure 6 3D Guidance Setup Wizard | 10 |
| Figure 7 3D Guidance Setup Wizard – Confirming Target Folder | 11 |
| Figure 8 3D Guidance Setup Wizard – Confirming Installation | 11 |
| Figure 9 3D Guidance Setup Wizard – Installing USB Driver | 12 |
| Figure 10 Windows Logo Testing Message Box | 12 |
| Figure 11 3D Guidance Readme File..... | 12 |
| Figure 12 Windows-Vista Driver Installation Wizard..... | 13 |
| Figure 13 Connecting the Transmitter | 13 |
| Figure 14 Connecting the Sensors..... | 14 |
| Figure 15 Connecting RS232 Interface Cable (trakSTAR 2 Rear Panel)..... | 14 |
| Figure 16 Connecting USB Interface Cable (trakSTAR 2 Rear Panel) | 15 |
| Figure 17 Location of AC Power Input Connector (trakSTAR 2 Rear Panel) | 15 |
| Figure 18 Location of AC Power Switch (trakSTAR 2 Rear Panel) | 15 |
| Figure 19 Location of Front Panel LED..... | 16 |
| Figure 20 Found New Hardware Wizard | 16 |
| Figure 21 Found New Hardware (Closing Wizard)..... | 17 |
| Figure 22 Location of Demo Utility (CUBES)..... | 17 |
| Figure 23 CUBES Window | 18 |
| Figure 24 Location of CUBES Run Button..... | 18 |
| Figure 25 Typical Data Collection Position and Orientation Display | 18 |
| Figure 26 Default Transmitter/Sensor Coordinate Frames..... | 28 |
| Figure 27 Flat Transmitter Coordinate Frame..... | 29 |
| Figure 28 MAGnet Transmitter Coordinate Frame..... | 29 |
| Figure 29 Mid-Range Transmitter Mounting Dimensions (inches) -Top and Side Views | 31 |
| Figure 30 Short-Range Transmitter Mounting Dimensions (inches) Top and Side Views..... | 32 |
| Figure 31 Wide-Range Transmitter Mounting Dimensions | 33 |
| Figure 32 Cable Mounting Requirements..... | 34 |
| Figure 33 trakSTAR 2 Rear Panel Connectors | 35 |
| Figure 34 Performance Motion Box – MRT and Model 800 (8 mm) Sensors | 42 |
| Figure 35 Performance Motion Box – Flat Transmitter..... | 44 |
| Figure 36 : Measurement Reference Frame (Mid-Range Transmitter)..... | 133 |
| Figure 37 Measurement Reference Frame (Mid-Range Transmitter) | 143 |
| Figure 38 Measurement Reference Frame | 190 |
| Figure 39 Receiver Zero Orientation (8mm Sensor)..... | 190 |
| Figure 40 Measurement Reference Frame (Standard Transmitter)..... | 197 |
| Figure 41 Receiver Zero Orientation (8mm Sensor)..... | 198 |
| Figure 42 Rear View: 9-Pin D-Sub Connector | 219 |
| Figure 43 Using the OFFSET command..... | 258 |
| Figure 44 Accessing the winBIRD Installer..... | 295 |
| Figure 45 Opening the winBIRD Demo Program..... | 295 |
| Figure 46 Setting up for RS232 | 296 |
| Figure 47 Setting the RS232 Baud Rate..... | 296 |
| Figure 48 Setting Sensor 1 (Device 1) Parameters..... | 297 |

3D GUIDANCE TRAKSTAR 2™ INSTALLATION AND OPERATION GUIDE

| | |
|--|-----|
| Figure 49 Setting Remaining Sensors Parameters..... | 297 |
| Figure 50 Selecting Animate from Take Data Drop-Down Menu | 297 |
| Figure 51 driveBAY 2 Utility Communication Initialize Window..... | 299 |
| Figure 52 driveBAY 2 Utility Window (Dipole Default Settings Tab Displayed)..... | 300 |
| Figure 53 driveBAY 2 Utility Flash Maintenance | 301 |
| Figure 54 Transmitter Excitation Waveform..... | 304 |

List of Tables

| | |
|--|-----|
| Table 1 Power Up Default Dipole Settings: | 22 |
| Table 2 Power Up Default NonDipole Settings: | 23 |
| Table 3 Configurable Power Up Settings..... | 24 |
| Table 4 trakSTAR 2 Rear Panel Connections..... | 35 |
| Table 5 Location of API Files and Demo Applications | 45 |
| Table 6 Location of Sample Utility Programs..... | 47 |
| Table 7 Diagnostic Test Usage..... | 129 |
| Table 8 TrakSTAR 2 9-pin RS-232C connector pinout..... | 219 |
| Table 9 RS232 Command Summary | 221 |
| Table 10 Hypothetical Command Sequence | 222 |
| Table 11 List of Examine Value Valid Parameters..... | 233 |
| Table 12 Tracker Status Bits | 236 |
| Table 13 Filter On/Off Status Bit Assignments | 238 |
| Table 14 DC Filter Constant Table Default Values (Std Range Transmitter) | 239 |
| Table 15 DC Filter Table Default Values (Dipole Transmitters)..... | 240 |
| Table 16 DC Filter Constant Table Default Values (Dipole Transmitters)..... | 241 |
| Table 17 System Model Identification..... | 242 |
| Table 18 List of Hemisphere Definitions..... | 243 |
| Table 19 System Status Bit Descriptions | 246 |
| Table 20 AUTOCONFIG Bit Definitions..... | 247 |
| Table 21 Error Notification Options | 275 |
| Table 22 Troubleshooting Symptoms, Possible Causes, and Solutions | 281 |
| Table 23 LED Definitions..... | 283 |

Introduction

Congratulations on your purchase of our Ascension Technology Corporation (ATC) 3DGuidance **trakSTAR 2**. Our company is proud of the quality of all our trackers and is committed to making your experience with this device a good one.

trakSTAR 2 is a high-accuracy electromagnetic tracker designed for short-range motion tracking applications. It employs our latest pulsed DC technology to track the position and orientation (up to six degrees-of-freedom or 6DOF) of multiple sensors within the operating range of the transmitter. Sensor data is reported serially to a host computer via a USB or RS232 interface.

This User's Guide will help you set up and install the 3DGuidance **trakSTAR 2** hardware and software as well as configuring it for optimal tracking. Please read this document carefully. Proper set up will maximize tracking performance for your particular application.

About This Guide

This Guide – along with its Quick Set-up Handout-- contains everything you'll need to install and run the tracker. It provides valuable information about testing performance and communicating with **trakSTAR 2**. You will also find descriptions of tools available for configuring your tracker and finding immediate help.

How This Guide is Organized

This manual contains eight chapters.

Chapter 1: Preparing for Setup and Safe Performance

- States the Intended Use
- Lists tracker software and hardware requirements
- Outlines the components of your system
- Describes safe use and handling precautions

Chapter 2: Quick Start: Setup and Checkout

- Describes how to connect your system components
- Guides you through a quick checkout, using the demo software

Chapter 3: Configuration and Basic Operation

- Outlines default configuration parameters and reference frames
- Provides component mounting information
- Details the basic principles of tracker operation
- Describes factors that affect tracker performance

Chapter 4: Software -Tools for Successful Tracking

- Outlines the methods for communicating with trakSTAR 2.
- Contains sample programs -- included on your CD-ROM --- that demonstrate the tracker's communication structure

Chapter 5: 3DGuidance API Reference

- Details the 3DGuidance Application Programming Interface (API) for communicating with the tracker using USB

Chapter 6: RS232 Command Interface Reference

- Describes an alternate method for communicating directly with the tracker using the Ascension RS232 Interface protocol

Chapter 7: Troubleshooting, Maintenance, Repair and Disposal

- Offers user-troubleshooting suggestions including setup problems and solutions.
- Addresses maintenance suggestions including cleaning and disinfecting methods.
- Explains how to contact Ascension Technology Corporation if you need assistance.
- Lists replacement part numbers
- Identifies disposal guidance.

Chapter 8: Regulatory Information and Product Specifications

- Lists applicable standards, specifications, and certifications.

Guide Conventions

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



Note: This call-out explains important information about the features of your system



Tips: This call-out provides advice for maximizing the performance of your system



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

Getting Assistance

If you are experiencing a problem with the installation, set up, or operation of your tracker, we suggest that you first consult the troubleshooting table in Chapter 7. It describes common setup problems and suggested solutions.

If you continue to experience problems after following the recommended actions, contact us for expert support.

| | |
|------------------------|---|
| World Wide Web: | http://www.ascension-tech.com/support/ |
| E-mail: | support@ascension-tech.com |
| Telephone: | Call (802) 893-6657 9 a.m.-- 5 p.m. U.S. Eastern Standard Time, Monday through Friday. |
| Fax: | (802) 893-6659 |

Preparing for Setup

This chapter describes everything you will need to setup your 3DGuidance trakSTAR 2

System Requirements

Intended Use Statement

The 3DGuidance trakSTAR 2 is designed to be integrated as part of a larger host system, requiring real-time tracking or measurement of an object's position and orientation in free space.

Software Requirements

Several Windows® based utilities are included on the trakSTAR 2 CD-ROM:

1. **Cubes** - a demonstration utility that communicates using USB and the 3DGuidance USB API.
2. **APITest** – An application that allows you to send any command defined in the 3DGuidance API to the tracker, and to view the associated response.
3. **driveBAY 2 Utility** - a utility for changing default configuration parameters of your system, or loading firmware upgrades. This same utility applicable to both driveBAY 2 and trakSTAR 2 systems.

These utilities and communication with the tracker via the 3DGuidance API require:

Windows XP®, Windows Vista®, or Windows 7®

Hardware Requirements

USB port: trakSTAR 2 can report data serially to a host computer using a USB cable. To do this, an unused USB 2.0 port on your computer is required.

COM port: For legacy Ascension tracker users, the trakSTAR 2 can report data serially to a host computer via a RS232 cable. To do this, an unused COM port on your computer is required.

Power: trakSTAR 2 contains an internal power supply that will operate from 100 to 240V AC, at frequencies of 50-60 Hz, and using up to 70 VA of power.

CD-ROM drive: Only required for accessing the utilities and drivers shipped on a CD-ROM. You may also download these utilities from the Internet by visiting Ascension's web site. (www.ascension-tech.com)

Unpacking the System

Package Checklist

The **trakSTAR 2** is packaged in a single shipping box. Make sure you have received the following items before proceeding to the setup and checkout section:

- Electronics Unit:



Figure 1 trakSTAR 2 Electronics Unit

- Sensor(s): Note that valid sensor options for the **trakSTAR 2** include: Model 800, Model180, Model 130, Model 90, and Model 55

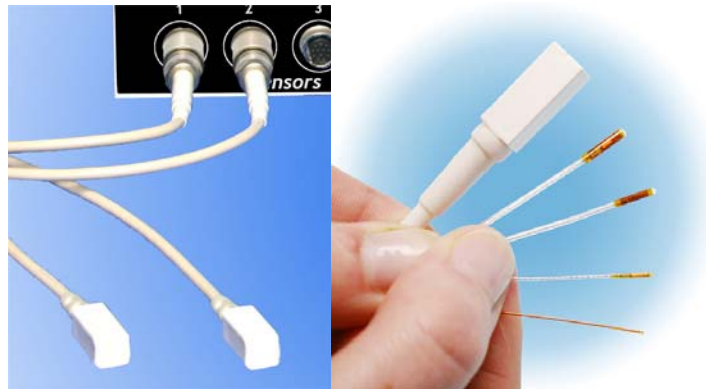
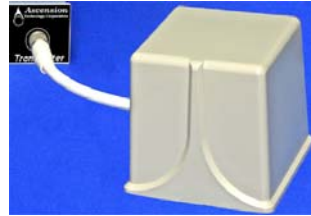


Figure 2 trakSTAR 2 Sensors

- One of the following transmitters



Note: Transmitters are specifically calibrated for use with the driveBAY 2 and trakSTAR 2 systems.



Mid-Range Transmitter



Short-Range Transmitter



Note: Wide Range transmitters are calibrated to use a Max Range Scale setting of '144'. See the [Default Configuration](#) section for details.



Wide-Range Transmitter



5DOF/6DOF Flat Transmitter



5DOF/6DOF MAGnet Transmitter

Figure 3 Transmitter Options Available

- Cables



AC Power Cable



USB Cable



RS232 Cable

Figure 4 trakSTAR 2 Cables

- 3D Guidance trakSTAR 2 CD-ROM:



Figure 5 3D Guidance trakSTAR 2 CD-ROM

If there are any discrepancies or the shipment is damaged, contact us by any of the following means:

| | |
|------------------------|---|
| World Wide Web: | http://www.ascension-tech.com/support/ |
| E-mail: | support@ascension-tech.com |
| Telephone: | Call (802) 893-6657 9 a.m.-- 5 p.m. U.S. Eastern Standard Time, Monday through Friday. |
| Fax: | (802) 893-6659 |

Safe Performance and Handling Precautions

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

- Read this Guide
- Handle the section of cable near the sensor and transmitter housing carefully. Repeated bending of the cable near the sensor or transmitter housing is the most common cause of tracker failure.
- When power is applied or the tracker is running, do not open the cover and touch exposed electronic components. Contact with exposed components could cause injury.
- Remove the sensor or transmitter from mounting brackets or holders gently. Do not yank or pull on the cable.
- Sensors and transmitters can easily be damaged if you carry, throw, or swing them by their cables or if you drop them against a hard surface.
- The Mid-Range (~5 lbs), Wide-Range (~38lbs), Flat (~23lbs), and MAGnet (~7 lbs) transmitters are heavy and may cause injury if dropped on a foot. Use caution when mounting or carrying them.
- Sensors and transmitters can easily be damaged if you carry, throw, or swing them by their cables or if you drop them against a hard surface.
- Be sure to implement a strain relief if you embed a sensor and its cable in an instrument or tool. The point where the sensor cable exits from your tool needs protection. Your sensor will last a long time if you take steps now to distribute forces over an extended region of the cable.
- Our sensor and transmitter cables are precisely bundled, shielded, and calibrated to minimize noise and ensure accurate performance. If add your own extension cables or connectors, you will compromise performance and void both regulatory approvals and your warranty.
- To clean components, wipe them with a cloth dampened with a general purpose cleaning solution such as soap and water, isopropyl alcohol, etc. Do not immerse the transmitters sensors, or cables in liquid.
- Keep the transmitter, sensors, and cables away from sources of heat.
- If mounting the transmitter inside an enclosure, be sure to provide adequate ventilation.
- Never power up the system or place the transmitter in an explosive atmosphere.
- To extend tracker life, shut down the transmitter when not in use. This can be done by turning off the power switch on the rear panel, or by holding the tracker in reset Using the '[reset on CTS](#)' feature or [SLEEP](#) commands (if using the RS232 mode), or setting the System Parameter Type [SELECT TRANSMITTER](#) value to **'-1'** in the 3DGuidance API.
- Tracker components may be subject to interference from or may interfere with other electrical equipment in your environment. Be sure to identify sources of interference in your particular environment as well as devices that might be affected by tracker operation. See the section [Electromagnetic and Other Interference in Tracking](#) for additional details.

Environmental Conditions

The **trakSTAR 2** must be used and maintained in the following ranges only:

Temperature

The tracker operates within specification when the ambient air temperature is between 5 degree C and 40 degree C. The **trakSTAR 2** can be packaged and shipped in environments with an ambient air temperature between -40 degree C and 70 degree C without degradation of its components.

Humidity

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

Quick Start: Setup and Checkout

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

Install the Software

The trakSTAR 2 CD-ROM has an installer that will copy all required software (utilities, sample code, API, etc) and user documentation onto your host PC. **Note that you must have administrator privileges for the installer to run.**



Note: Install the software before connecting the trakSTAR 2.

1. To begin the installer, place the CD-ROM in the tray, and close the drive door.

NOTE: If the installer does not start (drive not set to 'Autoplay'), then browse to the CD-ROM folder and run the 'setup.exe' file.

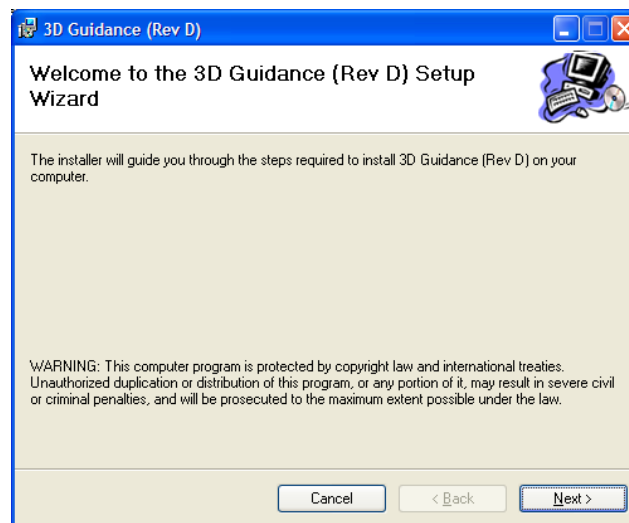


Figure 6 3D Guidance Setup Wizard

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

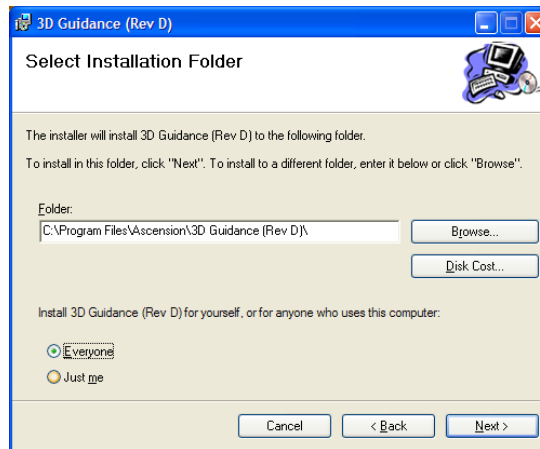


Figure 7 3D Guidance Setup Wizard – Confirming Target Folder

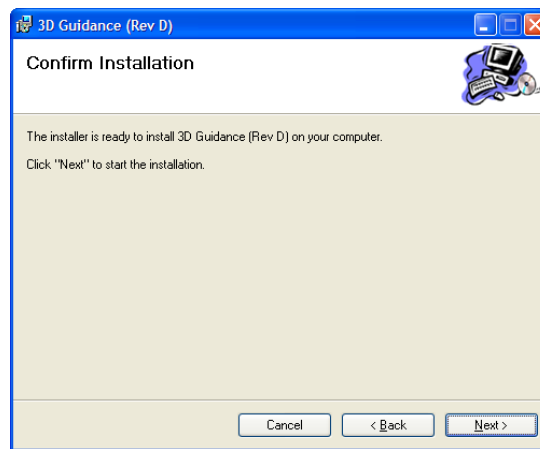


Figure 8 3D Guidance Setup Wizard – Confirming Installation

3. Select a USB driver. For new installations or systems running a 64-bit operating system, select the '**Windows USB driver device driver**' (winusb.sys) - recommended. For continued use of the legacy USB driver (cyusb.sys), select the '**Cypress USB device driver**'.



Figure 9 3D Guidance Setup Wizard – Installing USB Driver

4. When the **Windows Logo Testing** message box displays, click *Continue Anyway* to proceed with the installation.

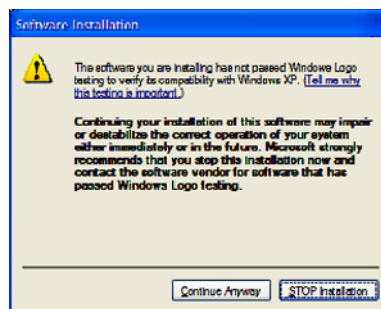


Figure 10 Windows Logo Testing Message Box

After installation is complete, the *Readme.txt* file is displayed providing important information about the software.

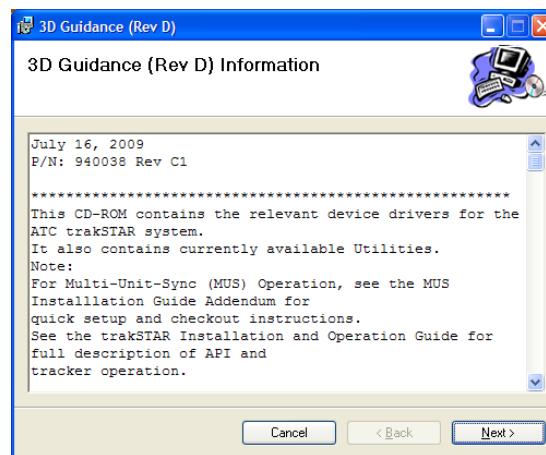


Figure 11 3D Guidance Readme File

Windows Vista-32 Bit Installation


Windows Vista32-bit systems prompt users to run the Driver Installation Wizard. Follow the prompts in the Wizard to copy over the required driver installation files.



Figure 12 Windows-Vista Driver Installation Wizard

Cable Connections

The trakSTAR 2 hardware can be connected and ready to run in four easy steps:

 **Note:** For those running a Multi-Unit Sync (MUS) Configuration, please see the *trakSTAR MUS Installation Guide Addendum*.

1. **Connect the Transmitter** –Plug the transmitter cable into the corresponding connector on the left side of the Electronics Unit's front panel.


 **Note:** For optimum performance, keep the transmitter at least 24 inches away from the Electronics Unit. See the [Mounting the Hardware](#) section for additional mounting information.



Figure 13 Connecting the Transmitter

2. **Connect the Sensor(s) to the Electronics Unit** – Plug the connector of each sensor into a receptacle on the electronics unit. Be sure to fully insert the connector, until audible ‘click’ is heard.



Note: The sensor and transmitter connectors are specifically keyed to mate. You may need to rotate the connectors before they will click into place.



Figure 14 Connecting the Sensors

3. **Connect the Interface Cable** –

RS232 Interface: To communicate with the **trakSTAR 2** using the legacy Ascension RS232 Interface protocol (RS232 link), connect the RS232 cable to the RS232 port on the rear panel of the **trakSTAR 2** and an open COM port on the Host PC.



Figure 15 Connecting RS232 Interface Cable (trakSTAR 2 Rear Panel)

USB Interface: To use the 3DGuidance API (USB interface), connect the USB Cable to the **trakSTAR 2** rear panel and any open USB port on the Host PC.



Figure 16 Connecting USB Interface Cable (trakSTAR 2 Rear Panel)

4. **Apply Power** – Connect the AC Power cord to the AC Power Input Connector on the trakSTAR 2 rear panel and to an AC power source (wall outlet). Then set the power switch to ON (up position).



Figure 17 Location of AC Power Input Connector (trakSTAR 2 Rear Panel)



Figure 18 Location of AC Power Switch (trakSTAR 2 Rear Panel)



Note: When AC power is applied, the trakSTAR 2 internal processors initialize and read data stored in the transmitter and sensor programmable read-only (PROMs) and/or Flash memories. Do NOT send commands to the trakSTAR 2 until the front panel LED is blinking GREEN -(see the [LED Definitions](#) table in Chapter 7 for more information).




Figure 19 Location of Front Panel LED

Installing the Driver

The **New Hardware Wizard** displays after plugging in the USB cable indicating new hardware is detected. (Note that Vista, Win7 systems will initiate the driver install automatically)

1. Follow the New Hardware Wizard prompts, allowing Windows to automatically search for a suitable driver. (This is the default option.)

 **Note:** If the installer (on CD-ROM) was run prior to connecting the hardware, the *Found New Hardware Wizard* will automatically find the drivers.

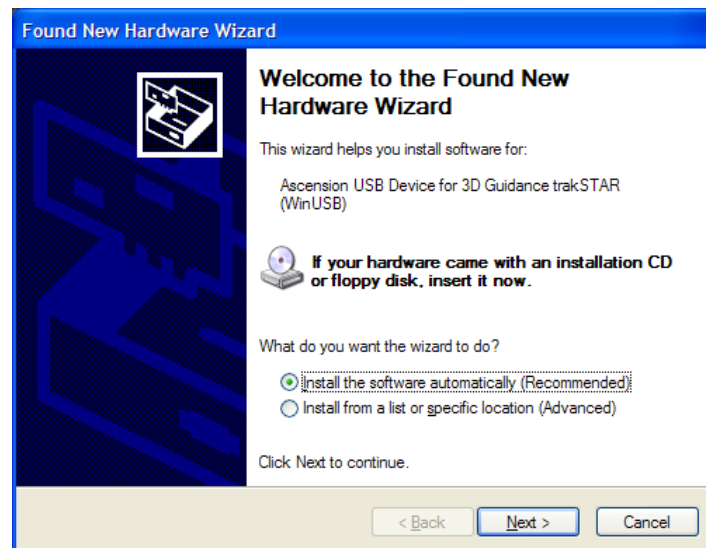


Figure 20 Found New Hardware Wizard

2. Follow the steps in the *New Hardware Wizard*, allowing Windows to install the USB driver.



Note: If a message displays that the software has not passed 'Windows Logo' testing, select *Continue Anyway*.

- Click **Finish** when the *Found New Hardware Wizard* has completed installing the driver to close the wizard.

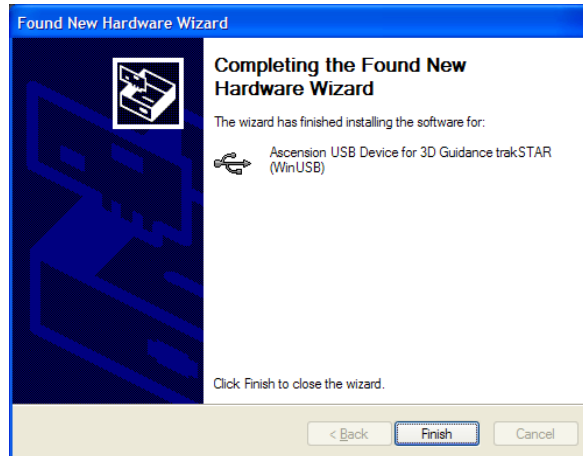


Figure 21 Found New Hardware (Closing Wizard)

System Checkout: Running the USB Demo Software

With the **trakSTAR 2** running and the drivers and utilities installed, you are ready to run the demo software and checkout the operation of your tracker. If you've elected to run with the RS232 interface, you can checkout the system using the driveBAY2 Utility. See [Appendix II : driveBAY 2 Utility](#) for details.

- Select **CUBES** from the **Ascension Program Group** in the **Windows® Start** menu to start the Demo Utility. After a brief pause (~20sec), the **CUBES** window displays.

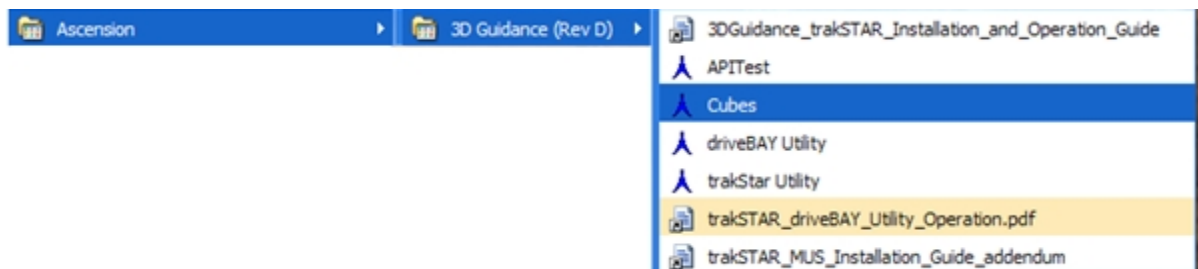


Figure 22 Location of Demo Utility (CUBES)

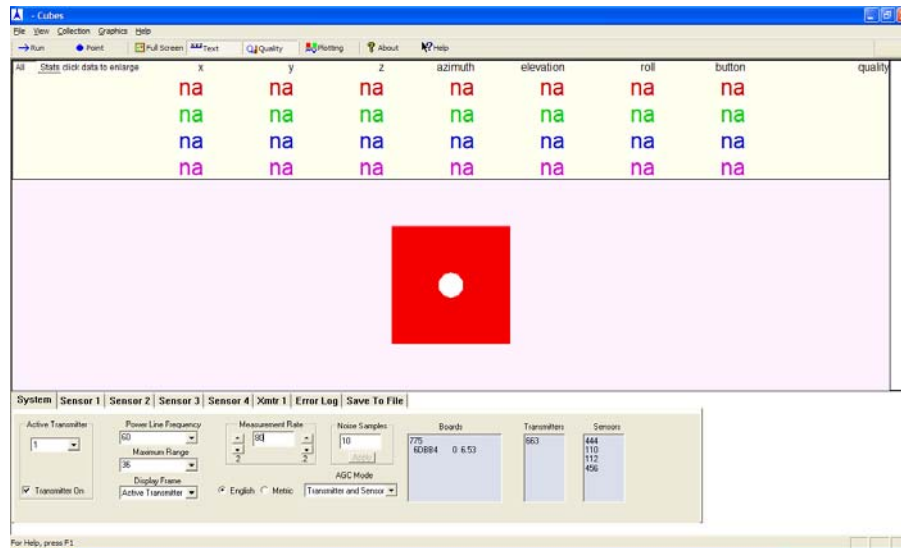


Figure 23 CUBES Window

2. Click **Run** to start collecting data from the system.



Figure 24 Location of CUBES Run Button

What you will see:

The top of the window displays the reported position and orientation data for each sensor attached to the tracker electronics unit. Each color-coded row contains outputs for a single sensor. The first three values in a row represent sensor position (in inches) relative to the transmitter's coordinate origin. The next three values in the row give sensor orientation in degrees, relative to the transmitter's default reference frame.

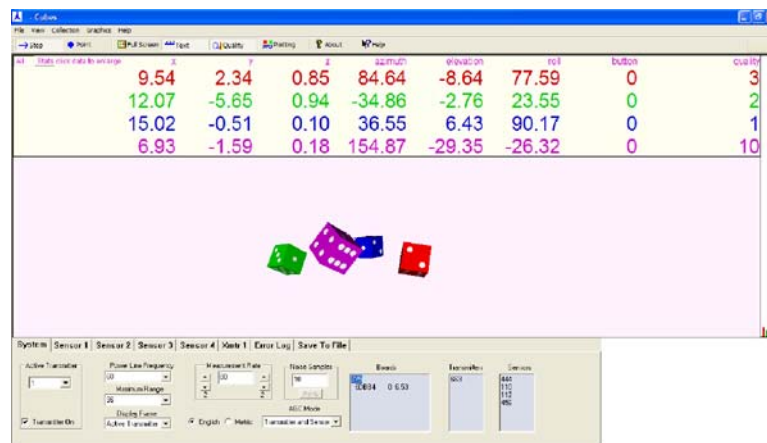



Figure 25 Typical Data Collection Position and Orientation Display

The **Button** column indicates the status of an external switch or contact closure that has been connected to the rear panel of the electronics unit. See the [Switch](#) rear panel connector description for details.


The last column in each row shows the reported QUALITY number. This value gives you an indication of the extent to which position and angle measurements are in error. Error may often be attributed to “uncompensated” metal in the environment. See the “[Performance Factors](#)” section of Chapter 3 for details.

 **Tip:** For additional information and assistance on using the QUALITY number, see [QUALITY](#) in Chapter 5.

The center of the window displays a colored cube for each sensor attached to the tracker. Note that cube colors correspond to the color of the numbers shown in the rows of sensor data.

The bottom of the window allows you to configure and adjust all system parameters.

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

 **Note:** While the demo utility is an excellent way to become familiar with the features of the tracker, it has some limitations as a data collection utility. For example, it is not possible to obtain the full update rate from the tracker (typical update rates are about 60Hz). If regular, consistent sampling and full update rates are required for your application, consider starting with the source code for our GetSynchronousRecordSample application, and modifying it to meet your application needs.

Configuration and Basic Operation

A method for configuring the trakSTAR 2 is presented in this chapter, along with a description of the basic operating characteristics and factors that affect performance.

When the trakSTAR 2 is configured at the factory, settings are optimized to meet the needs of most applications. However, you may find it helpful to customize the power-up behavior of the tracker to better meet your specific requirements.

Follow these steps to customize your tracker:

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

Default Configuration

The following settings are installed as power-up defaults. The driveBAY 2 Utility included on the CD-ROM may be used to alter this default power-up configuration for both the driveBAY 2 and the trakSTAR 2 tracking systems. Those settings not accessible with the driveBAY 2 Utility may be changed during normal operation, through the appropriate software command.

Note that settings are divided into 2 sections. These sections reflect the settings appropriate for the type of transmitter that is used.

- Dipole Settings (Table1): These are the default settings that the tracker will use when a Dipole transmitter is detected on power-up. Dipole transmitters include the **Short Range** (SRT), **Mid-Range** (MRT) and **Wide-Range** (WRT) transmitters.
- NonDipole Settings (Table2): These are the default settings that the tracker will use when a NonDipole transmitter is detected on power-up. NonDipole transmitters include the **5DOF/6DOF Flat** (Flat9) and the **5DOF/6DOF MAGnet** (MAG) transmitters.

Table 1 Power Up Default Dipole Settings:



| Feature | Setting | Utility | SW call |
|----------------------------|---|---------|---------|
| <u>System Settings:</u> | | | |
| Baud Rate: (RS232 only) | 115200 | | |
| Sync Mode: | Internal | ✓ | |
| Multi-Unit ID | 0 | ✓ | |
| Measurement Rate: | 80.0 Hz <div> Note: that the Available Update Rate for SRT, MRT, and WRT transmitters is always 3 times the measurement rate</div> | ✓ | ✓ |
| Scale: | SRT and MRT Systems: 36 inches WRT systems: 144 inches | ✓ | ✓ |
| Sleep on Reset: | Enabled | ✓ | |
| Full POST on Reset: | Disabled | ✓ | |
| Enable LED: | Enabled | ✓ | |
| Power Line Freq: | 60Hz | ✓ | ✓ |
| Report Rate: | 1 | ✓ | |
| Transmitter Settings: | | | |
| Reference Frame (deg): | az = 0, el = 0, rl = 0 | ✓ | ✓ |
| Enable XYZ Reference Frame | Disabled | ✓ | ✓ |
| Sensor Settings (per port) | | | |
| Hemisphere: | Front | ✓ | ✓ |
| Sensor Offsets (deg): | x = 0, y = 0, z = 0 | ✓ | ✓ |
| Angle Align (deg): | az = 0, el = 0, rl = 0 | ✓ | ✓ |
| Data Format: | POSITION/ANGLE | ✓ | ✓ |
| AC Wide Notch: | Enabled | ✓ | ✓ |
| AC Narrow Notch: | Disabled | ✓ | ✓ |
| Adaptive Filter (DC): | Enabled | ✓ | ✓ |
| Alpha Min: | 0.02 | ✓ | ✓ |
| Alpha Max: | 0.90 | ✓ | ✓ |
| Vm Table: | 2. 4. 4. 4. 4. 4. 4. 4 | ✓ | ✓ |

Table 2 Power Up Default NonDipole Settings:




| Feature | Setting | Utility | SW call |
|----------------------------|---|---------|---------|
| System Settings: | | | |
| Baud Rate: (RS232 only) | 115200 | | |
| Measurement Rate: | 42.0 Hz <div>  Note: The Available Update Rate for Flat9 transmitter is always: 42 x9 = 378 updates/sec Available Update Rate for MAGnet transmitter is always: 42x12 = 504 updates/sec </div> | ✓ | ✓ |
| Scale: | 36 inches | ✓ | ✓ |
| Sleep on Reset: | Enabled | ✓ | |
| Full POST on Reset: | Disabled | ✓ | |
| Enable LED: | Enabled | ✓ | |
| Power Line Freq: | 60Hz | ✓ | ✓ |
| Report Rate: | 1 | ✓ | |
| Transmitter Settings: | | | |
| Reference Frame (deg): | az = 0, el = 0, rl = 0 | ✓ | ✓ |
| Enable XYZ Reference Frame | Disabled | ✓ | ✓ |
| Sensor Settings (per port) | | | |
| Algorithm | 6DOF | ✓ | |
| Hemisphere: | NonDipole transmitters only operate in FRONT hemisphere | | |
| Sensor Offsets (deg): | x = 0, y = 0, z = 0 | ✓ | ✓ |
| Angle Align (deg): | az = 0, el = 0, rl = 0 | ✓ | ✓ |
| Data Format: | POSITION/ANGLE | ✓ | ✓ |
| AC Wide Notch: | Enabled | ✓ | ✓ |
| AC Narrow Notch: | Disabled | ✓ | ✓ |
| Adaptive Filter (DC): | Disabled | ✓ | ✓ |

| | | | |
|------------|--------------------------|---|---|
| Alpha Min: | 0.02 | ✓ | ✓ |
| Alpha Max: | 0.90 | ✓ | ✓ |
| Vm Table: | 400, 1, 1, 1, 1, 1, 1, 1 | ✓ | ✓ |

Configurable Power-up Settings:

Here is an explanation of the power-up settings that can be re-configured using the driveBAY 2 Configuration Utility.

Table 3 Configurable Power Up Settings

| Setting | Description |
|------------------|--|
| Baud Rate | <p>Establishes the default RS232 communication rate for power-up. Only supported Baud Rate for the system is 115200.</p> <div>  Note: Currently the only supported Baud Rate for RS232 communication is 115200. </div> |
| Sync Mode | <p>This setting establishes the mode the tracker should use for synchronization of data acquisition. Note that the Internal mode is used for both stand-alone and multi-unit configurations, where one of the tracker units establishes the data synchronization signal internally.</p> |
| Multi-Unit ID | <p>In multi-unit tracker configurations (i.e. 5-16 sensors), this setting lets the tracker know how the 4 attached sensors should be enumerated for the 3DGuidance API. Sensors connected to the unit with an ID of 0 will be enumerated 0 through 3, sensors connected to the unit with an ID of 1 will be enumerated 4 through 7, etc. See the Multi-Unit Sync Installation Guide Addendum (included with the installation) for details.</p> <div>  Note: For Multi-Unit Sync (MUS) Operation, be sure to read the MUS Installation Guide Addendum, included in the installation. </div> |
| Measurement Rate | <p>Sets the acquisition rate for the system. This can be altered to optimize susceptibility to distortion from certain metals. See Performance Factors section below.</p> |
| Scale | <p>Sets the scale factor used by the trakSTAR 2 to report the position of the sensor with respect to the transmitter. Valid values of 36, 72, and 144 represent the full-scale position output <u>in inches</u>.</p> <div>  Note: Wide Range transmitters are calibrated to use the Scale setting of '144'. </div> |

| Setting | Description |
|------------------------------|--|
| Sleep on Reset | When enabled, this setting will cause the trakSTAR 2 to enter SLEEP mode after completing a reset or following power-up. In the SLEEP mode, the transmitter is turned off, but the unit will continue to respond to commands. Issue the RUN command (or 3DGuidance API equivalent) to resume normal operation. See the SLEEP command description in the RS232 Command Reference section for details. |
| Full POST on Reset | <p>When enabled, this setting will cause the tracker to run through the extended Power On Self Test (same POST that the tracker normally executes on power-up) every time the tracker is reset. By default, the tracker will perform a reset for every call to InitializeBIRDSysyem (Note that this assumes that default ATC3DG.ini setting has been left as: reset = yes. See the System Parameter RESET for details). For Dipole Transmitter configurations, this ‘Cold-start’ reset with extended POST takes about 18 seconds to complete for a single unit. The ‘warm-start’ reset that results with this setting disabled (default) takes about 6 seconds.</p> <p>The reset for ‘programmed’ Non-Dipole Transmitter configurations takes about 12seconds. When a Non-Dipole Transmitter and electronics unit are paired for the very first time, this time extends up to 2 min. See the LED Definitions table in Chapter 7 for more information</p> |
| Enable LED | By default, the tracker will illuminate/flash the front panel LED to help indicate status. Disabling this setting will cause the LED to be turned off as soon as the initial POST has completed. |
| Power Line Freq | Sets the frequency (in Hertz) of the AC Power Source that the tracker should assume is in use. Valid values include 60 and 50Hz. |
| Report Rate | Output rate divisor setting. Reduces the number of records output during STREAM mode (also set by the GetSynchronousRecord call), to that determined by the setting. Default setting of 1 makes all outputs available. |
| Reference Frame | The default trakSTAR 2 Reference Frame is defined by the Transmitter’s X, Y, and Z axes (see Figure 26). This Reference Frame setting allows you to enter the angles required to mathematically align the axes of the Transmitter to those of a new reference frame. |
| Enable (XYZ) Reference Frame | The angles entered in the Reference Frame fields will only specify a new reference frame for measuring orientation angles. If you need the XYZ position measurements to correspond to the new reference frame then enable this setting. |

3D GUIDANCE TRAKSTAR 2™ INSTALLATION AND OPERATION GUIDE

| Setting | Description |
|-----------------|---|
| Algorithm | When using a non-dipole (flat) transmitter, each sensor port can either track a single 6DOF sensor or up to three 5DOF sensors. The tracking mode is chosen at startup based on this setting. There are three options: 6DOF - the port always computes a 6DOF tracking solution; 5DOF - the port always computes a 5DOF tracking solution; Auto - the port will configure itself at startup based on what sensor is plugged into the port. A 6DOF sensor will cause the port to be configured for 6DOF tracking. A 5DOF sensor or no sensor will cause the port to be configured for 5DOF tracking. |
| Hemisphere | This parameter is used to tell the trakSTAR 2 in which hemisphere, centered about the transmitter, the sensor will be operating. For Dipole transmitter types, there are six hemispheres from which to choose: the FRONT (forward), BACK (rear), TOP (upper), BOTTOM (lower), LEFT, and the RIGHT. If no HEMISPHERE parameter is specified, the FRONT is used by default. NOTE: NonDipole transmitters only support the FRONT hemisphere. |
| Sensor Offsets | Default position outputs from the trakSTAR 2 represent the X, Y, and Z position of the magnetic center of the sensor coil (approximate center of sensor housing) with respect to the transmitter origin. The Sensor Offsets allow you to configure the position outputs such that the tracker is reporting the position of a location that is offset from the center of the sensor. See the Sensor Parameter type SENSOR OFFSETS (or the OFFSET RS232 command) for details. |
| Angle Align | The Angle Align settings allow you to mathematically align the sensor's coordinate frame to the coordinate frame of the object being tracked. This is beneficial if you find that, when a sensor(s) is mounted to the object you are tracking, the angle outputs are not zero when in the normal 'resting' position. |
| Data Format | Sets the default data format for returned data records. |
| AC Wide Notch | The AC WIDE notch filter refers to an eight-tap finite impulse response (FIR) notch filter that is applied to the sensor data to eliminate sinusoidal signals with a frequency between 30 and 72 hertz. |
| AC Narrow Notch | The AC NARROW notch filter refers to a two tap FIR notch filter that is applied to signals measured by the tracker's sensor. Use this filter in place of the AC WIDE notch filter when you want to minimize the delay between trakSTAR 2 's 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. |
| Adaptive Filter | When set to ON, an adaptive low pass filter is 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 filter is ON, you can modify its noise/lag characteristics by changing ALPHA_MIN, ALPHA_MAX and Vm. See the Interface Reference Chapters for details. |

| Setting | Description |
|---------------|---|
| Kalman Filter | These parameters are used to tune the static and dynamic tracking performance when using non-dipole transmitters. These settings are to be altered by the factory only. If you think that tuning these parameters may be appropriate for your application, please contact us first. |

Default Reference Frame

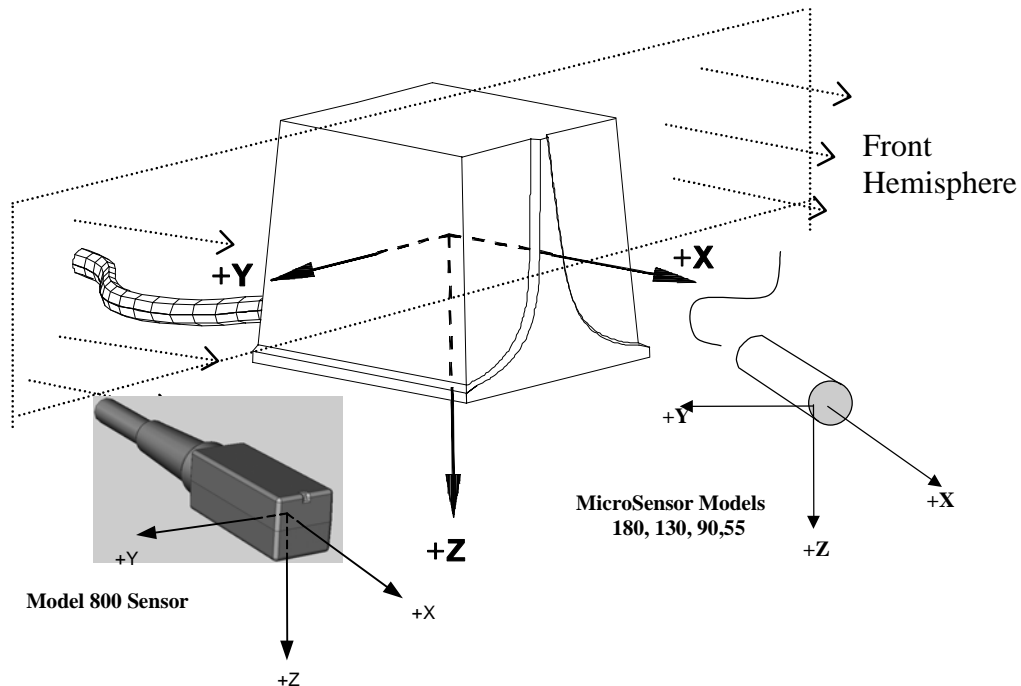


Figure 26 Default Transmitter/Sensor Coordinate Frames

NOTE: The origin of the Mid, Short, and Wide-Range Transmitters' default Reference Frames are defined to be at a location approximately in the center of the transmitter coil set. This location is approximately (with respect to the housing surfaces):

Mid-Range:

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

Short Range: (cable exits lower rear face) -

- 1.1 " from bottom front edge / 1.51" from bottom rear edge
- 0.90 " from the top face
- 1.02" from bottom left edge

Wide Range: (cable exits lower rear face) -

- 6" from bottom front edge / " from bottom rear edge
- 6" from the top face
- 6" from bottom left/right edge

Flat Transmitter

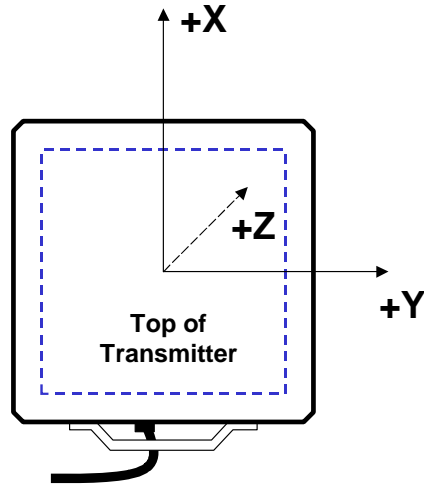


Figure 27 Flat Transmitter Coordinate Frame

5DOF/6DOF Axis Flat Transmitter Origin:

In the middle of the square that defines the top surface

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

MAGnet Transmitter

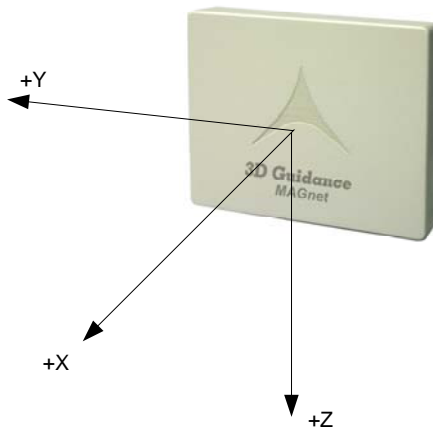


Figure 28 MAGnet Transmitter Coordinate Frame

5DOF/6DOF MAGnet Transmitter Origin:

In the middle of the square that defines the top surface

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

Changing Your Settings

The *driveBAY2 Utility* may be used to alter power-up defaults for driveBAY 2 and trakSTAR 2 systems. See [Appendix II: driveBAY 2 Utility](#).

Mounting the Hardware

Mid-Range Transmitter Mounting

Mount the mid and short range transmitters on a non-metallic surface such as wood or plastic, using non-metallic bolts or 300-series stainless steel bolts. For optimum performance, keep the transmitter at least twenty-four inches away from the electronics unit.

The mid-range transmitter's mounting holes are not strong enough to support the transmitter's weight if it is mounted upside down. If you choose to mount the transmitter upside down, use hardware that firmly holds the flanges along both sides of the transmitter in addition to bolting the two mounting holes.

Never mount the transmitter on the floor (including concrete), ceiling, or walls. Often these surfaces contain hidden metal objects.



Note: The Mounting holes are not strong enough to hold the Mid-Range Transmitter upside down.

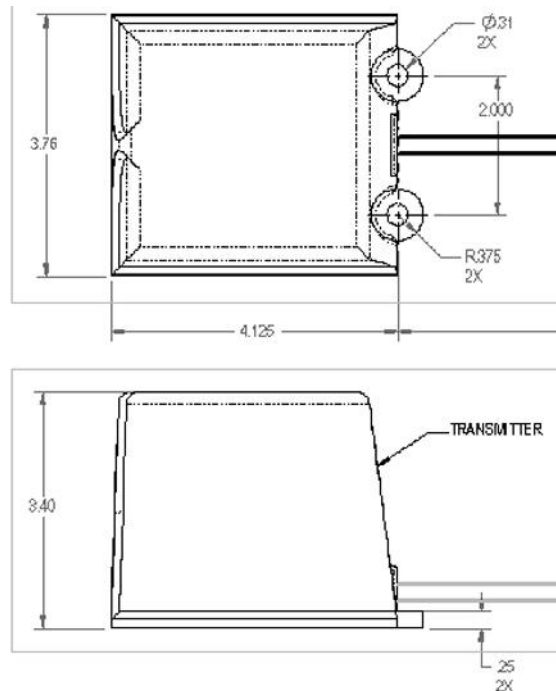


Figure 29 Mid-Range Transmitter Mounting Dimensions (inches) -Top and Side Views

Short-Range Transmitter Mounting

Mount the short-range transmitter on a non-metallic surface such as wood or plastic, using non-metallic or 300-series stainless steel screws.

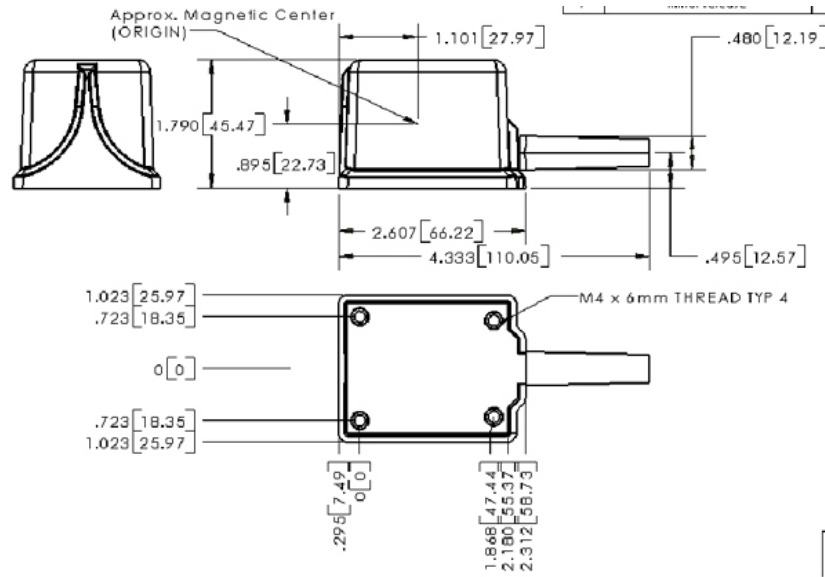


Figure 30 Short-Range Transmitter Mounting Dimensions (inches) Top and Side Views

The mounting provisions for the short-range transmitter are located in the base of its housing. The four threaded inserts molded into the base material, accept an M4 threaded screw. Never mount the transmitter on the floor (including concrete), ceiling, or walls. Often these surfaces contain hidden metal objects.

Wide-Range Transmitter Mounting

Usually the WRT is mounted on a pedestal in the center of the motion capture space, or mounted under the floor of a wooden stage. Because the transmitter is fairly heavy (~37lbs), fragile, and subject to performance degradation by nearby metal, the method that you use to support the transmitter must be strong and non-metallic. Small amounts of metal in the mount such as steel bolts are acceptable. Supporting the transmitter on a steel or aluminum framework is not acceptable. We recommend wood, structural fiberglass, or laminated phenolic for mounting materials. Two bolt holes in the bottom of the transmitter have been provided for maintaining the alignment of the transmitter to your support. The alignment bolt threads inside the bottom of the transmitter are 10-24. Thread engagement will occur ~1¾ inches into the base. The bolt should be screwed in and additional ½ inch for full engagement, but not more than this.



NOTE: THE BOLT HOLES ARE NOT STRONG ENOUGH TO SUPPORT THE WEIGHT OF THE TRANSMITTER. Do not use them to support or 'tie down' the transmitter to your mount.

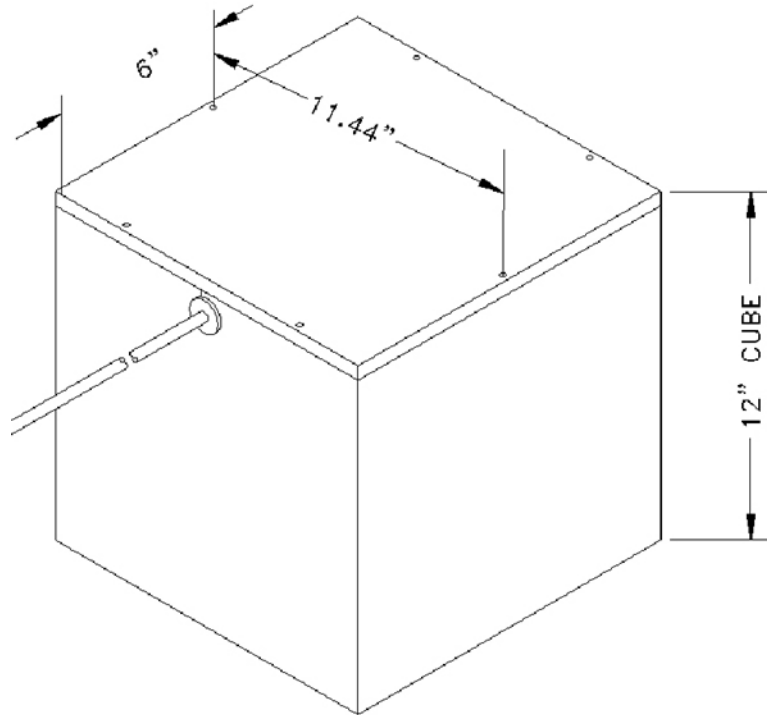


Figure 31 Wide-Range Transmitter Mounting Dimensions

Flat Transmitter Mounting

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

MAGnet Transmitter Mounting

Mount the MAGnet transmitter to a non-metallic surface such as wood or plastic, using non-metallic bolts or 300-series stainless steel bolts. For optimum performance, keep the transmitter at least twenty-four inches away from the electronics unit.

Sensor Mounting

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

The cables used in the Microsensors (Models 55, 90, 130 and 180) are designed for integration into the devices and cables of many OEM products. If using microsensors and cables independently, treat them carefully. Repeated bending, pulling, or yanking of the cable will result in damage and failure to track.

Model 800 sensors (8x8 mm) should maintain a minimum center-to-center separation distance of 1.3" (33mm). Positioning sensors too close to each other can cause distorted measurements.

Model 130 and Model 180 sensors should maintain a minimum center-to-center separation distance of 0.47" (12mm).

Adequate strain relief should be provided for the microsensors. The junction between the sensor head and the cable is a potential failure point and bending at this junction should be prevented.

If the cable is allowed to flex or bend through a large angle at the sensor, then **the sensor will break**. Restrain the cable near the sensor head to eliminate this problem.



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

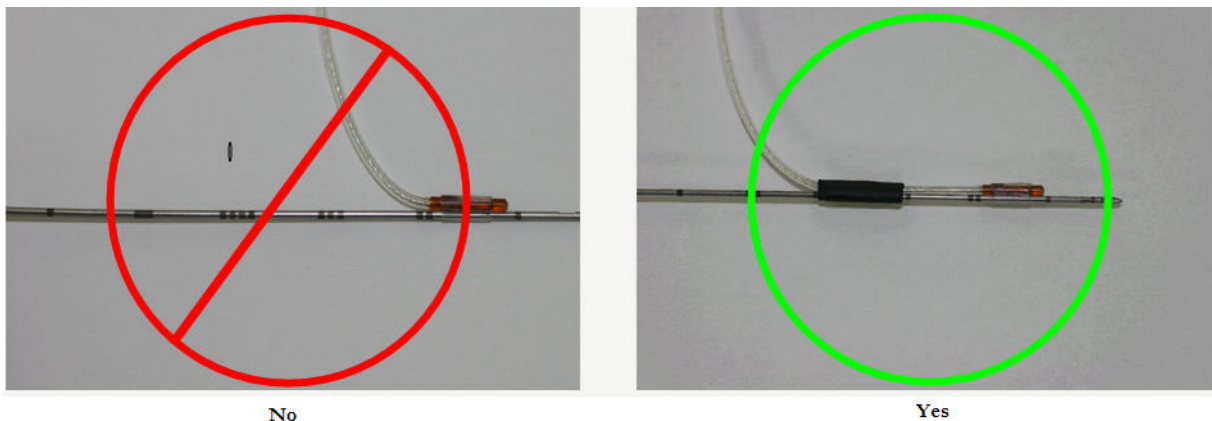


Figure 32 Cable Mounting Requirements

Electronics Unit Mounting

Mount the electronics unit on a secure flat surface such as a table or desk. If placed on a rolling cart or shelf, be sure to secure the unit with straps or fasteners, such that a tug on the cables will not allow the unit to fall on the floor.

For best performance, mount the electronics unit **at least 24 inches away** from the transmitter.






Rear Panel Connectors



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



Figure 33 trakSTAR 2 Rear Panel Connectors

Table 4 trakSTAR 2 Rear Panel Connections

| Name | Description |
|--------|---|
| Power | Standard 3-prong AC power inlet. Accepts power from 100 to 240V, at 50/60Hz. |
| RS232 | <p>This is one of the two communication interfaces for the trakSTAR 2. A pin-out and signal description of the RS-232C interface is included in Chapter 6. Note that the tracker requires connections only to pins 2, 3 and 5 of the 9-pin interface connector.</p> <div style="border: 1px solid black; padding: 5px; margin-top: 10px;">  Note: See the RS232 Signal Description in Chapter 6. </div> |
| USB | <p>The USB 2.0 connector is a standard USB TYPE B -Female for peripheral devices.</p> <p>PIN: 1: VCC +5VDC</p> <p>PIN: 2: DATA-</p> <p>PIN: 3: DATA+</p> <p>PIN: 4: GROUND</p> <div style="text-align: center; margin-top: 10px;">   </div> |
| Switch | <p>The SWITCH connector may be used as an input from any analog switch. The contact closure (such as a button or footswitch) should be connected between the center conductor and the barrel of a mating BNC connector, and exhibit less than 50 ohms total when in the closed position. To be correctly detected and reported by the unit, the switch must be closed for not less than 17mS.</p> <div style="text-align: right; margin-top: 10px;">  </div> <div style="border: 1px solid black; padding: 5px; margin-top: 10px;">  Note: See the API reference section for data formats that support the Button </div> |

| Name | Description |
|-----------|---|
| Unit Sync | <p>This connector is used in Multi-Unit Sync (MUS) configurations, for synchronizing the data acquisition across multiple electronics units. Up to 4 electronics units can be synchronized, providing tracking for up to 16 sensors simultaneously. See the MUS Installation Guide Addendum for details on the setup and checkout of these MUS configurations.</p>  <div data-bbox="527 516 1448 609">  Note: See the <i>MUS Installation Guide Addendum</i> for details on setting up Multi-Unit Sync Operation. </div> |

Basic Operation

The tracker determines six degrees-of-freedom (6DOF) position and orientation (X, Y, Z, Azimuth, Elevation, and Roll) of one or more sensors referenced to a fixed transmitter. In five degree of freedom (5DOF) sensor tracking, the Roll angle of the orientation is omitted. The transmitter sequentially generates magnetic fields and the sensor instantly measures the transmitted field vectors at a point in space. From theoretical knowledge of the transmitted field, the tracker accurately deduces the real-time location of the sensor(s) relative to the transmitter. The **trakSTAR 2** tracker is designed to work with two basic types of transmitters: Dipole and NonDipole.

Dipole Transmitters

Three Dipole transmitter options are available with **trakSTAR 2**.

The Short-Range, Mid-Range, and Wide-Range transmitters consist of a high permeability core with three concentric sets of coils, each coil having an axis at right angles to the other two. Magnetic fields along the X, Y, and Z axes of the transmitter are created when current flows in their respective windings. The strength of the magnetic field is highest near the transmitter and falls off with the cube of distance from the transmitter.

The transmitted field of a given axis has a trapezoidal magnitude characteristic as a function of time. Each of the three coils in the dipole transmitter are sequentially energized in this manner during each measurement cycle.

Only 6DOF sensors can be used with dipole transmitters.

NonDipole Transmitters

Non-dipole transmitters contain 9 (Flat) and 12 (MAGnet) coils arranged in complex geometries within their respective housings. They employ advanced tracking algorithms to track sensors in the volume above (in front of) their surfaces. Algorithms and proprietary construction techniques in the 5DOF/6DOF Flat are designed to shield the tracking volume from magnetic distorters below the transmitter and prevent performance degradation. The MAGnet construction does NOT provide this shielding.

Both 6DOF and 5DOF sensors can be used with all non-dipole transmitters.

6DOF Sensor

A 6DOF sensor uses three receiving coils to measure and provide a tracking solution that includes the position in three dimensions and the orientation of the three sensor axes relative to the tracker reference frame. 6DOF sensors are factory-calibrated and the calibration data is stored on a memory chip in the sensor's connector housing. This calibration information is read by the electronics unit upon power-up, and when a hot-swap of a sensor into a sensor port is detected.

5DOF Sensor

A 5DOF sensor uses a single receiving coil to measure the position in three dimensions and the pointing direction of the sensor relative to the tracker reference frame. The roll of a 5DOF sensor is not tracked. 5DOF sensors utilize a proprietary run-time calibration. Identifying data such as part

number and serial number are stored in a memory chip in the sensor's connector housing. Because of the simplicity of their design, 5DOF sensors can be fabricated in much smaller form factors than 6DOF sensors.

Electronics Unit

In addition to computing tracking solutions, the electronics unit, packaged in a desktop enclosure with built-in power supply, contains the transmitter drive circuitry, sensor signal processing, power conditioning, and host interface functions.

The transmitter drive is a precision current source, with a maximum output of 3.0A. The system detects the absence of a transmitter by monitoring the current. If current is interrupted, the transmit driver will turn off until a valid transmitter EEPROM is detected. This ensures that the connector is de-energized when open. Also, the transmitter is fault-protected for ground shorts. In the event of a short to ground on any transmitter pin, no damage will result to the tracker and no excessive current hazard conditions will occur.

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

The electronics unit contains up to six onboard processors:

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

Measurement Cycle

The tracking system activates transmitter coils sequentially and produces a data record (i.e. full tracking solution) following each coil measurement. Once each transmitter coil has been activated, a system measurement cycle is complete and a new cycle begins. Thus, a dipole (3-coil) transmitter running at measurement rate of 50 Hz will compute 150 tracking solutions per second.

 **Note:** The Update rate available from the tracker with a dipole transmitter is always 3 times the system Measurement Rate. See [Appendix III: Application Notes](#) for additional details.

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

Calibration

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

Calibration allows components to be interchanged with minimal effect on the resulting tracking values. The calibration values for each component are stored within the hardware of the individual component. For the Transmitter and Sensor, the values are stored in an EEPROM at the connector. For the Electronics unit, the calibration values are stored in an EEPROM mounted on the printed circuit board. Non-dipole Transmitters include an additional Flash memory chip in the connector.

The calibration data in each component is programmed at the factory and is read-only.

Performance Factors

Electromagnetic and Other Interference in Tracking

Other electrical and magnetic devices sharing the immediate near volume with the Ascension tracking device may influence tracking data.

These two factors may affect the stability of the tracking area and thus the tracking accuracy:

- Excessive electrical noise
- Magnetic distortion

Noise

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

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

Causes of Noise

There are two types of noise in tracking data, generated from both internal and external sources. The larger of these are external sources. External sources of electrical noise may be generated by electrical motors, switching power supplies, fluorescent lighting, video CRT monitors, Uninterruptible Power Supplies, and wiring or devices which use or carry large amounts of current that vary over time.

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

Internal sources include small variations in measurement timing, amplified electronic component thermal activity, algorithm division by very small numbers, electrical power line noise which is not fully suppressed, and other sources.

Reducing Noise

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

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

Distortion

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

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

Causes of Distortion

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

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

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

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

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

Reducing Distortion

Metal is the primary cause of distortion. Removal and reduction of the amount of metal in and around the tracking system is the most effective strategy. Sources of metal distortion cannot be shielded, as is often the case with noise sources.

Many alternatives to metal exist. Structural fiberglass, plastics, woods, and ceramics are just a few of the nonmetal alternatives available for your use. Of the metals available, nonmagnetic and high electrically resistive metals are the next desirable. Some alloys of stainless steel (medical grade) can be used with the tracker resulting in no or minimal distortion. Brass and aluminum are less desirable and are not recommended, but they may be used in some situations. Care must be observed, as machined metal

may become magnetic. All metal should be tested with the tracker before being incorporated into a product or protocol that employs magnetic tracking.

We recommend that you always try to eliminate and reduce the presence of metals in your immediate tracking area. Since the tracker is not a line-of-sight device, care must be taken that the entire area around the transmitter is considered with regard to metal. The area behind and under the transmitter should be examined and modified as closely as the area between the sensor and the transmitter.

Metal Detection

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

Quality/Metal Number

Alternatively, you may wish to experiment with the use of the QUALITY number computed by the tracker. Also referred to as the METAL error or Distortion number, this returned value gives you an indication of the extent to which the position and angle measurements are in error. See the [Quality Sensor Parameter Type](#) for details.

Tracker as the Cause of Interference

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

Factors in Tracker Accuracy

Warm-up

As with most electrical components, the tracker goes through a period of drift before it reaches a thermal equilibrium. **TrakSTAR 2** is designed to meet accuracy specifications within five minutes of power up. For effective warm up, the system must be active.

A previously unused sensor may be swapped without concern of sensor warm-up or drift.

Measurement Rate

Usually you will track motions using our default measurement rate. However there are times when you may wish to adjust it higher or lower. Here are a few examples:

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

Equipment Alteration

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

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

Power Grid Magnetic Interference

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

Advanced software filters are implemented in the **trakSTAR 2** to reduce this effect without compromising dynamic performance, but the power line is still a potential source of tracker noise. . Highest performance will be achieved by operating the system away from walls, floors, or other structures in which electrical wiring is routed. Also, high current devices such as heaters, motors, and transformers should be kept as far away from the system as practical. The filters used to reduce power grid magnetic interference require the correct power line frequency value to operate effectively. Make sure you input the correct frequency to the **trakSTAR 2** through the API.

Performance Motion Box

All tracking components are subjected to a calibration procedure that optimizes performance over a given region. This region is referred to as the **Performance Motion Box**

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

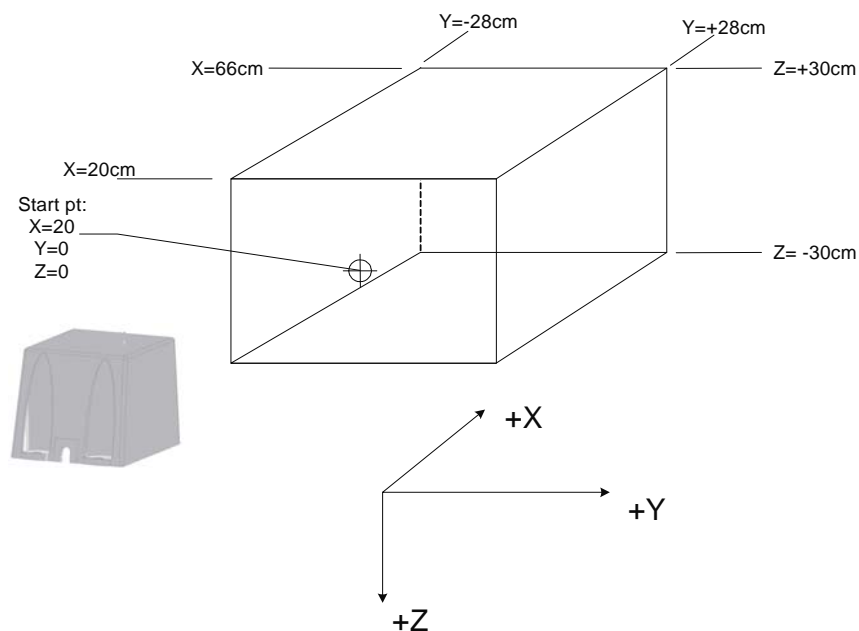


Figure 34 Performance Motion Box – MRT and Model 800 (8 mm) Sensors

Dimensions in each axis:

- For the Mid-Range transmitter:

For **Model 800** sensors:

X = 20 to 66cm from the transmitter center

Y = ±28 cm from the transmitter center

Z = ±30 cm from the transmitter center

For **Model 180** sensors:

X = 20 to 51cm from the transmitter center

Y = ±23 cm from the transmitter center

Z = ±15 cm from the transmitter center

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

X = 20 to 36cm from the transmitter center

Y = ±20 cm from the transmitter center

Z = ±20 cm from the transmitter center

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

For **Model 55*** sensors:

X = 16 to 32cm from the transmitter center

Y = ±10 cm from the transmitter center

Z = ±10 cm from the transmitter center

*Note –Recommended max range for the Model 55 sensor with the mid-range transmitter is 25cm (10")

- For the Short-Range Transmitter:

For **Model 800** sensors:

X = 15 to 41cm from the transmitter center

Y = ±12 cm from the transmitter center

Z = ±12 cm from the transmitter center

Microsensors not recommended with Short Range Transmitter

- For the Wide-Range Transmitter:

For **Model 800** sensors:

X = 91 to 137cm from the transmitter center

Y = ±28 cm from the transmitter center

Z = ±30 cm from the transmitter center

Microsensors not recommended with Short Range Transmitter

- For the MAGnet Transmitter:

For **Model 800,180, and 130** sensors:

X = 10 to 41cm from the transmitter center

Y = ±20 cm from the transmitter center


Z = ±20 cm from the transmitter center

Flat Transmitter Motion Box

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

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

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

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

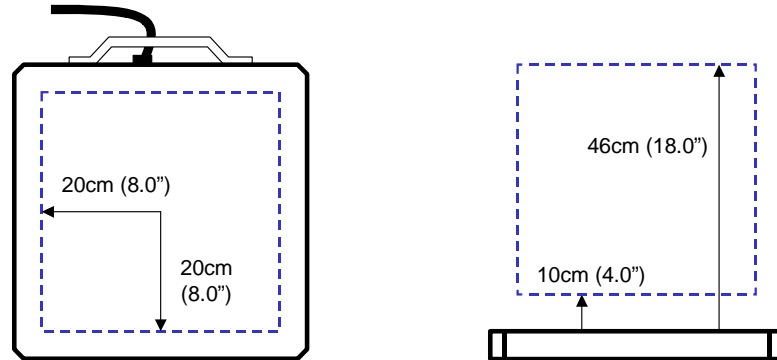


Figure 35 Performance Motion Box – Flat Transmitter

Software Operation - Tools for Successful Tracking

This chapter outlines the interfaces and methods for communicating with the 3DGuidance trackers and lists the sample programs available for the USB and RS232 interface protocols.

Software Overview

Communication with the 3DGuidance trackers can occur using either of two Ascension interface protocols: 3DGuidance Windows API, and the Ascension RS232 Interface.

3DGuidance Windows API

You can communicate with the tracker via USB using our 3D Guidance Windows interface. This interface is standard in our legacy magnetic tracking products and has been continued for the new generation of 3D Guidance trackers.

If you are new to the 3DGuidance trackers and the 3DGuidance API, you will find several tools included on the CD-ROM to assist in experimenting with the tracker's capabilities and in quickly creating custom code. These tools include two demo applications, CUBES and APITest, and three utility programs containing sample C++ project files. The tables below describe both the tools available and where they can be found following software installation. See Chapter 5 [3DGuidance API Reference](#) for full details.

Table 5 Location of API Files and Demo Applications

| 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\Ascension\3D GuidanceXXXX\3D G API Developer\3DG API |
| ATC3DG.lib | Library file required during compiling of any code that makes calls to the API | |
| ATC3DG.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. | |

| API Components | Description | Location (After software installation) |
|--------------------|---|--|
| <i>Cubes.exe</i> | <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> <ul style="list-style-type: none"> • TX On/off -turn the transmitter on or off • System RESET - reset/re-initialize the tracker • System settings - change measurement rate, line frequency, AGC mode, English vs. metric units • Sensor settings- change quality parameters, angle align, and filter settings • Transmitter - change reference frame • Graph Mode - Plot up to 3 parameters • Noise Statistics - Collects X samples and shows AVG, pk-to-pk and RMS deviation <p><i>NOTE: Cubes is a demo program and should NOT be used when fast data collection is required.</i></p> | <p>Program Files\Ascension\3D GuidanceXXXX\Applications\ Cubes.exe</p> |
| <i>APITest.exe</i> | <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> <ul style="list-style-type: none"> • Check response to particular command without implementing it in their original code. • Check response using a particular non-default setting (filter, etc.) without implementing in their code. • Confirm/Debug hardware and their code by reproducing chosen settings and viewing response. • Save system settings that have been tried to an .ini file (using SaveSystemConfig call). <p>NOTE: It's important to know how commands are defined and the sequence of commands to send, for this tool to be used effectively.</p> | <p>Program Files\Ascension\3D GuidanceXXXX\Applications\ APITest.exe</p> |

Table 6 Location of Sample Utility Programs

| Sample Code | Description | Location |
|----------------|--|---|
| <i>Sample</i> | <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> <ul style="list-style-type: none"> Initializes the Tracker <ul style="list-style-type: none"> Reads in the system configuration (status of board, sensors, tx, etc.) Turns on the transmitter. If all above is valid, it then collects 100 records (POS/ANG format) from each of the sensors and streams these records to the screen. Error Handler - A simple error handler is included. It takes any error codes returned from the DLL/Tracker, and sends them to the screen. TX Off - Before closing the application, shows how to turn off the TX. <p>All necessary source files to re-build and run the application are included, and each of the above steps is accompanied by detailed comment descriptions directly in the code.</p> | <p>Program Files\Ascension3D GuidanceXXXX\API Developer\Samples\ Sample\ Sample.vcproj</p> |
| <i>Sample2</i> | <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.</p> <p>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 is accompanied by detailed comment descriptions directly in the code.</p> | <p>Program Files\Ascension3D GuidanceXXXX\ API Developer \Samples\ Sample2\Sample2.vcproj</p> |

| Sample Code | Description | Location |
|--|--|---|
| <i>GetSynchronousRecordSample</i> | <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.</p> <p>Specifically, it:</p> <ul style="list-style-type: none"> • Initializes the Tracker • Reads in the system configuration (status of board, sensors, tx, etc.), and sets the measurement rate to the maximum supported for the tracker. • Turns on the transmitter • If all above is valid, collects 10000 records (POS/ANG/TimeStamp format) from each of the attached sensors and streams these records to a file. The status of the sensors is queried after each record is received. • Error Handler - A simple error handler is included. It takes any error codes returned from the DLL/Tracker, and sends them to the screen. <p>TX Off - Before closing the application, shows how to turn off the TX</p> <p>All necessary source files to re-build and run the application are included, and each of the above steps is accompanied by detailed comment descriptions directly in the code.</p> | <p>Program Files\Ascension\3D GuidanceXXXX\ API Developer \Samples\ GetSynchronousRecord Sample\ GetSynchronousRecordSa mple.vcproj</p> |

Ascension RS232 Interface

Communicating with **the** tracker via RS232 can be done using the Ascension Technology RS232 Interface protocol. If you are familiar with the legacy Flock of Birds products, you will find the **trakSTAR 2** is compatible with most applications written using this protocol. Using the protocol gives instant access to a wide range of motion capture, virtual reality, and biomechanics applications.

For users wishing to access the tracker via this RS232 interface, there are two options:

1. Use an existing driver for the legacy protocol (i.e. Bird.dll - Win32)
2. Communicate directly via the ATC RS232 Interface protocol

Documentation and source files for these options can be downloaded directly from the Ascension website: <http://www.ascension-tech.com/technical/index.php>

RS232 Windows Driver

The RS232 Windows driver (Bird.dll) provides an interface to the Ascension Technology trackers for WIN32 applications. With the exception of the initialization routines, this interface is standardized across many platforms and devices. The following Ascension Technology products are supported by this driver: Flock of Birds, pcBIRD, miniBIRD, MotionStar, MotionStar Wireless, laserBIRD, phasorBIRD, and 3DGuidance systems. To use the driver, simply include the header file Bird.h in your

source code, link with the library module Bird.lib, and put the dynamic-link library Bird.dll anywhere on the DLL search path. Definitions for the data structures and API calls documented here are in the file bird.h.

The Windows Driver User Manual and driver files can be downloaded from the Technical Support page of the Ascension website: <http://www.ascension-tech.com/technical/index.php>

Direct Communication: ATC RS232 Interface Protocol

The RS232 Interface protocol involves interpreting RS232 commands sent directly to the tracker. The command set supported by this protocol are detailed in [Chapter 6: Ascension RS232 Interface Reference](#).

3DGuidance API Reference

This chapter is designed to show you how to write an application that will access the tracker using the 3DGuidance API that is provided in the ATC3DG.dll. This chapter also describes the setup of the tracker and defines various parameters.

Using 3DGuidance

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

[Quick Reference](#)

[Pre-Initialization Setup](#)

[System Initialization](#)

[System Setup](#)

[Transmitter Setup](#)

[Sensor Setup](#)

[Acquiring Position and Orientation Data](#)

[Error Handling](#)

Quick Reference

SYSTEM

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

- Transmitter: No transmitter selected
- Power Line Frequency: 60.0 (Hz)
- AGC Mode: 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 downsampling of the GetSynchronousRecord() 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)

- **TRANSMITTER**

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

- **POWER LINE FREQUENCY**

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

- **AGC_MODE**

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 [SENSOR_AGC_ONLY](#)). In this mode the firmware will



Note:

Transmitter AGC not used in 3DGuidance systems.



Note:

Transmitter AGC not used in 3DGuidance systems.

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 system sampling rate. 3DGuidance trackers running Mid-Range transmitters are nominally set at 80.0 measurements/second. You can increase the measurement rate to a maximum of 255 measurements/sec for these configurations. 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() API. 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. In some cases an application may not be able to keep up with the streaming bandwidth. 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 3rd, etc.

- **POSITION_SCALING**

This represents the scale factor for position data returned as signed binary integers. The position scaling can be set to 36, 72, or 144 (inches). It is referred to in the documentation as the [MAXIMUM_RANGE](#) parameter. The value set will be the maximum possible full-scale value returned by the system. 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 set up can be performed individually for each sensor. The parameters can be set and read by making calls to [SetSensorParameter](#) and [GetSensorParameter](#). Upon power up with a **DIPOLE** transmitter (provided power-up defaults have not been configured differently), each sensor channel is setup with the following **defaults**:

- 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 **Alpha max** table = **0.9000**, all values in **alpha min** table = **0.0200**, **Vm** table values = **2, 4, 4, 4, 4, 4, 4**
- 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 BUTTON RECORD](#),
[DOUBLE POSITION MATRIX TIME Q BUTTON RECORD](#),
[DOUBLE POSITION QUATERNION TIME Q BUTTON RECORD](#)

These data formats can be set up by using the [SetSensorParameter](#) command for each individual sensor.

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

- **ANGLE ALIGN**

Aligns sensor to reference direction. These parameters can be set up for each individual sensor by using the [SetSensorParameter](#) command. The current setting of ANGLE ALIGN can be accessed using the [GetSensorParameter](#) command. 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 but its operation can be modified by changing the values in 3 tables, which are contained in the [FILTER ALPHA PARAMETERS](#). These parameters are changed through use of the [SetSensorParameter](#) function call and the current state of the alpha filter parameters may be observed by calling the [GetSensorParameter](#) function call. See [FILTER ALPHA PARAMETERS](#) for a complete description of their meaning and use.

- **AC Narrow Notch Filter**

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

- **AC Wide Notch Filter**

- This filter is an 8 tap finite impulse response (FIR) filter that is applied to the sensor data to eliminate signals with a frequency between 30 and 72 Hz. Note: for this filter to work properly the system parameter: [POWER LINE FREQUENCY](#) must be correctly initialized using the [SetSystemParameter](#) function call.

**Note:**

Sudden Output filter not implemented at time of manual writing.

- **Sudden Output Change Filter**

- If selected this filter will lock the output data to the current position and orientation if a sudden large change in position or orientation is detected. 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 6 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.

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

- **SERIAL NUMBER**

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

- **SENSOR OFFSETS**

Default position outputs from the trakSTAR represent the X, Y, Z position of the magnetic center of the sensor coil (approximate center of sensor housing) with respect to the transmitter origin. The Sensor Offsets allow you to configure the position outputs such that the tracker is reporting the position of a location that is offset from the center of the sensor. See the Sensor Parameter type [SENSOR_OFFSET](#) (or [OFFSET](#) RS232 command) 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 tracking 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. The buffer available to the system for each sensor is 8 records long. If the host application does not keep up with the constant stream of data being provided in this mode, this buffer will overflow and records will be lost.

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

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

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

Some specific information of interest to the user concerning the Main PCB hardware is available.

- **SERIAL NUMBER**

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

- **SOFTWARE REVISION NUMBER**

The Tracker's main firmware version number is stored as a 2 digit revision number. It can be accessed by issuing the [GetBoardConfiguration](#) command for the 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.

- **NEXT 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. Upon power up the Reference Frame is initialized to 0,0,0 and the XYZ Reference Frame is disabled. Note: No transmitter is selected at power up.

- **REFERENCE_FRAME**

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

- **XYZ_REFERENCE_FRAME**

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

- **SERIAL_NUMBER**

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

- **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](#).

Pre-Initialization Setup


Note:

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

There are some specific operations that are available PRIOR to initializing the 3DGuidance tracking system. These operations utilize parameters passed with the [GetSystemParameter](#) and [SetSystem Parameter](#) 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.

- **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_PARAMETER](#) for additional details.

NOTE: This parameter can be used prior to a InitializeBIRDSysyem() 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 InitBIRDSysyem() 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 InitializeBIRDSysyem API call. Disabling reset will make the InitBIRDSysyem 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 InitializeBIRDSysyem() 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. For Non-Dipole configurations, 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.

System Initialization

With the exception of the ‘Pre-Init’ commands, the first operation that must be performed before the system can be used is initialization. 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 which either indicate a condition that has prevented the system from initializing or they indicate a prevailing condition that disallows the system from completing the initialization. For example, the error code:

`BIRD_ERROR_COMMAND_TIME_OUT` probably indicates a non-responding board. This is a hard failure. The error code `BIRD_ERROR_INVALID_DEVICE_ID` indicates that although the board is functional, initialization will not be allowed to proceed because the board is incompatible with the driver and API. The error codes are provided as a diagnostic and indicate a system condition that needs to be rectified before initialization can complete. Without a complete and successful initialization the system cannot be used.

Note: Initialization is an all-inclusive operation. Internally, the first task it performs is to enumerate the 3DGuidance™ trackers 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
}
```

Note: An error handler should be called which will display or log the error reported. The application should terminate as no further progress is possible without successful initialization. Note that the error returned by a call when the system has not been

initialized yet (BIRD_ERROR_SYSTEM_UNINITIALIZED) can be decoded at any time using the GetErrorText() call.

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 power line

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
}

```

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 a configuration file (.ini) 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.

 **Tip:** 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 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:\trakSTAR\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 most other API calls the `InitializeBIRDSystem()` call must be made before `RestoreSystemConfiguration()` can be used.

Sensor Setup

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

 **Note:** The ALL_SENSORS Sensor ID can only be used in the GetXXRecord() calls.

In most cases the default filter and quality parameters will be found to provide adequate performance for the intended application. 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 need be changed and the system will still operate successfully.

Note: It is necessary to set or change the parameter for each of the sensors individually as required. This allows each sensor to have its parameters set to different values. The following code fragment gives an example of how to call the set parameter function in this case to set the data format to a double floating point value of position and matrix:

```
USHORT sensorID = 2;
int errorCode;
DATA_FORMAT_TYPE format = DOUBLE_POSITION_MATRIX;

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

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

```
#include "ATC3DG.h"
```

////////////////////
 //////////////////

```

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

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

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

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

```

```

SET_SENSOR_PARAMETER(sensorID, DATA_FORMAT,
DOUBLE_POSITION_ANGLES_TIME_STAMP);

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

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

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

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

```

Transmitter Setup

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

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

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

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

Acquiring Tracking Data

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

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

```

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

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

////////////////////////////////////
////////
//
// Get a record from sensor #0.

```



```
// The default record type is DOUBLE_POSITION_ANGLES
//
USHORT sensorID = 0;
DOUBLE_POSITION_ANGLES_RECORD record;
errorCode = GetAsynchronousRecord(sensorID, &record,
sizeof(record));
if(errorCode!=BIRD_ERROR_SUCCESS)
{
    errorHandler(errorCode);
}
```

Error Handling

Each call to the API will return either an error code or a status code depending on the command issued. Commands that return a status code are the `GetSystemStatus()`, `GetBoardStatus()`, `GetSensorStatus()` and `GetTransmitterStatus()` commands.

The only acceptable response to a command is `BIRD_ERROR_SUCCESS`. If any other response is received in response to sending a command, 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. Note that `GetErrorText()` does not require the 3D Guidance system to be initialized before it can be used.

It is possible for the software to recover from some error code conditions. For example if the system has not been initialized then the error code `BIRD_ERROR_SYSTEM_UNINITIALIZED` will be returned. The software could recover by calling `InitializeBIRDSystem()` before doing anything else.

But this error is usually an indication of a software “bug”. Other errors like `BIRD_ERROR_NO_SENSOR_ATTACHED` can be recovered from by displaying a message to the user suggesting that they attach a sensor to the system.



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

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).

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.

3DGuidance API

The following elements define the API used with the 3DGuidance systems.

[3D Guidance API Functions](#)

[3D Guidance API Structures](#)

[3DGuidance API Enumeration Types](#)

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

[3D Guidance Initialization Files](#)

3D Guidance API Functions

The following functions are used with the 3DGuidance systems.

[InitializeBIRDSystem](#)

[GetBIRDSystemConfiguration](#)

[GetTransmitterConfiguration](#)

[GetSensorConfiguration](#)

[GetBoardConfiguration](#)

[GetSystemParameter](#)

[GetSensorParameter](#)

[GetTransmitterParameter](#)

[GetBoardParameter](#)

[SetSystemParameter](#)

[SetSensorParameter](#)

[SetTransmitterParameter](#)

[SetBoardParameter](#)

[GetAsynchronousRecord](#)

[GetSynchronousRecord](#)

[GetBIRDError](#)

[GetErrorText](#)

[GetSensorStatus](#)

[GetTransmitterStatus](#)

[GetBoardStatus](#)

[GetSystemStatus](#)

[SaveSystemConfiguration](#)

[RestoreSystemConfiguration](#)

[CloseBIRDSystem](#)

InitializeBIRDSystem

The **InitializeBIRDSystem** function resets (when configured to do so) and initializes the 3DGuidance 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 3DGuidance 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 3DGuidance device has been found that is not supported by this API dll. Verify 3DGuidance model installed. |
| BIRD_ERROR_FAILED_LOCKING_DEVICE | Driver could not lock 3DGuidance resources. Check that there is not another application using the hardware. |
| 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_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. |
| 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_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_UNRECOGNIZED_MODEL_STRING | The firmware is reporting a model string that is unrecognized by the API dll. This could be due to a |

| | |
|-----------------------------------|---|
| | hardware failure causing the model string data to be corrupted or 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_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. With the exception of a few [pre-initialization](#) operations, this function must be called first, before other commands can be sent.

Requirements

Windows NT/2000: Requires Windows 2000 or later.

Header: Declared in ATC3DG.h

Library: Use ATC3DG.lib

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 InitializeBIRDSystem function must be called first. |

Remarks

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

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

Requirements

Windows NT/2000: Requires Windows 2000 or later.

Header: Declared in ATC3DG.h

Library: Use ATC3DG.lib

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 InitializeBIRDSystem 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

Windows NT/2000: Requires Windows 2000 or later.

Header: Declared in ATC3DG.h

Library: Use ATC3DG.lib

See Also

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

Windows NT/2000: Requires Windows 2000 or later.

Header: Declared in ATC3DG.h

Library: Use ATC3DG.lib

See Also

[GetSensorParameter](#), [SetSensorParameter](#), [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 InitializeBIRDSystem function must be called first. |
| BIRD_ERROR_INVALID_DEVICE_ID | The boardID passed was out of range for the system. |

Remarks

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

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

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

Requirements

Windows NT/2000: Requires Windows 2000 or later.

Header: Declared in ATC3DG.h

Library: Use ATC3DG.lib

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. 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 InitializeBIRDSystem 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 SYSTEM_PARAMETER_TYPE used. |

Remarks

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

Requirements

Windows NT/2000: Requires Windows 2000 or later.

Header: Declared in ATC3DG.h

Library: Use ATC3DG.lib

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 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 InitializeBIRDSystem function must be called first. |
| BIRD_ERROR_INCORRECT_PARAMETER_SIZE | The value of the <i>bufferSize</i> parameter passed did not match the size of the parameter being returned. |
| BIRD_ERROR_ILLEGAL_COMMAND_PARAMETER | Invalid enumerated constant of type SENSOR_PARAMETER_TYPE used. |
| BIRD_ERROR_INVALID_DEVICE_ID | The sensorID passed was out of range for the system. |

Remarks

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

Requirements

Windows NT/2000: Requires Windows 2000 or later.

Header: Declared in ATC3DG.h

Library: Use ATC3DG.lib

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 InitializeBIRDSystem function must be called first. |
| BIRD_ERROR_INCORRECT_PARAMETER_SIZE | The value of the <i>bufferSize</i> parameter passed did not match the size of the parameter being returned. |
| BIRD_ERROR_ILLEGAL_COMMAND_PARAMETER | Invalid enumerated constant of type TRANSMITTER_PARAMETER_TYPE used. |
| BIRD_ERROR_INVALID_DEVICE_ID | The transmitterID passed was out of range for the system. |

Remarks

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

Requirements

Windows NT/2000: Requires Windows 2000 or later.

Header: Declared in ATC3DG.h

Library: Use ATC3DG.lib

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 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 InitializeBIRDSystem function must be called first. |
| BIRD_ERROR_INCORRECT_PARAMETER_SIZE | The value of the <i>bufferSize</i> parameter passed did not match the size of the parameter being returned. |
| BIRD_ERROR_ILLEGAL_COMMAND_PARAMETER | Invalid enumerated constant of type BOARD_PARAMETER_TYPE used. |
| BIRD_ERROR_INVALID_DEVICE_ID | The boardID passed was out of range for the system. |

Remarks

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

Requirements

Windows NT/2000: Requires Windows 2000 or later.

Header: Declared in ATC3DG.h

Library: Use ATC3DG.lib

See Also

[SetBoardParameter](#), [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 InitializeBIRDSystem 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 SYSTEM_PARAMETER_TYPE used. |
| 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_NO_TRANSMITTER_ATTACHED | A request was made to do one of the following: 1) Turn off the currently running transmitter and there is no transmitter attached to the system 2) Turn on the transmitter with the selected ID and there is no transmitter attached at that ID. |
| BIRD_ERROR_COMMAND_TIME_OUT | 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

Windows NT/2000: Requires Windows 2000 or later.

Header: Declared in ATC3DG.h

Library: Use ATC3DG.lib

See Also

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

SetSensorParameter

The **SetSensorParameter** function allows an application to select and change specific characteristics of individual sensors in a trakSTAR 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 InitializeBIRDSystem 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. |
| 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

Windows NT/2000: Requires Windows 2000 or later.

Header: Declared in ATC3DG.h

Library: Use ATC3DG.lib

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 than 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 InitializeBIRDSystem 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

Windows NT/2000: Requires Windows 2000 or later.

Header: Declared in ATC3DG.h

Library: Use ATC3DG.lib

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 InitializeBIRDSystem function must be called first. |
| BIRD_ERROR_INCORRECT_PARAMETER_SIZE | The value of the <i>bufferSize</i> parameter passed did not match the size of the parameter being returned. |
| BIRD_ERROR_ILLEGAL_COMMAND_PARAMETER | Invalid enumerated constant of type BOARD_PARAMETER_TYPE used. |
| BIRD_ERROR_INVALID_DEVICE_ID | The boardID passed was out of range for the system. |

Remarks

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

Requirements

Windows NT/2000: Requires Windows 2000 or later.

Header: Declared in ATC3DG.h

Library: Use ATC3DG.lib

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 currently selected DATA_FORMAT_TYPE exactly.

Return Values

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

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

| | |
|------------------------------|---|
| BIRD_ERROR_SENSOR_SATURATED | The attached sensor 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 GetSensorStatus function to determine the precise problem. |
| BIRD_ERROR_INVALID_DEVICE_ID | The transmitterID passed was out of range for the system. |
| BIRD_ERROR_COMMAND_TIME_OUT | 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 Code](#) 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

Windows NT/2000: Requires Windows 2000 or later.

Header: Declared in ATC3DG.h

Library: Use ATC3DG.lib

See Also

[GetSynchronousRecord](#)

GetSynchronousRecord

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

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

Parameters

sensorID

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

pRecord

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

recordSize

[in] Contains the size of the buffer whose address is passed in *pRecord*. Note the *recordSize* value must match the size of the currently selected DATA_FORMAT_TYPE exactly.

Return Values

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

| Value | Meaning |
|-----------------------------------|---|
| BIRD_ERROR_SUCCESS | No errors occurred. Call completed successfully |
| BIRD_ERROR_SYSTEM_UNINITIALIZED | The 3DGuidance hardware and system has not been initialized yet. The InitializeBIRDSystem function must be called first. |
| BIRD_ERROR_NO_SENSOR_ATTACHED | Request for data record from a sensor channel where no sensor is attached or the sensor has been removed. |
| BIRD_ERROR_NO_TRANSMITTER_RUNNING | Request for data record but there is no transmitter running. Either the application failed to turn a transmitter on or the currently running transmitter has a problem or has been removed. If a transmitter problem is suspected use the GetTransmitterStatus function to determine the precise problem. |

| | |
|----------------------------------|--|
| BIRD_ERROR_INCORRECT_RECORD_SIZE | The <i>recordSize</i> of the buffer passed to the function does not match the size of the data format currently selected. |
| BIRD_ERROR_SENSOR_SATURATED | The attached sensor 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 GetSensorStatus function to determine the precise problem. |
| BIRD_ERROR_INVALID_DEVICE_ID | The transmitterID passed was out of range for the system. |
| BIRD_ERROR_COMMAND_TIME_OUT | 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 trakSTAR 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 [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 Code](#) 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

Windows NT/2000: Requires Windows 2000 or later.

Header: Declared in ATC3DG.h

Library: Use ATC3DG.lib

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 InitializeBIRDSystem 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

Windows NT/2000: Requires Windows 2000 or later.

Header: Declared in ATC3DG.h

Library: Use ATC3DG.lib

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

Return Values

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

| Value | Meaning |
|--------------------------------------|--|
| BIRD_ERROR_SUCCESS | No errors occurred. Call completed successfully |
| BIRD_ERROR_SYSTEM_UNINITIALIZED | The 3DGuidance hardware and system has not been initialized yet. The InitializeBIRDSystem 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

Windows NT/2000: Requires Windows 2000 or later.

Header: Declared in ATC3DG.h

Library: Use ATC3DG.lib

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 InitializeBIRDSystem | x | x | x | x |
| 7 | NO_TRANSMITTER_RUNNING | An attempt was made to call GetAsynchronousRecord 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 |
|---------|------------|----------------------|---|---|---|---|

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

Windows NT/2000: Requires Windows 2000 or later.

Header: Declared in ATC3DG.h

Library: Use ATC3DG.lib

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 InitializeBIRDSsystem | x | x | x | x |
| 7 | NO_TRANSMITTER_RUNNING | An attempt was made to call GetAsynchronousRecord 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 |
|---------|------------|----------------------|---|---|---|---|

Remarks

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

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

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

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

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

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

Requirements

Windows NT/2000: Requires Windows 2000 or later.

Header: Declared in ATC3DG.h

Library: Use ATC3DG.lib

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 InitializeBIRDSsystem | x | x | x | x |
| 7 | NO_TRANSMITTER_RUNNING | An attempt was made to call GetAsynchronousRecord 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_INITIA | The sensor has not acquired enough raw magnetic data | | | x | |

| | | | | | | |
|---------|------------|--------------------------------------|---|---|---|---|
| | LIZING | to compute an accurate P&O solution. | | | | |
| 14 - 31 | <reserved> | Always returns zero. | x | x | x | x |

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

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

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

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

Requirements

Windows NT/2000: Requires Windows 2000 or later.

Header: Declared in ATC3DG.h

Library: Use ATC3DG.lib

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 InitializeBIRDSystem | x | x | x | x |
| 7 | NO_TRANSMITTER_RUNNING | An attempt was made to call GetAsynchronousRecord 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 |

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 InitializeBIRDSystem has been called. If the function InitializeBIRDSystem 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

Windows NT/2000: Requires Windows 2000 or later.

Header: Declared in ATC3DG.h

Library: Use ATC3DG.lib

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 InitializeBIRDSysyem function must be called first. |
| BIRD_ERROR_UNABLE_TO_CREATE_FILE | The call was unable to complete for some unspecified reason. Check the format of the file name string. |
| BIRD_ERROR_CONFIG_INTERNAL | Internal error in configuration file handler. Report to vendor. |

Remarks

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

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

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

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

will save the configuration file to <C:\MyPrograms\MyConfiguration.ini>. This default mode of operation differs from the `RestoreSystemConfiguration()` call 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

Windows NT/2000: Requires Windows 2000 or later.

Header: Declared in ATC3DG.h

Library: Use ATC3DG.lib

See Also

[RestoreSystemConfiguration](#)

RestoreSystemConfiguration

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

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

Parameters

lpFileName

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

Return Values

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

| Value | Meaning |
|---------------------------------------|---|
| BIRD_ERROR_SUCCESS | No errors occurred. Call completed successfully |
| BIRD_ERROR_SYSTEM_UNINITIALIZED | The 3DGuidance hardware and system has not been initialized yet. The InitializeBIRDSystem 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 <code>SaveSystemConfiguration()</code> 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 (<code>NumberOfBoards=3</code>) but the system initialization routine – <code>InitializeBIRDSystem()</code> only detected two. |
| BIRD_ERROR_CONFIG_INTERNAL | Internal error in configuration file handler. Report to vendor. |

Remarks

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

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

NOTE: if the full pathname specification is not provided then the default search path is in the 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 [3DGuidance Initialization File Format](#)

Requirements

Windows NT/2000: Requires Windows 2000 or later.

Header: Declared in ATC3DG.h

Library: Use ATC3DG.lib

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

Windows NT/2000: Requires Windows 2000 or later.

Header: Declared in ATC3DG.h

Library: Use ATC3DG.lib

See Also

[InitializeBIRDSystem](#)

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 PARAMETER](#)
[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](#)

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

Windows NT/2000: Requires Windows 2000 or later.

Header: Declared in ATC3DG.h
Library: Use ATC3DG.lib

See Also

TRANSMITTER_CONFIGURATION

The **TRANSMITTER_CONFIGURATION** structure contains an individual transmitter's information.

```
typedef struct tagTRANSMITTER_CONFIGURATION{
    ULONG          serialNumber;
    USHORT         boardNumber;
    USHORT         channelNumber;
    DEVICE_TYPES    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

Windows NT/2000: Requires Windows 2000 or later.

Header: Declared in ATC3DG.h

Library: Use ATC3DG.lib

See Also

SENSOR_CONFIGURATION

The **SENSOR_CONFIGURATION** structure contains an individual sensor's information.

```
typedef struct tagSENSOR_CONFIGURATION{
    ULONG          serialNumber;
    USHORT         boardNumber;
    USHORT         channelNumber;
    DEVICE_TYPES    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

Windows NT/2000: Requires Windows 2000 or later.

Header: Declared in ATC3DG.h

Library: Use ATC3DG.lib

See Also

BOARD_CONFIGURATION

The **BOARD_CONFIGURATION** structure contains an individual board's information.

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

Members

serialNumber

The serial number of the board.

type

The board type. The type is of the enumeration type [BOARD_TYPES](#). 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 "6DPC18MM ". The string is padded with space characters to the end of the buffer.

Requirements

Windows NT/2000: Requires Windows 2000 or later.

Header: Declared in ATC3DG.h

Library: Use ATC3DG.lib

See Also

ADAPTIVE_PARAMETERS

The **ADAPTIVE_PARAMETERS** structure contains the adaptive DC filter parameters for an individual sensor channel.

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

Members

alphaMin[7]

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

alphaMax[7]

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

vm[7]

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

alphaOn

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

Requirements

Windows NT/2000: Requires Windows 2000 or later.

Header: Declared in ATC3DG.h

Library: Use ATC3DG.lib

See Also

QUALITY_PARAMETERS

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

```
typedef struct tagQUALITY_PARAMETERS{  
    SHORT          error_slope;  
    SHORT          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. Valid values range between –127 and 127.

error_offset

This value is the offset of the inherent system error. Valid values range between –127 and 127.

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. Valid values range between 0 and 127.

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. Valid values range between 0 and 127.

Requirements

Windows NT/2000: Requires Windows 2000 or later.

Header: Declared in ATC3DG.h

Library: Use ATC3DG.lib

See Also

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

Windows NT/2000: Requires Windows 2000 or later.

Header: Declared in ATC3DG.h

Library: Use ATC3DG.lib

See Also

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

Windows NT/2000: Requires Windows 2000 or later.

Header: Declared in ATC3DG.h

Library: Use ATC3DG.lib

See Also

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/SetSystemParameter` 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 Table 7.

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 Table 7.

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 Suite Name | 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 Suite/Test Name . | 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) | If successful returns the number of the final suite. If fails stops at and returns the number of the suite | If successful returns the number of the final test. If fails stops at and returns the number of the test | 64 character zero terminated string containing the Suite/Test Name . | If successful returns an error code of zero If fails returns an error code. * |
| SetSystemParameter() () | Execute entire set of tests for the selected suite. | N | 0 | Suite number N is returned. | If successful returns the number of the final test. If fails stops at and returns the number of the test | 64 character zero terminated string containing the Suite/Test name . | If successful returns an error code of zero If fails returns an error code. * |
| SetSystemParameter() () | Execute an individual diagnostic test. | N | M | Suite number N is returned. | Test number M is returned. | 64 character zero terminated string containing the Suite/Test name . | If successful returns an error code of zero If fails returns an error code. * |

| | | | | | | | |
|--|--|--|--|--|--|--|--|
| | | | | | | | |
|--|--|--|--|--|--|--|--|

Table 7 Diagnostic Test Usage

* NOTE: A character string describing the error condition can be obtained by calling GetLastError() and passing the error code.

Requirements

Windows NT/2000: Requires Windows 2000 or later.

Header: Declared in PCIBird3.h

Library: Use PCIBird3.lib

See Also

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

Windows NT/2000: Requires Windows 2000 or later.

Header: Declared in `ATC3DG.h`

Library: Use `ATC3DG.lib`

See Also

BOARD_REVISIONS

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

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

Members

boot_loader_sw_number

Major revision number for the boot loader.

boot_loader_sw_revision

Minor revision number for the boot loader.

mdsp_sw_number

Major revision number for the acquisition DSP.

mdsp_sw_revision

Minor revision number for the acquisition DSP.

nondipole_sw_number

Major revision number for the 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

Windows NT/2000: Requires Windows 2000 or later.

Header: Declared in ATC3DG.h

Library: Use ATC3DG.lib

See Also

SHORT_POSITION_RECORD

The **SHORT_POSITION_RECORD** structure contains position information only in 16-bit signed integer format.

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

Members

x

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

y

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

z

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

Remarks

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

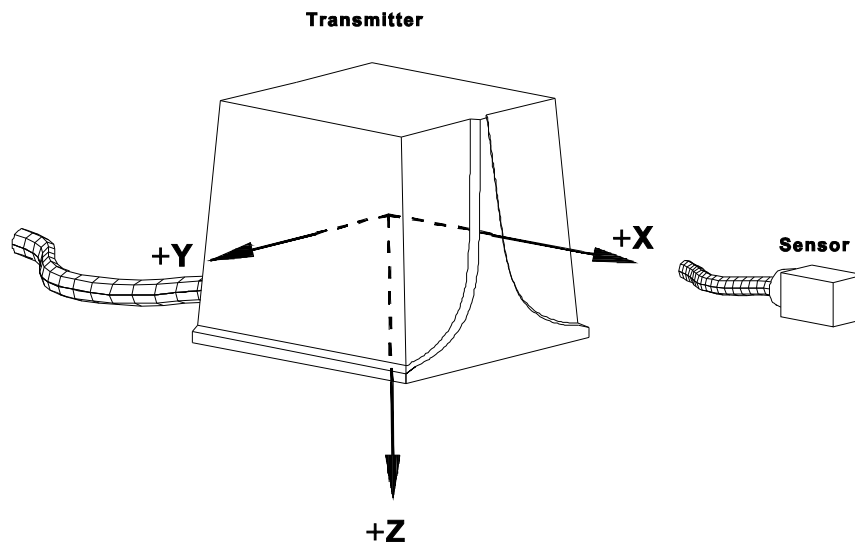


Figure 36 : Measurement Reference Frame (Mid-Range Transmitter)

Requirements

Windows NT/2000: Requires Windows 2000 or later.

Header: Declared in ATC3DG.h

Library: Use ATC3DG.lib

See Also

SHORT_ANGLES_RECORD

The **SHORT_ANGLES_RECORD** structure contains Euler angle information only in 16-bit signed integer format.

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

Members

a

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

e

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

r

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

Remarks

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

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

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

Angle information is 0 when sensor saturation occurs.

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

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

Requirements

Windows NT/2000: Requires Windows 2000 or later.

Header: Declared in ATC3DG.h

Library: Use ATC3DG.lib

See Also

3D GUIDANCE API REFERENCE

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:

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

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

$$\begin{vmatrix} \cos(Y) \cdot \cos(Z) & \cos(Y) \cdot \sin(Z) & -\sin(Y) \\ -(\cos(X) \cdot \sin(Z)) & (\cos(X) \cdot \cos(Z)) & \sin(X) \cdot \cos(Y) \\ (\sin(X) \cdot \sin(Z)) & -(\sin(X) \cdot \cos(Z)) & \cos(X) \cdot \cos(Y) \end{vmatrix}$$

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

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

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

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

Matrix information is 0 when sensor saturation occurs.

Requirements

Windows NT/2000: Requires Windows 2000 or later.

Header: Declared in ATC3DG.h

Library: Use ATC3DG.lib

See Also

SHORT_QUATERNIONS_RECORD

The **SHORT_QUATERNIONS_RECORD** structure contains only the 4 quaternion values in 16-bit signed integer format.

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

Members

q[4]

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

Remarks

In the QUATERNION mode, the Tracker outputs the four quaternion parameters that describe the orientation of the sensor with respect to the transmitter. The 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

Windows NT/2000: Requires Windows 2000 or later.

Header: Declared in ATC3DG.h

Library: Use ATC3DG.lib

See Also

SHORT_POSITION_ANGLES_RECORD

The **SHORT_POSITION_ANGLES_RECORD** structure contains position and angles information in 16-bit signed integer format.

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

Members

x

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

y

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

z

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

a

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

e

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

r

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

Requirements

Windows NT/2000: Requires Windows 2000 or later.

Header: Declared in ATC3DG.h

Library: Use ATC3DG.lib

See Also

[SHORT_POSITION_RECORD](#), [SHORT_POSITION_ANGLES_RECORD](#)

SHORT_POSITION_MATRIX_RECORD

The **SHORT_POSITION_MATRIX_RECORD** structure contains position and matrix information in 16-bit signed integer format.

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

Members

x

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

y

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

z

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

s[3][3]

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

Requirements

Windows NT/2000: Requires Windows 2000 or later.

Header: Declared in ATC3DG.h

Library: Use ATC3DG.lib

See Also

[SHORT_POSITION_RECORD](#), [SHORT_MATRIX_RECORD](#)

SHORT_POSITION_QUATERNION_RECORD

The **SHORT_POSITION_QUATERNION_RECORD** structure contains position and quaternion information in 16-bit signed integer format.

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

Members

x

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

y

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

z

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

q[4]

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

Requirements

Windows NT/2000: Requires Windows 2000 or later.

Header: Declared in ATC3DG.h

Library: Use ATC3DG.lib

See Also

[SHORT_POSITION_RECORD](#), [SHORT_QUATERNIONS_RECORD](#)

DOUBLE_POSITION_RECORD

The **DOUBLE_POSITION_RECORD** structure contains position information only in double floating point format.

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

Members

x

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

y

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

z

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

Remarks

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

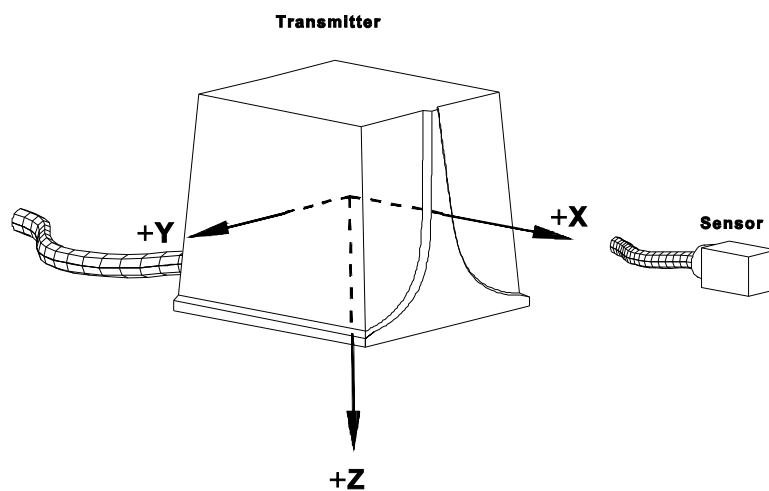


Figure 37 Measurement Reference Frame (Mid-Range Transmitter)

Requirements

Windows NT/2000: Requires Windows 2000 or later.

Header: Declared in ATC3DG.h

Library: Use ATC3DG.lib

See Also

DOUBLE_ANGLES_RECORD

The **DOUBLE_ANGLES_RECORD** structure contains Euler angles information only in double floating point format.

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

Members

a

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

e

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

r

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

Remarks

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

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

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

Angle information is 0 when sensor saturation occurs.

Requirements

Windows NT/2000: Requires Windows 2000 or later.

Header: Declared in ATC3DG.h

Library: Use ATC3DG.lib

See Also

DOUBLE_MATRIX_RECORD

The **DOUBLE_MATRIX_RECORD** structure contains only a 3x3 rotation matrix in double floating point format.

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

Members

s[3][3]

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

Remarks

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

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

| | | |
|---------------|---------------|---------------|
| M(1,1) | M(1,2) | M(1,3) |
| M(2,1) | M(2,2) | M(2,3) |
| M(3,1) | M(3,2) | M(3,3) |

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

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

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

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

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

Matrix information is 0 when sensor saturation occurs.

Requirements

Windows NT/2000: Requires Windows 2000 or later.

Header: Declared in ATC3DG.h

Library: Use ATC3DG.lib

See Also

DOUBLE_QUATERNIONS_RECORD

The **DOUBLE_QUATERNIONS_RECORD** structure contains only an array of 4 quaternion values in double floating point format.

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

Members

q[4]

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

Remarks

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

Requirements

Windows NT/2000: Requires Windows 2000 or later.

Header: Declared in ATC3DG.h

Library: Use ATC3DG.lib

See Also

DOUBLE_POSITION_ANGLES_RECORD

The **DOUBLE_POSITION_ANGLES_RECORD** structure contains position and Euler angle information in double floating point format.

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

Members

x

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

y

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

z

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

a

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

e

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

r

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

Requirements

Windows NT/2000: Requires Windows 2000 or later.

Header: Declared in ATC3DG.h

Library: Use ATC3DG.lib

See Also

[DOUBLE_POSITION_RECORD](#), [DOUBLE_ANGLES_RECORD](#)

DOUBLE_POSITION_MATRIX_RECORD

The **DOUBLE_POSITION_MATRIX_RECORD** structure contains position and matrix information in double floating point format.

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

Members

x

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

y

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

z

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

s[3][3]

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

Requirements

Windows NT/2000: Requires Windows 2000 or later.

Header: Declared in ATC3DG.h

Library: Use ATC3DG.lib

See Also

[DOUBLE_POSITION_RECORD](#), [DOUBLE_MATRIX_RECORD](#)

DOUBLE_POSITION_QUATERNION_RECORD

The **DOUBLE_POSITION_QUATERNION_RECORD** structure contains position and quaternion information in double floating point format.

```
typedef struct tagDOUBLE_POSITION_QUATERNION{
    double      x;
    double      y;
    double      z;
    double      q[4];
} DOUBLE_POSITION_QUATERNION_RECORD, *PDOUBLE_POSITION_QUATERNION_RECORD;
```

Members

x

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

y

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

z

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

q[4]

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

Requirements

Windows NT/2000: Requires Windows 2000 or later.

Header: Declared in ATC3DG.h

Library: Use ATC3DG.lib

See Also

[DOUBLE_POSITION_RECORD](#), [DOUBLE_QUATERNIONS_RECORD](#)

DOUBLE_POSITION_TIME_STAMP_RECORD

The **DOUBLE_POSITION_TIME_STAMP_RECORD** structure contains position information only in double floating point format.

```
typedef struct tagDOUBLE_POSITION_TIME_STAMP{
    double      x;
    double      y;
    double      z;
    double      time;
} DOUBLE_POSITION_TIME_STAMP_RECORD, *PDOUBLE_POSITION_TIME_STAMP_RECORD;
```

Members

x

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

y

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

z

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

time

The time variable is the time stamp for the data record and is returned as a **double** value. The integer portion of the variable represents the number of elapsed seconds since midnight, Jan 1, 1970, UTC, and is the standard way of representing time and date. If this is cast as a time_t structure, it can be converted into a date and time string using ctime() for example. The fractional part of the time variable represents fractions of a second.

Requirements

Windows NT/2000: Requires Windows 2000 or later.

Header: Declared in ATC3DG.h

Library: Use ATC3DG.lib

See Also

[DOUBLE_POSITION_RECORD](#)

DOUBLE_ANGLES_TIME_STAMP_RECORD

The **DOUBLE_ANGLES_TIME_STAMP_RECORD** structure contains Euler angles information only in double floating point format.

```
typedef struct tagDOUBLE_ANGLES_TIME_STAMP{
    double      a;
    double      e;
    double      r;
    double      time;
} DOUBLE_ANGLES_TIME_STAMP_RECORD, *PDOUBLE_ANGLES_TIME_STAMP_RECORD;
```

Members

a

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

e

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

r

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

time

The time variable is the time stamp for the data record and is returned as a **double** value. The integer portion of the variable represents the number of elapsed seconds since midnight, Jan 1, 1970, UTC, and is the standard way of representing time and date. If this is cast as a time_t structure, it can be converted into a date and time string using ctime() for example. The fractional part of the time variable represents fractions of a second.

Requirements

Windows NT/2000: Requires Windows 2000 or later.

Header: Declared in ATC3DG.h

Library: Use ATC3DG.lib

See Also

[DOUBLE_ANGLES_RECORD](#)

DOUBLE_MATRIX_TIME_STAMP_RECORD

The **DOUBLE_MATRIX_TIME_STAMP_RECORD** structure contains only a 3x3 rotation matrix in double floating point format.

```
typedef struct tagDOUBLE_MATRIX_TIME_STAMP{  
    double      s[3][3];  
    double      time;  
} DOUBLE_MATRIX_TIME_STAMP_RECORD, *PDOUBLE_MATRIX_TIME_STAMP_RECORD;
```

Members

s[3][3]

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

time

The time variable is the time stamp for the data record and is returned as a **double** value. The integer portion of the variable represents the number of elapsed seconds since midnight, Jan 1, 1970, UTC, and is the standard way of representing time and date. If this is cast as a time_t structure, it can be converted into a date and time string using ctime() for example. The fractional part of the time variable represents fractions of a second.

Requirements

Windows NT/2000: Requires Windows 2000 or later.

Header: Declared in ATC3DG.h

Library: Use ATC3DG.lib

See Also

[DOUBLE_MATRIX_RECORD](#)

DOUBLE_QUATERNIONS_TIME_STAMP_RECORD

The **DOUBLE_QUATERNIONS_TIME_STAMP_RECORD** structure contains only an array of 4 quaternion values in double floating point format.

```
typedef struct tagDOUBLE_QUATERNIONS_TIME_STAMP{
    double      q[4];
    double      time;
} DOUBLE_QUATERNIONS_TIME_STAMP_RECORD, *PDOUBLE_QUATERNIONS_TIME_STAMP_RECORD;
```

Members

q[4]

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

time

The time variable is the time stamp for the data record and is returned as a **double** value. The integer portion of the variable represents the number of elapsed seconds since midnight, Jan 1, 1970, UTC, and is the standard way of representing time and date. If this is cast as a time_t structure, it can be converted into a date and time string using ctime() for example. The fractional part of the time variable represents fractions of a second.

Requirements

Windows NT/2000: Requires Windows 2000 or later.

Header: Declared in ATC3DG.h

Library: Use ATC3DG.lib

See Also

[DOUBLE_QUATERNIONS_RECORD](#)

DOUBLE_POSITION_ANGLES_TIME_STAMP_RECORD

The **DOUBLE_POSITION_ANGLES_TIME_STAMP_RECORD** structure contains position and Euler angle information in double floating point format.

```
typedef struct tagDOUBLE_POSITION_ANGLES_TIME_STAMP{
    double      x;
    double      y;
    double      z;
    double      a;
    double      e;
    double      r;
    double      time;
} DOUBLE_POSITION_ANGLES_TIME_STAMP_RECORD, *PDOUBLE_POSITION_ANGLES_TIME_STAMP_RECORD;
```

Members

x

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

y

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

z

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

a

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

e

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

r

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

time

The time variable is the time stamp for the data record and is returned as a **double** value. The integer portion of the variable represents the number of elapsed seconds since midnight, Jan 1, 1970, UTC, and is the standard way of representing time and date. If this is cast as a time_t structure, it can be converted into a date and time string using ctime() for example. The fractional part of the time variable represents fractions of a second.

Requirements

Windows NT/2000: Requires Windows 2000 or later.

Header: Declared in ATC3DG.h

Library: Use ATC3DG.lib

See Also

[DOUBLE_POSITION_RECORD](#), [DOUBLE_ANGLES_RECORD](#)

DOUBLE_POSITION_MATRIX_TIME_STAMP_RECORD

The **DOUBLE_POSITION_MATRIX_TIME_STAMP_RECORD** structure contains position and matrix information in double floating point format.

```
typedef struct tagDOUBLE_POSITION_MATRIX_TIME_STAMP{
    double      x;
    double      y;
    double      z;
    double      s[3][3];
    double      time;
} DOUBLE_POSITION_MATRIX_TIME_STAMP_RECORD, *PDOUBLE_POSITION_MATRIX_TIME_STAMP_RECORD;
```

Members

x

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

y

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

z

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

s[3][3]

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

time

The time variable is the time stamp for the data record and is returned as a **double** value. The integer portion of the variable represents the number of elapsed seconds since midnight, Jan 1, 1970, UTC, and is the standard way of representing time and date. If this is cast as a time_t structure, it can be converted into a date and time string using time() for example. The fractional part of the time variable represents fractions of a second.

Requirements

Windows NT/2000: Requires Windows 2000 or later.

Header: Declared in ATC3DG.h

Library: Use ATC3DG.lib

See Also

[DOUBLE_POSITION_RECORD](#), [DOUBLE_MATRIX_RECORD](#)

DOUBLE_POSITION_QUATERNION_TIME_STAMP_RECORD

The **DOUBLE_POSITION_QUATERNION_TIME_STAMP_RECORD** structure contains position and quaternion information in double floating point format.

```
typedef struct tagDOUBLE_POSITION_QUATERNION_TIME_STAMP{
    double      x;
    double      y;
    double      z;
    double      q[4];
    double      time;
} DOUBLE_POSITION_QUATERNION_TIME_STAMP_RECORD,
*PDOUBLE_POSITION_QUATERNION_TIME_STAMP_RECORD;
```

Members

x

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

y

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

z

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

q[4]

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

time

The time variable is the time stamp for the data record and is returned as a **double** value. The integer portion of the variable represents the number of elapsed seconds since midnight, Jan 1, 1970, UTC, and is the standard way of representing time and date. If this is cast as a time_t structure, it can be converted into a date and time string using ctime() for example. The fractional part of the time variable represents fractions of a second.

Requirements

Windows NT/2000: Requires Windows 2000 or later.

Header: Declared in ATC3DG.h

Library: Use ATC3DG.lib

See Also

[DOUBLE_POSITION_RECORD](#), [DOUBLE_QUATERNIONS_RECORD](#)

DOUBLE_POSITION_TIME_Q_RECORD

The **DOUBLE_POSITION_TIME_Q_RECORD** structure contains position, timestamp and distortion information in double floating point format.

```
typedef struct tagDOUBLE_POSITION_TIME_Q{
    double      x;
    double      y;
    double      z;
    double      time;
    USHORT      quality;
} DOUBLE_POSITION_TIME_Q_RECORD, *PDOUBLE_POSITION_TIME_Q_RECORD;
```

Members

x

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

y

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

z

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

time

The time variable is the time stamp for the data record and is returned as a **double** value. The integer portion of the variable represents the number of elapsed seconds since midnight, Jan 1, 1970, UTC, and is the standard way of representing time and date. If this is cast as a **time_t** structure, it can be converted into a date and time string using **ctime()** for example. The fractional part of the time variable represents fractions of a second.

quality

The quality value is a 16-bit unsigned integer. A very small quality number indicates no or minimal position and orientation errors due to distortion of the transmitter field depending on how sensitive you have set the error indicator. A large quality number indicates maximum error for the sensitivity level selected.

Requirements

Windows NT/2000: Requires Windows 2000 or later.

Header: Declared in ATC3DG.h

Library: Use ATC3DG.lib

See Also

[DOUBLE_POSITION_RECORD](#)

DOUBLE_ANGLES_TIME_Q_RECORD

The **DOUBLE_ANGLES_TIME_Q_RECORD** structure contains Euler angles, timestamp and distortion information in double floating point format.

```
typedef struct tagDOUBLE_ANGLES_TIME_Q{
    double      a;
    double      e;
    double      r;
    double      time;
    USHORT      quality;
} DOUBLE_ANGLES_TIME_Q_RECORD, *PDOUBLE_ANGLES_TIME_Q_RECORD;
```

Members

a

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

e

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

r

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

time

The time variable is the time stamp for the data record and is returned as a **double** value. The integer portion of the variable represents the number of elapsed seconds since midnight, Jan 1, 1970, UTC, and is the standard way of representing time and date. If this is cast as a `time_t` structure, it can be converted into a date and time string using `ctime()` for example. The fractional part of the time variable represents fractions of a second.

quality

The quality value is a 16-bit unsigned integer. A very small quality number indicates no or minimal position and orientation errors due to distortion of the transmitter field depending on how sensitive you have set the error indicator. A large quality number indicates maximum error for the sensitivity level selected.

Requirements

Windows NT/2000: Requires Windows 2000 or later.

Header: Declared in ATC3DG.h

Library: Use ATC3DG.lib

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

Windows NT/2000: Requires Windows 2000 or later.

Header: Declared in ATC3DG.h

Library: Use ATC3DG.lib

See Also

[DOUBLE_MATRIX_RECORD](#)

DOUBLE_QUATERNIONS_TIME_Q_RECORD

The **DOUBLE_QUATERNIONS_TIME_Q_RECORD** structure contains an array of 4 quaternion values, timestamp and distortion information in double floating point format.

```
typedef struct tagDOUBLE_QUATERNIONS_TIME_Q{
    double      q[4];
    double      time;
    USHORT      quality;
} DOUBLE_QUATERNIONS_TIME_Q_RECORD, *PDOUBLE_QUATERNIONS_TIME_Q_RECORD;
```

Members

q[4]

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

time

The time variable is the time stamp for the data record and is returned as a **double** value. The integer portion of the variable represents the number of elapsed seconds since midnight, Jan 1, 1970, UTC, and is the standard way of representing time and date. If this is cast as a time_t structure, it can be converted into a date and time string using ctime() for example. The fractional part of the time variable represents fractions of a second.

quality

The quality value is a 16-bit unsigned integer. A very small quality number indicates no or minimal position and orientation errors due to distortion of the transmitter field depending on how sensitive you have set the error indicator. A large quality number indicates maximum error for the sensitivity level selected.

Requirements

Windows NT/2000: Requires Windows 2000 or later.

Header: Declared in ATC3DG.h

Library: Use ATC3DG.lib

See Also

[DOUBLE_QUATERNIONS_RECORD](#)

DOUBLE_POSITION_ANGLES_TIME_Q_RECORD

The **DOUBLE_POSITION_ANGLES_TIME_Q_RECORD** structure contains position Euler angle, timestamp and distortion information in double floating point format.

```
typedef struct tagDOUBLE_POSITION_ANGLES_TIME_Q{
    double          x;
    double          y;
    double          z;
    double          a;
    double          e;
    double          r;
    double          time;
    USHORT          quality;
} DOUBLE_POSITION_ANGLES_TIME_Q_RECORD, *PDOUBLE_POSITION_ANGLES_TIME_Q_RECORD;
```

Members

x

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

y

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

z

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

a

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

e

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

r

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

time

The time variable is the time stamp for the data record and is returned as a **double** value. The integer portion of the variable represents the number of elapsed seconds since midnight, Jan 1, 1970, UTC, and is the standard way of representing time and date. If this is cast as a time_t structure, it can be converted into a date and time string using ctime() for example. The fractional part of the time variable represents fractions of a second.

quality

The quality value is a 16-bit unsigned integer. A very small quality number indicates no or minimal position and orientation errors due to distortion of the transmitter field depending on how sensitive you have set the error indicator. A large quality number indicates maximum error for the sensitivity level selected.

Requirements

Windows NT/2000: Requires Windows 2000 or later.

Header: Declared in ATC3DG.h

Library: Use ATC3DG.lib

See Also

[DOUBLE_POSITION_RECORD](#), [DOUBLE_ANGLES_RECORD](#)

DOUBLE_POSITION_MATRIX_TIME_Q_RECORD

The **DOUBLE_POSITION_MATRIX_TIME_Q_RECORD** structure contains position, matrix, timestamp and distortion information in double floating point format.

```
typedef struct tagDOUBLE_POSITION_MATRIX_TIME_Q{
    double      x;
    double      y;
    double      z;
    double      s[3][3];
    double      time;
    USHORT      quality;
} DOUBLE_POSITION_MATRIX_TIME_Q_RECORD, *PDOUBLE_POSITION_MATRIX_TIME_Q_RECORD;
```

Members

x

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

y

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

z

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

s[3][3]

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

time

The time variable is the time stamp for the data record and is returned as a **double** value. The integer portion of the variable represents the number of elapsed seconds since midnight, Jan 1, 1970, UTC, and is the standard way of representing time and date. If this is cast as a time_t structure, it can be converted into a date and time string using ctime() for example. The fractional part of the time variable represents fractions of a second.

quality

The quality value is a 16-bit unsigned integer. A very small quality number indicates no or minimal position and orientation errors due to distortion of the transmitter field depending on how sensitive you have set the error indicator. A large quality number indicates maximum error for the sensitivity level selected.

Requirements

Windows NT/2000: Requires Windows 2000 or later.

Header: Declared in ATC3DG.h

Library: Use ATC3DG.lib

See Also

[DOUBLE_POSITION_RECORD](#), [DOUBLE_MATRIX_RECORD](#)

DOUBLE_POSITION_QUATERNION_TIME_Q_RECORD

The **DOUBLE_POSITION_QUATERNION_TIME_Q_RECORD** structure contains position, quaternion, timestamp and distortion information in double floating point format.

```
typedef struct tagDOUBLE_POSITION_QUATERNION_TIME_Q{
    double      x;
    double      y;
    double      z;
    double      q[4];
    double      time;
    USHORT      quality;
} DOUBLE_POSITION_QUATERNION_TIME_Q_RECORD, *PDOUBLE_POSITION_QUATERNION_TIME_Q_RECORD;
```

Members

x

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

y

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

z

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

q[4]

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

time

The time variable is the time stamp for the data record and is returned as a **double** value. The integer portion of the variable represents the number of elapsed seconds since midnight, Jan 1, 1970, UTC, and is the standard way of representing time and date. If this is cast as a time_t structure, it can be converted into a date and time string using ctime() for example. The fractional part of the time variable represents fractions of a second.

quality

The quality value is a 16-bit unsigned integer. A very small quality number indicates no or minimal position and orientation errors due to distortion of the transmitter field depending on how sensitive you have set the error indicator. A large quality number indicates maximum error for the sensitivity level selected.

Requirements

Windows NT/2000: Requires Windows 2000 or later.

Header: Declared in ATC3DG.h

Library: Use ATC3DG.lib

See Also

[DOUBLE_POSITION_RECORD](#), [DOUBLE_QUATERNIONS_RECORD](#)

SHORT_ALL_RECORD

The **SHORT_ALL_RECORD** structure contains position, Euler angles, rotation matrix and quaternion information in 16-bit signed integer format.

```
typedef struct tagSHORT_ALL{
    short      x;
    short      y;
    short      z;
    short      a;
    short      e;
    short      r;
    short      s[3][3];
    short      q[4];
} SHORT_ALL_RECORD, *PSHORT_ALL_RECORD;
```

Members

x

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

y

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

z

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

a

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

e

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

r

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

s[3][3]

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

q[4]

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

Requirements

Windows NT/2000: Requires Windows 2000 or later.

Header: Declared in ATC3DG.h

Library: Use ATC3DG.lib

See Also

[SHORT_ANGLES_RECORD](#), [SHORT_MATRIX_RECORD](#), [SHORT_POSITION_RECORD](#), [SHORT_QUATERNIONS_RECORD](#)

DOUBLE_ALL_RECORD

The **DOUBLE_ALL_RECORD** structure contains position, Euler angles, rotation matrix and quaternion information in double floating point format.

```
typedef struct tagDOUBLE_ALL{
    double      x;
    double      y;
    double      z;
    double      a;
    double      e;
    double      r;
    double      s[3][3];
    double      q[4];
} DOUBLE_ALL_RECORD, *PDOUBLE_ALL_RECORD;
```

Members

x

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

y

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

z

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

a

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

e

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

r

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

s[3][3]

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

q[4]

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

Requirements

Windows NT/2000: Requires Windows 2000 or later.

Header: Declared in ATC3DG.h

Library: Use ATC3DG.lib

See Also

[DOUBLE_ANGLES_RECORD](#), [DOUBLE_MATRIX_RECORD](#), [DOUBLE_POSITION_RECORD](#),
[DOUBLE_QUATERNIONS_RECORD](#)

DOUBLE_ALL_TIME_STAMP_RECORD

The **DOUBLE_ALL_TIME_STAMP_RECORD** structure contains position, Euler angles, rotation matrix, quaternion and timestamp information in double floating point format.

```
typedef struct tagDOUBLE_ALL_TIME_STAMP{
    double      x;
    double      y;
    double      z;
    double      a;
    double      e;
    double      r;
    double      s[3][3];
    double      q[4];
    double      time;
} DOUBLE_ALL_TIME_STAMP_RECORD, *PDOUBLE_ALL_TIME_STAMP_RECORD;
```

Members

x

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

y

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

z

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

a

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

e

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

r

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

s[3][3]

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

q[4]

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

time

The time variable is the time stamp for the data record and is returned as a **double** value. The integer portion of the variable represents the number of elapsed seconds since midnight, Jan 1, 1970, UTC, and is the standard way of representing time and date. If this is cast as a time_t structure, it can be converted into a date and time string using ctime() for example. The fractional part of the time variable represents fractions of a second.

Requirements

Windows NT/2000: Requires Windows 2000 or later.

Header: Declared in ATC3DG.h

Library: Use ATC3DG.lib

See Also

[DOUBLE_ANGLES_RECORD](#), [DOUBLE_MATRIX_RECORD](#), [DOUBLE_POSITION_RECORD](#),
[DOUBLE_QUATERNIONS_RECORD](#)

DOUBLE_ALL_TIME_STAMP_Q_RECORD

The **DOUBLE_ALL_TIME_STAMP_Q_RECORD** structure contains position, Euler angles, rotation matrix, quaternion, timestamp and quality information in double floating point format.

```
typedef struct tagDOUBLE_ALL_TIME_STAMP_Q{
    double      x;
    double      y;
    double      z;
    double      a;
    double      e;
    double      r;
    double      s[3][3];
    double      q[4];
    double      time;
    USHORT      quality;
} DOUBLE_ALL_TIME_STAMP_Q_RECORD, *PDOUBLE_ALL_TIME_STAMP_Q_RECORD;
```

Members

x

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

y

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

z

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

a

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

e

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

r

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

s[3][3]

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

q[4]

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

time

The time variable is the time stamp for the data record and is returned as a **double** value. The integer portion of the variable represents the number of elapsed seconds since midnight, Jan 1, 1970, UTC, and is the standard way of

representing time and date. If this is cast as a `time_t` structure, it can be converted into a date and time string using `ctime()` for example. The fractional part of the time variable represents fractions of a second.

quality

The quality value is a 16-bit unsigned integer. A very small quality number indicates no or minimal position and orientation errors due to distortion of the transmitter field depending on how sensitive you have set the error indicator. A large quality number indicates maximum error for the sensitivity level selected.

Requirements

Windows NT/2000: Requires Windows 2000 or later.

Header: Declared in `ATC3DG.h`

Library: Use `ATC3DG.lib`

See Also

[DOUBLE_ANGLES_RECORD](#), [DOUBLE_MATRIX_RECORD](#), [DOUBLE_POSITION_RECORD](#),
[DOUBLE_QUATERNIONS_RECORD](#)

DOUBLE_ALL_TIME_STAMP_Q_RAW_RECORD

The **DOUBLE_ALL_TIME_STAMP_Q_RAW_RECORD** structure contains position, Euler angles, rotation matrix, quaternion, timestamp, quality and raw matrix information in double floating point format.

```
typedef struct tagDOUBLE_ALL_TIME_STAMP_Q_RAW{
    double          x;
    double          y;
    double          z;
    double          a;
    double          e;
    double          r;
    double          s[3][3];
    double          q[4];
    double          time;
    USHORT          quality;
    Double          raw[3][3];
} DOUBLE_ALL_TIME_STAMP_Q_RAW_RECORD, *PDOUBLE_ALL_TIME_STAMP_Q_RAW_RECORD;
```

Members

x

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

y

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

z

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

a

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

e

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

r

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

s[3][3]

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

q[4]

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

time

The time variable is the time stamp for the data record and is returned as a **double** value. The integer portion of the variable represents the number of elapsed seconds since midnight, Jan 1, 1970, UTC, and is the standard way of representing time and date. If this is cast as a `time_t` structure, it can be converted into a date and time string using `ctime()` for example. The fractional part of the time variable represents fractions of a second.

quality

The quality value is a 16-bit unsigned integer. A very small quality number indicates no or minimal position and orientation errors due to distortion of the transmitter field depending on how sensitive you have set the error indicator. A large quality number indicates maximum error for the sensitivity level selected.

raw[3][3]

This is a 3x3 array of values. Each value is in the range +0.99997 to -1.00000. These values are the raw sensor values after they have been corrected for all known system error sources. This information is for factory use only.

Requirements

Windows NT/2000: Requires Windows 2000 or later.

Header: Declared in `ATC3DG.h`

Library: Use `ATC3DG.lib`

See Also

[DOUBLE_ANGLES_RECORD](#), [DOUBLE_MATRIX_RECORD](#), [DOUBLE_POSITION_RECORD](#),
[DOUBLE_QUATERNIONS_RECORD](#)

DOUBLE_POSITION_ANGLES_TIME_Q_BUTTON_RECORD

The **DOUBLE_POSITION_ANGLES_TIME_Q_BUTTON_RECORD** structure contains position Euler angle, timestamp, distortion and button information in double floating point format.

```
typedef struct tagDOUBLE_POSITION_ANGLES_TIME_Q_BUTTON{
    double      x;
    double      y;
    double      z;
    double      a;
    double      e;
    double      r;
    double      time;
    USHORT      quality;
    USHORT      button;
}DOUBLE_POSITION_ANGLES_TIME_Q_BUTTON_RECORD,*PDOUBLE_POSITION_ANGLES_TIME_Q_BUTTON_RECORD
;
```

Members

x

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

y

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

z

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

a

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

e

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

r

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

time

The time variable is the time stamp for the data record and is returned as a **double** value. The integer portion of the variable represents the number of elapsed seconds since midnight, Jan 1, 1970, UTC, and is the standard way of representing time and date. If this is cast as a time_t structure, it can be converted into a date and time string using ctime() for example. The fractional part of the time variable represents fractions of a second.

quality

The quality value is a 16-bit unsigned integer. A very small quality number indicates no or minimal position and orientation errors due to distortion of the transmitter field depending on how sensitive you have set the error indicator. A large quality number indicates maximum error for the sensitivity level selected.

button

The button value is a 16-bit unsigned integer that represents the state of an external contact closure that the user has been connected to the tracker (see [Rear Panel Connectors](#) for connector description). When a switch is connected, a 1 in the button value indicates the contact or switch is CLOSED, and a 0 indicates the contact is OPEN. The button line is sampled and available in this data record once per transmitter axis cycle - 3 times the system measurement rate for a mid or short-range transmitter.

Requirements

Windows NT/2000: Requires Windows 2000 or later.

Header: Declared in ATC3DG.h

Library: Use ATC3DG.lib

See Also

[DOUBLE POSITION RECORD](#), [DOUBLE ANGLES RECORD](#)

DOUBLE_POSITION_MATRIX_TIME_Q_BUTTON_RECORD

The **DOUBLE_POSITION_MATRIX_TIME_Q_BUTTON_RECORD** structure contains position, matrix, timestamp, distortion and button information in double floating point format.

```
typedef struct tagDOUBLE_POSITION_MATRIX_TIME_Q_BUTTON{
    double          x;
    double          y;
    double          z;
    double          s[3][3];
    double          time;
    USHORT          quality;
    USHORT          button;
}DOUBLE_POSITION_MATRIX_TIME_Q_BUTTON_RECORD,*PDOUBLE_POSITION_MATRIX_TIME_Q_BUTTON_RECORD
;
```

Members

x

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

y

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

z

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

s[3][3]

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

time

The time variable is the time stamp for the data record and is returned as a **double** value. The integer portion of the variable represents the number of elapsed seconds since midnight, Jan 1, 1970, UTC, and is the standard way of representing time and date. If this is cast as a time_t structure, it can be converted into a date and time string using ctime() for example. The fractional part of the time variable represents fractions of a second.

quality

The quality value is a 16-bit unsigned integer. A very small quality number indicates no or minimal position and orientation errors due to distortion of the transmitter field depending on how sensitive you have set the error indicator. A large quality number indicates maximum error for the sensitivity level selected.

button

The button value is a 16-bit unsigned integer that represents the state of an external contact closure that the user has been connected to the tracker (see [Rear Panel Connectors](#) for connector description). When a switch is connected, a 1 in the button value indicates the contact or switch is CLOSED, and a 0 indicates the contact is OPEN. The button line is sampled and available in this data record once per transmitter axis cycle - 3 times the system measurement rate for a mid or short-range transmitter.

Requirements

Windows NT/2000: Requires Windows 2000 or later.

Header: Declared in ATC3DG.h

Library: Use ATC3DG.lib

See Also

[DOUBLE_POSITION_RECORD](#), [DOUBLE_MATRIX_RECORD](#)

DOUBLE_POSITION_QUATERNION_TIME_Q_BUTTON_RECORD

The **DOUBLE_POSITION_QUATERNION_TIME_Q_BUTTON_RECORD** structure contains position, quaternion, timestamp, distortion and button information in double floating point format.

```
typedef struct tagDOUBLE_POSITION_QUATERNION_TIME_Q_BUTTON{
    double          x;
    double          y;
    double          z;
    double          q[4];
    double          time;
    USHORT          quality;
    USHORT          button;
}DOUBLE_POSITION_QUATERNION_TIME_Q_BUTTON_RECORD,*PDOUBLE_POSITION_QUATERNION_TIME_Q_BUTTON_RECORD;
```

Members

x

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

y

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

z

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

q[4]

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

time

The time variable is the time stamp for the data record and is returned as a **double** value. The integer portion of the variable represents the number of elapsed seconds since midnight, Jan 1, 1970, UTC, and is the standard way of representing time and date. If this is cast as a time_t structure, it can be converted into a date and time string using ctime() for example. The fractional part of the time variable represents fractions of a second.

quality

The quality value is a 16-bit unsigned integer. A very small quality number indicates no or minimal position and orientation errors due to distortion of the transmitter field depending on how sensitive you have set the error indicator. A large quality number indicates maximum error for the sensitivity level selected.

button

The button value is a 16-bit unsigned integer that represents the state of an external contact closure that the user has been connected to the tracker (see [Rear Panel Connectors](#) for connector description). When a switch is connected, a 1 in the button value indicates the contact or switch is CLOSED, and a 0 indicates the contact is OPEN. The button line is sampled and available in this data record once per transmitter axis cycle - 3 times the system measurement rate for a mid or short-range transmitter.

Requirements

Windows NT/2000: Requires Windows 2000 or later.

Header: Declared in ATC3DG.h

Library: Use ATC3DG.lib

See Also

[DOUBLE POSITION RECORD, DOUBLE QUATERNIONS RECORD](#)

3D Guidance Enumeration Types

The following enumeration types are used with the trakSTAR

[BIRD_ERROR_CODES](#)

[MESSAGE_TYPE](#)

[TRANSMITTER_PARAMETER_TYPE](#)

[SENSOR_PARAMETER_TYPE](#)

[BOARD_PARAMETER_TYPE](#)

[SYSTEM_PARAMETER_TYPE](#)

[HEMISPHERE_TYPE](#)

[AGC_MODE_TYPE](#)

[DATA_FORMAT_TYPE](#)

[BOARD_TYPES](#)

[DEVICE_TYPES](#)

BIRD_ERROR_CODES

The **BIRD_ERROR_CODES** enumeration type defines the values that the enumerated error code field of the ERRORCODE can be returned with from a function call.

```
enum BIRD_ERROR_CODES{
    BIRD_ERROR_SUCCESS=0,
    BIRD_ERROR_PCB_HARDWARE_FAILURE,
    BIRD_ERROR_TRANSMITTER_EEPROM_FAILURE,
    BIRD_ERROR_SENSOR_SATURATION_START,
    BIRD_ERROR_ATTACHED_DEVICE_FAILURE,
    BIRD_ERROR_CONFIGURATION_DATA_FAILURE,
    BIRD_ERROR_ILLEGAL_COMMAND_PARAMETER,
    BIRD_ERROR_PARAMETER_OUT_OF_RANGE,
    BIRD_ERROR_NO_RESPONSE,
    BIRD_ERROR_COMMAND_TIME_OUT,
    BIRD_ERROR_INCORRECT_PARAMETER_SIZE,
    BIRD_ERROR_INVALID_VENDOR_ID,
    BIRD_ERROR_OPENING_DRIVER,
    BIRD_ERROR_INCORRECT_DRIVER_VERSION,
    BIRD_ERROR_NO_DEVICES_FOUND,
    BIRD_ERROR_ACCESSING_PCI_CONFIG,
    BIRD_ERROR_INVALID_DEVICE_ID,
    BIRD_ERROR_FAILED_LOCKING_DEVICE,
    BIRD_ERROR_BOARD_MISSING_ITEMS,
    BIRD_ERROR_NOTHING_ATTACHED,
    BIRD_ERROR_SYSTEM_PROBLEM,
    BIRD_ERROR_INVALID_SERIAL_NUMBER,
    BIRD_ERROR_DUPLICATE_SERIAL_NUMBER,
    BIRD_ERROR_FORMAT_NOT_SELECTED,
    BIRD_ERROR_COMMAND_NOT_IMPLEMENTED,
    BIRD_ERROR_INCORRECT_BOARD_DEFAULT,
    BIRD_ERROR_INCORRECT_RESPONSE,
    BIRD_ERROR_NO_TRANSMITTER_RUNNING,
    BIRD_ERROR_INCORRECT_RECORD_SIZE,
    BIRD_ERROR_TRANSMITTER_OVERCURRENT,
    BIRD_ERROR_TRANSMITTER_OPEN_CIRCUIT,
    BIRD_ERROR_SENSOR_EEPROM_FAILURE,
    BIRD_ERROR_SENSOR_DISCONNECTED,
    BIRD_ERROR_SENSOR_REATTACHED,
    BIRD_ERROR_NEW_SENSOR_ATTACHED,
    BIRD_ERROR_UNDOCUMENTED,
    BIRD_ERROR_TRANSMITTER_REATTACHED,
    BIRD_ERROR_WATCHDOG,
    BIRD_ERROR_CPU_TIMEOUT_START,
    BIRD_ERROR_PCB_RAM_FAILURE,
    BIRD_ERROR_INTERFACE,
    BIRD_ERROR_PCB_EPROM_FAILURE,
    BIRD_ERROR_SYSTEM_STACK_OVERFLOW,
    BIRD_ERROR_QUEUE_OVERRUN,
    BIRD_ERROR_PCB_EEPROM_FAILURE,
    BIRD_ERROR_SENSOR_SATURATION_END,
    BIRD_ERROR_NEW_TRANSMITTER_ATTACHED,
    BIRD_ERROR_SYSTEM_UNINITIALIZED,
    BIRD_ERROR_12V_SUPPLY_FAILURE,
    BIRD_ERROR_CPU_TIMEOUT_END,
    BIRD_ERROR_INCORRECT_PLD,
    BIRD_ERROR_NO_TRANSMITTER_ATTACHED,
    BIRD_ERROR_NO_SENSOR_ATTACHED,
    BIRD_ERROR_SENSOR_BAD,
    BIRD_ERROR_SENSOR_SATURATED,
```

```

BIRD_ERROR_CPU_TIMEOUT,
BIRD_ERROR_UNABLE_TO_CREATE_FILE,
BIRD_ERROR_UNABLE_TO_OPEN_FILE,
BIRD_ERROR_MISSING_CONFIGURATION_ITEM,
BIRD_ERROR_MISMATCHED_DATA,
BIRD_ERROR_CONFIG_INTERNAL,
BIRD_ERROR_UNRECOGNIZED_MODEL_STRING,
BIRD_ERROR_INCORRECT_SENSOR,
BIRD_ERROR_INCORRECT_TRANSMITTER,
BIRD_ERROR_ALGORITHM_INITIALIZATION,
BIRD_ERROR_LOST_CONNECTION,
BIRD_ERROR_INVALID_CONFIGURATION,
BIRD_ERROR_TRANSMITTER_RUNNING,
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. *pciBIRD 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 InitializeBIRDSysyem 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: <ol style="list-style-type: none"> 1) Turn off the currently running transmitter and there is no transmitter attached to the system 2) Turn on the transmitter with the selected ID and there is no transmitter attached at that ID. |
| BIRD_ERROR_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 – InitializeBIRDSysyem() 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

Windows NT/2000: Requires Windows 2000 or later.

Header: Declared in ATC3DG.h

Library: Use ATC3DG.lib

See Also

SENSOR_PARAMETER_TYPE

The **SENSOR_PARAMETER_TYPE** enumeration type defines values that are used with the `GetSensorParameter` and `SetSensorParameter` functions to specify the operational characteristics of an individual sensor.

```
enum SENSOR_PARAMETER_TYPE{
    DATA_FORMAT,
    ANGLE_ALIGN,
    HEMISPHERE,
    FILTER_AC_WIDE_NOTCH,
    FILTER_AC_NARROW_NOTCH,
    FILTER_DC_ADAPTIVE,
    FILTER_ALPHA_PARAMETERS,
    FILTER_LARGE_CHANGE,
    QUALITY,
    SERIAL_NUMBER_RX,
    SENSOR_OFFSET,
    VITAL_PRODUCT_DATA_RX,
    VITAL_PRODUCT_DATA_PREAMP,
    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 38 and are measured with respect to rotations about the physical X, Y and Z axes of the sensor Figure 39. 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> |

ANGLE_ALIGN,
continued

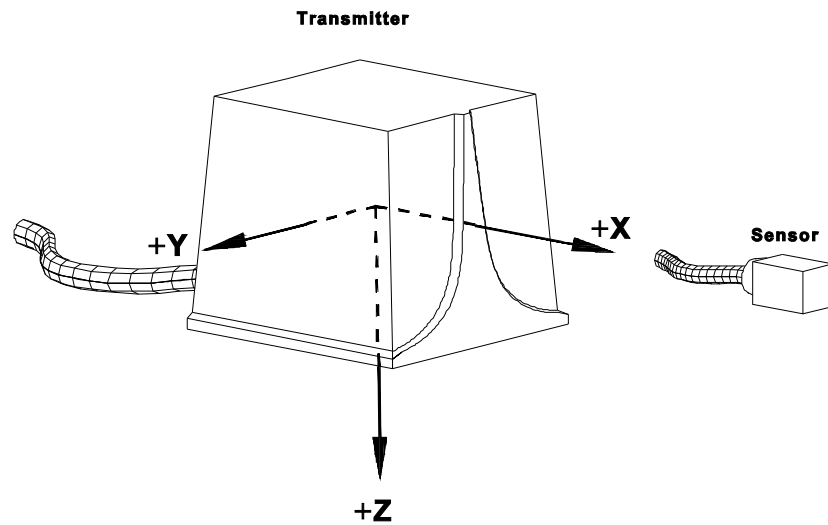


Figure 38 Measurement Reference Frame

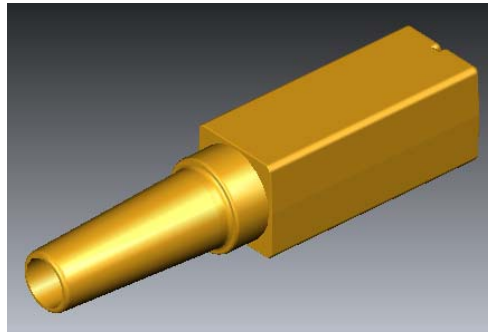


Figure 39 Receiver Zero Orientation (8mm Sensor)

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, centered 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.

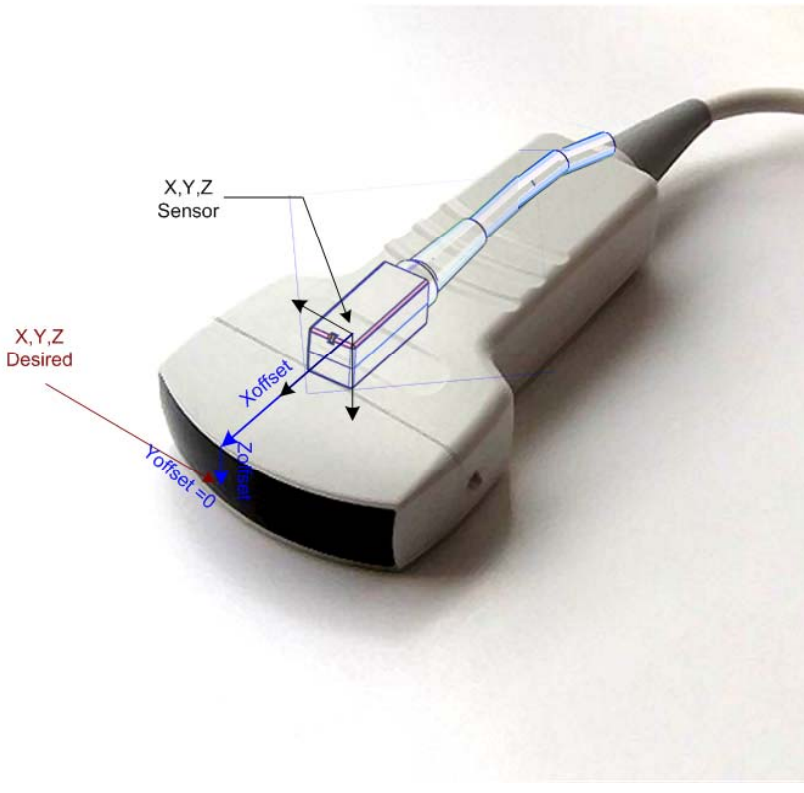
The ambiguity in position determination can be eliminated if your host computer's software continuously 'tracks' the sensor location. In order to implement tracking, you must understand the behavior of the signs (\pm) of the X, Y, and Z position outputs when the sensor crosses a hemisphere boundary. When you select a given

| | |
|------------------------|--|
| | <p>hemisphere of operation, the sign on the position axes that defines the hemisphere direction is forced to positive, even when the sensor moves into another hemisphere. For example, the power-up default hemisphere is the front hemisphere. This forces X position outputs to always be positive. The signs on Y and Z will vary between plus and minus depending on where you are within this hemisphere. If you had selected the bottom hemisphere, the sign of Z would always be positive and the signs on X and Y will vary between plus and minus. If you had selected the left hemisphere, the sign of Y would always be negative, etc.</p> <p>Regarding the default front hemisphere, if the sensor moved into the back hemisphere, the signs on Y and Z would instantaneously change to opposite polarities while the sign on X remained positive. To 'track' the sensor, your host software, on detecting this sign change, would reverse the signs on The Tracker's X, Y, and Z outputs. In order to 'track' correctly: You must start 'tracking' in the selected hemisphere so that the signs on the outputs are initially correct, and you must guard against the case where the sensor legally crossed the Y = 0, Z = 0 axes simultaneously without having crossed the X = 0 axes into the other hemisphere.</p> |
| FILTER_AC_WIDE_NOTCH | <p>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 30 and 72 hertz. If your application requires minimum transport delay between measurement of the sensor's position/orientation and the output of these measurements, you may want to evaluate the effect on your application with this filter shut off and the AC NARROW notch filter on.</p> |
| FILTER_AC_NARROW_NOTCH | <p>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.</p> |

| | |
|-------------------------|--|
| FILTER_DC_ADAPTIVE | <p>The DC filter refers to an adaptive, infinite impulse response (IIR) low pass filter applied to the sensor data to eliminate high frequency noise. Generally, this filter is always required in the system unless your application can work with noisy outputs. When the DC filter is enabled, you can modify its noise/lag characteristics by changing alphaMin and Vm.</p> <p>To use the default filter settings, just set the FILTER_DC_ADAPTIVE value to 1.0. To disable the filter set the value to 0.0</p> |
| FILTER_ALPHA_PARAMETERS | <p>To modify the filter characteristics, configure the elements of the structure ADAPTIVE_PARAMETERS.</p> <p>The alphaMin and alphaMax values define the lower and upper ends of the adaptive range that the filter constant alpha can assume in the DC filter, as a function of sensor to transmitter separation. When alphaMin = 0 Hex, the DC filter will provide an infinite amount of filtering (the outputs will never change even if you move the sensor). When alphaMin = 0.99996 = 7FFF Hex, the DC filter will provide no filtering of the data. At the shorter ranges you may want to increase alphaMin to obtain less lag while at longer ranges you may want to decrease alphaMin to provide more filtering (less noise/more lag). Note that alphaMin must always be less than alphaMax.</p> <p>When there is a fast motion of the sensor, the adaptive filter reduces the amount of filtering by increasing the ALPHA used in the filter. It will increase ALPHA only up to the limiting alphaMax value. By doing this, the lag in the filter is reduced during fast movements. When alphaMax = 0.99996 = 7FFF Hex, the DC filter will provide no filtering of the data during fast movements. Some users may want to decrease alphaMax to increase the amount of filtering if the Tracker's outputs are</p> |

| | |
|-------------------------------------|---|
| | <p>too noisy during rapid sensor movement.</p> <p>The 7 words that make up the Vm table values represent the expected noise that the DC filter will measure. By changing the table values, you can increase or decrease the DC filter's lag as a function of sensor range from the transmitter.</p> <p>The DC filter is adaptive in that it tries to reduce the amount of low pass filtering in the Tracker as it detects translation or rotation rates in the Tracker's sensor. Reducing the amount of filtering results in less filter lag.</p> <p>Unfortunately, electrical noise in the environment—when measured by the sensor—also makes it look like the sensor is undergoing a translation and rotation. As the sensor moves farther and farther away from the transmitter, the amount of noise measured by the sensor appears to increase because the measured transmitted signal level is decreasing and the sensor amplifier gain is increasing. In order to decide if the amount of filtering should be reduced, the Tracker has to know if the measured rate is a real sensor rate due to movement or a false rate due to noise. The Tracker gets this knowledge by the user specifying what the expected noise levels are in the operating environment as a function of distance from the transmitter. These noise levels are the 7 words that form the Vm table. The Vm values can range from 1 for almost no noise to 32767 for a lot of noise.</p> |
| FILTER_LARGE_CHANGE | <p>When the LARGE_CHANGE filter is selected, the position and orientation outputs are not allowed to change if the system detects a sudden large change in the outputs. Large undesirable changes may occur at large separation distances between the transmitter and sensor when the sensor undergoes a fast rotation or translation. If the LARGE_CHANGE value is TRUE the outputs will not be updated if a large change is detected. If value is FALSE, the outputs will change.</p> |
| <p>QUALITY</p> <p>QUALITY –cont</p> | <p>This data structure is used to adjust the output characteristics of the Quality number. Also referred to as the METAL error or Distortion number, this value is returned with certain data formats and gives the user an indication of the degree to which the position and angle measurements are in error. This error may be due to 'bad' metals located near the transmitter and sensor, or due to TRACKER 'system' errors. 'Bad' metals are metals with high electrical conductivity such as aluminum, or high magnetic permeability such as steel. 'Good' metals have low conductivity and low permeability such as 300 series stainless steel, or titanium. The 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.</p> <p>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. User beware. You need to evaluate your application for suitability of this metal detector.</p> <p>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.</p> <p>The QUALITYerror number will always show there is some error in the system</p> |

| | |
|---------------|--|
| QUALITY –cont | <p>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.</p> <p>The QUALITYError number that is output is computed from the following equation:</p> $\text{QUALITYError} = \text{Sensitivity} \times (\text{ErrorSYSTEM} - (\text{Offset} + \text{Slope} \times \text{Range}))$ <p>Where Range is the distance between the transmitter and sensor.</p> <p>Sensitivity</p> <p>The user supplies a Sensitivity value based on how little or how much he or she wants the QUALITYError value to reflect errors.</p> <p>Offset</p> <p>If you are trying to minimize the base errors in the system by adjusting the Offset you could set the Sensitivity =1, and the Slope=0 and read the Offset directly as the QUALITYError number.</p> <p>Slope</p> <p>You can determine the slope by setting the Sensitivity=1 and looking at the change in QUALITYError as you translate the sensor from range=0 to range max for the system (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.</p> <p>Alpha</p> <p>The QUALITYError value is filtered before output to the user to minimize noise jitter. The Alpha value determines how much filtering is applied to the error value. Range is FFFF -> 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.</p> |
|---------------|--|

| | |
|------------------|--|
| SERIAL_NUMBER_RX | Returns the serial number of the attached sensor. |
| SENSOR_OFFSETS | <p>Normally the position outputs from the 3DGuidance™ represent the x, y, z position of the center of the sensor with respect to the origin of the Transmitter. The SENSOR_OFFSETS command allows the user to specify a location that is offset from the center of the sensor.</p> <p>The x, y, z offset distances you supply in a DOUBLE_POSITION_RECORD with this command are measured in the sensor reference frame and are measured from the sensor center to the desired position on the tracked object. NOTE these values must be supplied in inches.</p>  <p>The diagram shows a 3D Guidance sensor, which is a white, rectangular device with a black cable attached. A blue wireframe box is superimposed on the sensor, representing the sensor's reference frame. A red arrow points from the text 'X,Y,Z Desired' to a point on the sensor's surface. Blue arrows indicate the offset distances: 'Xoffset' (horizontal), 'Yoffset=0' (vertical), and 'Zoffset' (depth). The text 'X,Y,Z Sensor' is also present, indicating the sensor's center position.</p> |

| | |
|---------------------------|--|
| VITAL_PRODUCT_DATA_RX | <p>Used to read or write to individual bytes in the Vital Product Data (VPD) storage area on the sensor. The VPD section comprises 128 bytes of user modifiable data storage. It is the user's responsibility to define the contents and structure and to maintain that structure. The parameter passed with these commands is a structure VPD_COMMAND_PARAMETERS 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 (GetSensorParameter), the value of the byte read from the VPD is placed in the value location in the structure.</p> <p>Note: Reading or writing VPD is not allowed when the transmitter is running.</p> |
| VITAL_PRODUCT_DATA_PREAMP | <p>Used to read or write to individual bytes in the Vital Product Data (VPD) storage area on the preamp. The VPD section comprises 128 bytes of user modifiable data storage. It is the user's responsibility to define the contents and structure and to maintain that structure. The parameter passed with these commands is a structure VPD_COMMAND_PARAMETERS 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 (GetSensorParameter), the value of the byte read from the VPD is placed in the value location in the structure.</p> <p>Note: Reading or writing VPD is not allowed when the transmitter is running.</p> |
| 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

See Also

MESSAGE_TYPE

The **MESSAGE_TYPE** enumeration type defines a value used with the `GetErrorText` function to define the type of message string returned from the call.

```
enum MESSAGE_TYPE{  
    SIMPLE_MESSAGE,  
    VERBOSE_MESSAGE  
};
```

| Enumerator Value | Meaning |
|------------------|--|
| SIMPLE_MESSAGE | The call <code>GetErrorText</code> will return a short terse message string describing the meaning of the error code passed. |
| VERBOSE_MESSAGE | The call <code>GetErrorText</code> will return a message string containing a full and comprehensive description of the problem and possible resolutions where appropriate. |

Requirements

Windows NT/2000: Requires Windows 2000 or later.

Header: Declared in `ATC3DG.h`

Library: Use `ATC3DG.lib`

See Also

TRANSMITTER_PARAMETER_TYPE

The **TRANSMITTER_PARAMETER_TYPE** enumeration type defines values that are used with the `GetTransmitterParameter` and `SetTransmitterParameter` functions to specify the operational characteristics of an individual transmitter.

```
enum TRANSMITTER_PARAMETER_TYPE{
    SERIAL_NUMBER_TX,
    REFERENCE_FRAME,
    XYZ_REFERENCE_FRAME,
    VITAL_PRODUCT_DATA_TX,
    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. 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="505 1161 1356 1648" data-label="Image"> <p>The diagram illustrates the standard measurement reference frame for a transmitter. It shows a 3D perspective of a transmitter unit. A coordinate system is defined with three axes: +X pointing to the right, +Y pointing to the left, and +Z pointing downwards. A cable labeled 'Sensor' is connected to the right side of the transmitter.</p> </div> <p>Figure 40 Measurement Reference Frame (Standard Transmitter)</p> |

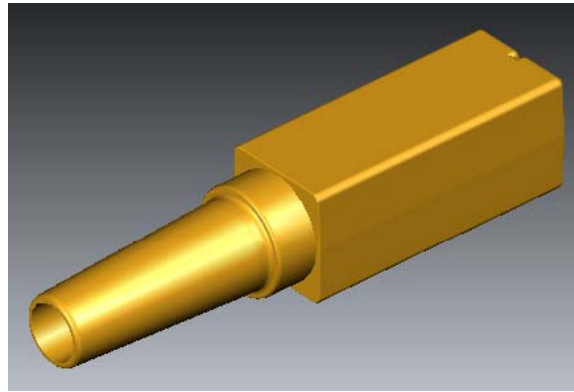


Figure 41 Receiver Zero Orientation (8mm Sensor)

| | |
|-----------------------|--|
| | |
| 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 VPD_COMMAND_PARAMETERS which contains the address of the target byte and, in the case of the write command (SetTransmitterParameter) the value of the byte to be written. In the case of the read command (GetTransmitterParameter) 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

Windows NT/2000: Requires Windows 2000 or later.

Header: Declared in ATC3DG.h

Library: Use ATC3DG.lib

See Also

BOARD_PARAMETER_TYPE

The **BOARD_PARAMETER_TYPE** enumeration type defines parameters that can be changed and/or inspected with the `GetBoardParameter` and `SetBoardParameter` functions. These parameters control the operational characteristics of the board. One of these enumerated values is passed as a parameter to the call to indicate the type and size of the actual parameter passed. The table below describes the actual type and size and purpose of the parameters passed for each of these types.

```
enum BOARD_PARAMETER_TYPE{
    SERIAL_NUMBER_PCB,
    BOARD_SOFTWARE_REVISIONS,
    POST_ERROR_PCB,
    DIAGNOSTIC_TEST_PCB,
    VITAL_PRODUCT_DATA_PCB,
    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 BOARD_REVISIONS structure. |
| POST_ERROR_PCB | Used to examine results of the POST (Power ON Self Test). See Parameter Structure POST_ERROR_PARAMETER |
| DIAGNOSTIC_TEST_PCB | Used to execute diagnostic tests. May also be used to acquire information on available diagnostic tests. See Parameter structure: DIAGNOSTIC_TEST_PARAMETERS |
| 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 VPD_COMMAND_PARAMETERS which contains the address of the target byte and, in the case of the write command (SetBoardParameter) the value of the byte to be written. In the case of the read command (GetBoardParameter), 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

Windows NT/2000: Requires Windows 2000 or later.

Header: Declared in ATC3DG.h

Library: Use ATC3DG.lib

See Also

SYSTEM_PARAMETER_TYPE

The **SYSTEM_PARAMETER_TYPE** enumeration type defines parameters that can be changed and/or inspected with the `GetSystemParameter` and `SetSystemParameter` functions. These parameters control the operational characteristics of the system. One of these enumerated values is passed as a parameter to the call to indicate the type and size of the actual parameter passed. The table below describes the actual type and size and purpose of the parameters passed for each of these types.

```
enum SYSTEM_PARAMETER_TYPE{
    SELECT_TRANSMITTER,
    POWER_LINE_FREQUENCY,
    AGC_MODE,
    MEASUREMENT_RATE,
    MAXIMUM_RANGE,
    METRIC,
    VITAL_PRODUCT_DATA,
    POST_ERROR,
    DIAGNOSTIC_TEST,
    REPORT_RATE,
    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 | Informs the hardware of the frequency of the AC power source. The parameter passed is a double value describing the frequency in Hz. There are only two valid values: either 50.0 or 60.0 Hz. |
| AGC_MODE | Select the automatic gain control (AGC) mode. The parameter passed is one of the enumerated type <code>AGC_MODE_TYPE</code> . |
| MEASUREMENT_RATE | Set the measurement rate. The parameter passed is a double value and represents the measurement rate in Hz. The valid range of values is $20.0 < \text{rate} < 255.0$ |
| 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 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 (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 POST_ERROR_PARAMETER |
| DIAGNOSTIC_TEST* | Used to execute diagnostic tests. May also be used to acquire information on available diagnostic tests. See Parameter structure: DIAGNOSTIC_TEST_PARAMETERS |
| 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: COMMUNICATIONS_MEDIA_PARAMETER 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). For accessing this information/functionality across all units, see the [Board Parameters](#).

Requirements

Windows NT/2000: Requires Windows 2000 or later.

Header: Declared in ATC3DG.h

Library: Use ATC3DG.lib

See Also

HEMISPHERE_TYPE

The **HEMISPHERE_TYPE** enumeration type defines values that are used when setting the HEMISPHERE with the SetSensorParameter call. The default HEMISPHERE_TYPE is FRONT.

```
enum HEMISPHERE_TYPE{
    FRONT,
    BACK,
    TOP,
    BOTTOM,
    LEFT,
    RIGHT
};
```

| Enumerator Value | Meaning |
|------------------|---|
| FRONT | The FRONT is the forward hemisphere in front of the transmitter. The front of the transmitter is the side with the Ascension logo molded into the case. It is the side opposite the side with the 2 positioning holes. This is the default. |
| BACK | The BACK is the opposite hemisphere to the FRONT hemisphere. |
| TOP | The TOP hemisphere is the upper hemisphere. When the transmitter is sitting on a flat surface with the locating holes on the surface the TOP hemisphere is above the transmitter. |
| BOTTOM | The BOTTOM hemisphere is the opposite hemisphere to the TOP hemisphere. |
| LEFT | The LEFT hemisphere is the hemisphere to the left of the observer when looking at the transmitter from the back. |
| RIGHT | The RIGHT hemisphere is the opposite hemisphere to the LEFT hemisphere. The LEFT hemisphere is on the left side of the observer when looking at the transmitter from the back. |

Requirements

Windows NT/2000: Requires Windows 2000 or later.

Header: Declared in ATC3DG.h

Library: Use ATC3DG.lib

See Also

AGC_MODE_TYPE

The **AGC_MODE_TYPE** enumeration type defines values that are used when setting the AGC_MODE with the SetSensorParameter call. The default is TRANSMITTER_AND_SENSOR_AGC.

```
enum AGC_MODE_TYPE{
    TRANSMITTER_AND_SENSOR_AGC,
    SENSOR_AGC_ONLY
};
```



Note: Transmitter AGC not active in 3DGuidance Svstems.

| Enumerator Value | Meaning |
|----------------------------|--|
| TRANSMITTER_AND_SENSOR_AGC | Select both transmitter power switching and sensor gain control for the AGC implementation. This is the default. NOTE: As the sensor moves away from the transmitter the signal decreases so it is necessary to increase the gain of the sensor amplifier. As the sensor approaches the transmitter the signal increases so the sensor amp gain needs to be reduced. But, there comes a point where the sensor is so close to the transmitter that the signal saturates the sensor and at that point it becomes necessary to reduce the power of the transmitter. Doing this allows the sensor to be used close to the transmitter. All transmitter power switching and sensor amp gain control is handled automatically for the user. |
| SENSOR_AGC_ONLY | Disable transmitter power switching and use only sensor gain control for the AGC implementation. NOTE: When the power switching is disabled the transmitter will run at full power all the time. This means that there comes a point at which the sensor as it is approaching the transmitter will saturate. Since the transmitter will not reduce power this is the minimum limiting range for this mode of operation. |

Requirements

Windows NT/2000: Requires Windows 2000 or later.

Header: Declared in ATC3DG.h

Library: Use ATC3DG.lib

See Also

DATA_FORMAT_TYPE

The **DATA_FORMAT_TYPE** enumeration type

```
enum DATA_FORMAT_TYPE{
    NO_FORMAT_SELECTED=0,
    SHORT_POSITION,
    SHORT_ANGLES,
    SHORT_MATRIX,
    SHORT_QUATERNIONS,
    SHORT_POSITION_ANGLES,
    SHORT_POSITION_MATRIX,
    SHORT_POSITION_QUATERNION,
    DOUBLE_POSITION,
    DOUBLE_ANGLES,
    DOUBLE_MATRIX,
    DOUBLE_QUATERNIONS,
    DOUBLE_POSITION_ANGLES,
    DOUBLE_POSITION_MATRIX,
    DOUBLE_POSITION_QUATERNION,
    DOUBLE_POSITION_TIME_STAMP,
    DOUBLE_ANGLES_TIME_STAMP,
    DOUBLE_MATRIX_TIME_STAMP,
    DOUBLE_QUATERNIONS_TIME_STAMP,
    DOUBLE_POSITION_ANGLES_TIME_STAMP,
    DOUBLE_POSITION_MATRIX_TIME_STAMP,
    DOUBLE_POSITION_QUATERNION_TIME_STAMP,
    DOUBLE_POSITION_TIME_Q,
    DOUBLE_ANGLES_TIME_Q,
    DOUBLE_MATRIX_TIME_Q,
    DOUBLE_QUATERNIONS_TIME_Q,
    DOUBLE_POSITION_ANGLES_TIME_Q,
    DOUBLE_POSITION_MATRIX_TIME_Q,
    DOUBLE_POSITION_QUATERNION_TIME_Q,
    SHORT_ALL,
    DOUBLE_ALL,
    DOUBLE_ALL_TIME_STAMP,
    DOUBLE_ALL_TIME_STAMP_Q,
    DOUBLE_ALL_TIME_STAMP_Q_RAW,
    DOUBLE_POSITION_ANGLES_TIME_Q_BUTTON,
    DOUBLE_POSITION_MATRIX_TIME_Q_BUTTON,
    DOUBLE_POSITION_QUATERNION_TIME_Q_BUTTON,
    MAXIMUM_FORMAT_CODE
};
```

| Enumerator Value | Selects Data Record of Structure Type: |
|------------------------|--|
| NO_FORMAT_SELECTED=0, | No data format selected. |
| SHORT_POSITION, | SHORT_POSITION_RECORD |
| SHORT_ANGLES, | SHORT_ANGLES_RECORD |
| SHORT_MATRIX, | SHORT_MATRIX_RECORD |
| SHORT_QUATERNIONS, | SHORT_QUATERNIONS_RECORD |
| SHORT_POSITION_ANGLES, | SHORT_POSITION_ANGLES_RECORD |
| SHORT_POSITION_MATRIX, | SHORT_POSITION_MATRIX_RECORD |

| | |
|--|--|
| SHORT_POSITION_QUATERNION, | SHORT_POSITION_QUATERNION_RECORD |
| DOUBLE_POSITION, | DOUBLE_POSITION_RECORD |
| DOUBLE_ANGLES, | DOUBLE_ANGLES_RECORD |
| DOUBLE_MATRIX, | DOUBLE_MATRIX_RECORD |
| DOUBLE_QUATERNIONS, | DOUBLE_QUATERNIONS_RECORD |
| DOUBLE_POSITION_ANGLES, | DOUBLE_POSITION_ANGLES_RECORD |
| DOUBLE_POSITION_MATRIX, | DOUBLE_POSITION_MATRIX_RECORD |
| DOUBLE_POSITION_QUATERNION, | DOUBLE_POSITION_QUATERNION_RECORD |
| DOUBLE_POSITION_TIME_STAMP, | DOUBLE_POSITION_TIME_STAMP_RECORD |
| DOUBLE_ANGLES_TIME_STAMP, | DOUBLE_ANGLES_TIME_STAMP_RECORD |
| DOUBLE_MATRIX_TIME_STAMP, | DOUBLE_MATRIX_TIME_STAMP_RECORD |
| DOUBLE_QUATERNIONS_TIME_STAMP, | DOUBLE_QUATERNIONS_TIME_STAMP_RECORD |
| DOUBLE_POSITION_ANGLES_TIME_STAMP, | DOUBLE_POSITION_ANGLES_TIME_STAMP_RECORD |
| DOUBLE_POSITION_MATRIX_TIME_STAMP, | DOUBLE_POSITION_MATRIX_TIME_STAMP_RECORD |
| DOUBLE_POSITION_QUATERNION_TIME_STAMP, | DOUBLE_POSITION_QUATERNION_TIME_STAMP_RECORD |
| DOUBLE_POSITION_TIME_Q, | DOUBLE_POSITION_TIME_Q_RECORD |
| DOUBLE_ANGLES_TIME_Q, | DOUBLE_ANGLES_TIME_Q_RECORD |
| DOUBLE_MATRIX_TIME_Q, | DOUBLE_MATRIX_TIME_Q_RECORD |
| DOUBLE_QUATERNIONS_TIME_Q, | DOUBLE_QUATERNIONS_TIME_Q_RECORD |
| DOUBLE_POSITION_ANGLES_TIME_Q, | DOUBLE_POSITION_ANGLES_TIME_Q_RECORD |
| DOUBLE_POSITION_MATRIX_TIME_Q, | DOUBLE_POSITION_MATRIX_TIME_Q_RECORD |
| DOUBLE_POSITION_QUATERNION_TIME_Q, | DOUBLE_POSITION_QUATERNION_TIME_Q_RECORD |
| SHORT_ALL, | SHORT_ALL_RECORD |
| DOUBLE_ALL, | DOUBLE_ALL_RECORD |
| DOUBLE_ALL_TIME_STAMP, | DOUBLE_ALL_TIME_STAMP_RECORD |
| DOUBLE_ALL_TIME_STAMP_Q, | DOUBLE_ALL_TIME_STAMP_Q_RECORD |
| DOUBLE_ALL_TIME_STAMP_Q_RAW, | DOUBLE_ALL_TIME_STAMP_Q_RAW_RECORD |
| DOUBLE_POSITION_ANGLES_TIME_Q_BUTTON | DOUBLE_POSITION_ANGLES_TIME_Q_BUTTON_RECORD |
| DOUBLE_POSITION_MATRIX_TIME_Q_BUTTON | DOUBLE_POSITION_MATRIX_TIME_Q_BUTTON |
| DOUBLE_POSITION_QUATERNION_TIME_Q_BUTTON | DOUBLE_POSITION_QUATERNION_TIME_Q_BUTTON |
| MAXIMUM_FORMAT_CODE | End of table place holder |

Requirements

Windows NT/2000: Requires Windows 2000 or later.

Header: Declared in ATC3DG.h

Library: Use ATC3DG.lib

See Also

BOARD_TYPES

A value of the **BOARD_TYPES** enumeration type is returned from a call to GetBoardConfiguration in the *type* parameter location of the structure BOARD_CONFIGURATION.

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

Windows NT/2000: Requires Windows 2000 or later.

Header: Declared in ATC3DG.h

Library: Use ATC3DG.lib

See Also

DEVICE_TYPES

A value of the **DEVICE_TYPES** enumeration type is returned in the *type* parameter location of either the **SENSOR_CONFIGURATION** or the **TRANSMITTER_CONFIGURATION** structures which are returned from a call to either **GetSensorConfiguration** or **GetTransmitterConfiguration**.

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/traksSTAR 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/traksSTAR sensor (bayBIRD) |

Requirements

Windows NT/2000: Requires Windows 2000 or later.

Header: Declared in ATC3DG.h

Library: Use ATC3DG.lib

See Also

COMMUNICATIONS_MEDIA_TYPES

A value of the **COMMUNICATIONS_MEDIA_TYPES** enumeration type is set or returned in the *mediaType* field of the [COMMUNICATIONS_MEDIA_PARAMETER](#) 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

Windows NT/2000: Requires Windows 2000 or later.

Header: Declared in `ATC3DG.h`

Library: Use `ATC3DG.lib`

See Also

3D Guidance API Status/Error Bit Definitions

The following bit definitions are used with the trakSTAR

[ERRORCODE](#)

[DEVICE_STATUS](#)

ERRORCODE

The **ERRORCODE** *int* has the following format:

| Bit | Meaning |
|---------|---|
| 0-15 | Enumerated error code of type BIRD_ERROR_CODES |
| 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 InitializeBIRDSys | x | x | x | x |
| 7 | NO_TRANSMITTER_RUNNING | An attempt was made to call GetAsynchronousRecord 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.

3D Guidance Initialization Files

The Initialization File is used to set a trakSTAR to a predetermined state.

3D Guidance Initialization File Format Reference

The following sections describe the syntax and meaning of the items used in each type of initialization file section. Initialization files must follow these general rules:

- ❖ Sections begin with the section name enclosed in brackets.
- ❖ A **System** section must be included in any initialization file used with the 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)

[\[Sensor*x*\]](#) (Where *x* is replaced with a decimal number representing the *id* of the sensor.)

[\[Transmitter*x*\]](#) (Where *x* is replaced with a decimal number representing the *id* of the transmitter.)

[System]

The **System** section must be included in all initialization files formatted for use with the 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
```

[Sensorx]

The **Sensor** section is optional.

```
[Sensorx]
Format=format-type
Hemisphere=hemisphere-type
AC_Narrow_Filter=narrow-flag
AC_Wide_Filter=wide-flag
DC_Filter=dc-flag
Alpha_Min=min-params
Alpha_Max=max-params
Vm=vm-params
Angle_Align=align-angles
Filter_Large_Change=change-flag
Distortion=distortion-params
```

Format-type

This parameter takes the form of 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
```

[Transmitter x]

The **Transmitter** section is optional.

```
[Transmitter $x$ ]  
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
```


Ascension RS232 Interface Reference

*This chapter details the method for communicating with the **trakSTAR 2** directly using the RS232 interface protocol and command set.*

RS232 Signal Description

A pinout and signal description of the RS-232C interface is found below. Note that **trakSTAR 2** requires connections only to pins 2, 3 and 5 of the 9-pin interface connector.

Table 8 TrakSTAR 2 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 | Tracker to host Signal reference |
| 7 | Request to Sent | Host to Tracker | Holds The Tracker in RESET when high |

Note: These are the Electronic Industries Association (EIA) RS232 signals names. The Tracker is configured as Data Communication equipment (DCE) and therefore Transmit Data is an input and Receive Data is an output.

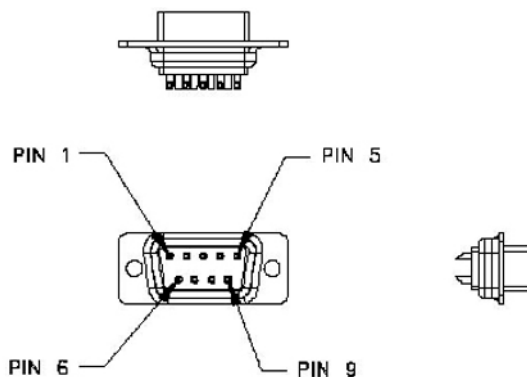


Figure 42 Rear View: 9-Pin D-Sub Connector

Using the 'reset on CTS' feature

The **trakSTAR 2** 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: Users running applications in a Windows® environment who would like to enable this feature, should take steps to ensure that command of this line (and of the serial port) is clearly asserted.

RS232 Commands

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

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

| MS BIT | | | | | | | | | | LS BIT |
|--------|---|---|---|---|---|---|---|---|-------|--------|
| Stop | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | Start | |

RS232

| | | | | | | | | | | |
|---------|---|-----|-----|-----|-----|-----|-----|-----|-----|---|
| Command | 1 | BC7 | BC6 | BC5 | BC4 | BC3 | BC2 | BC1 | BC0 | 0 |
|---------|---|-----|-----|-----|-----|-----|-----|-----|-----|---|

- where BC7-BC0 is the 8-bit command value (see RS232 Command Reference)
- and the MS BIT (Stop = 1) and LS BIT (Start = 0) refers to the bit values that the UART in your computer's RS232 port automatically inserts into the serial data stream.

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

| MS BIT | | | | | | | | | | LS BIT |
|--------|---|---|---|---|---|---|---|---|-------|--------|
| Stop | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | Start | |

RS232

| | | | | | | | | | | |
|------|---|-----|-----|-----|-----|-----|-----|-----|-----|---|
| Data | 1 | BD7 | BD6 | BD5 | BD4 | BD3 | BD2 | BD1 | BD0 | 0 |
|------|---|-----|-----|-----|-----|-----|-----|-----|-----|---|

- where BD7-BD0 is the 8-bit data value associated with a given command.

Command Summary

The following summarizes the action of each RS232 command. The details of command usage are presented later in this chapter in the section entitled [RS232 Command Reference](#).


 **Tip:** For a listing of valid system parameters to use with the **CHANGE** or **EXAMINE VALUE** commands, see **CHANGE VALUE** in the 'Command Reference' section.

Table 9 RS232 Command Summary

| Command | Description |
|--------------------------------|--|
| ANGLES | Data record contains 3 rotation angles |
| ANGLE ALIGN | Aligns sensor to reference direction |
| BORESIGHT (See Note) | Aligns sensor to the reference frame |
| BORESIGHT REMOVE (See Note) | 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 center of the Sensor. |
| POINT | One data record is output from the trakSTAR unit 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 |

| Command | Description |
|-----------------|---|
| 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 tracker. |
| STREAM STOP | Stops any data output that was started with the STREAM |

Note: Commands not implemented at time of manual writing

Command Utilization

The host may tell the tracker 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 trakSTAR's output characteristics.

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

Table 10 Hypothetical Command Sequence

| Command | Description |
|---------|---|
| RUN | Turn on the transmitter (needed if system configured to sleep on reset) |
| ANGLES | Output records will contain angles only |

| Command | Description |
|---------|--|
| POINT | Tracker outputs ANGLES data record. |
| STREAM | ANGLE data records start streaming from tracker and will not stop until the mode is changed to POINT or the STREAM STOP command is issued. |
| POINT | An ANGLE data record is output and the streaming is stopped |

Response Format

Two types of binary data are returned from **trakSTAR 2**:

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) that 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 | 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 |

3D GUIDANCE TRAKSTAR 2™ INSTALLATION AND OPERATION GUIDE

| | | | | | | | | |
|---|-----|-----|-----|-----|-----|-----|----|-----------|
| 0 | C15 | C14 | C13 | C12 | C11 | C10 | C9 | #2 MSbyte |
| 0 | . | . | . | . | . | . | . | . |
| 0 | . | . | . | . | . | . | . | . |
| 0 | . | . | . | . | . | . | . | . |
| 0 | N8 | N7 | N6 | N5 | N4 | N5 | N2 | #N LSbyte |
| 0 | N15 | N14 | N13 | N12 | N11 | N10 | N9 | #N MSbyte |

The MS (most significant) bits are the phasing bits, and are not part of the data.

For example, the tracker is about to send a data record consisting of these three data words:

| Word# | Decimal | Hex | Binary (2 bytes) | |
|-------|---------|------|------------------|----------|
| | | | MSbyte | LSbyte |
| #1 | 4386 | 1122 | 00010001 | 00100010 |
| #2 | 13124 | 3344 | 00110011 | 01000100 |
| #3 | 21862 | 5566 | 01010101 | 01100110 |

The conversion to the binary data format by the tracker is as follows:

Tracker

1) Shifts each data word right

2) Breaks each word into MSByte

| one bit | | LSByte pairs | |
|----------|----------|--------------|----|
| MS | LS | | |
| 00001000 | 10010001 | 10010001 | LS |
| 00011001 | 10100010 | 00001000 | MS |
| 00101010 | 10110011 | 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

| MS BIT | 7 | 6 | 5 | 4 | 3 | 2 | LS BIT | 1 | 0 | WORD # |
|--------|---|---|---|---|---|---|--------|---|---|-----------|
| 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | #1 LSByte |
| 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | #1 MSByte |
| 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | #2 LSByte |
| 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | #2 MSByte |
| 0 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | #3 LSByte |
| 0 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | #3 MSByte |

The user's computer can identify the beginning of the data record by catching the leading "1", and converting subsequent data bytes back to their proper binary values.

HOST:

- 1) Receives data bytes in stream after catching first marked "1" (Changes that "1" back to a "0")
- 2) Shifts each LSByte left one bit

| | | | |
|----------|----|----------|----|
| 01001000 | LS | 10010000 | LS |
| 00001000 | MS | 00001000 | MS |
| 01010001 | LS | 10100010 | LS |
| 00011001 | MS | 00011001 | MS |
| 01011001 | LS | 10110010 | LS |
| 00101010 | MS | 00101010 | MS |

- 3) Combines each MSByte/LSByte pair left one more into data words correct original
- 4) Shifts each word bit, giving the binary value

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

You don't need to worry about the fact that the two least significant bits are different because the data words do not use these bits.

RS232 Command Reference

All commands are listed alphabetically in the following section. Each command description contains the command codes required to initiate the commands, as well as the format and scaling of the data records that the **trakSTAR** 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 | | | | BYTE # |
|-----|-----|-----|-----|-----|-----|-----|----|----------------|
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
| 1 | Z8 | Z7 | Z6 | Z5 | Z4 | Z3 | Z2 | #1 LSbyte Zang |
| 0 | Z15 | Z14 | Z13 | Z12 | Z11 | Z10 | Z9 | #2 MSbyte Zang |
| 0 | Y8 | Y7 | Y6 | Y5 | Y4 | Y3 | Y2 | #3 LSbyte Yang |
| 0 | Y15 | Y14 | Y13 | Y12 | Y11 | Y10 | Y9 | #4 MSbyte Yang |
| 0 | X8 | X7 | X6 | X5 | X4 | X3 | X2 | #5 LSbyte Xang |
| 0 | X15 | X14 | X13 | X12 | X11 | X10 | X9 | #6 MSbyte Xang |

Zang (Azimuth) takes on values between the binary equivalent of +/- 180 degrees. Yang (Elevation) takes on values between +/- 90 degrees, and Xang (Roll) takes on values between +/- 180 degrees. As Yang (Elevation) approaches +/- 90 degrees, the Zang (Azimuth) and Xang (Roll) become very noisy and exhibit large errors. At 90 degrees the Zang (Azimuth) and Xang (Roll) become undefined. This behavior is not a limitation of The Tracker - 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 [Figure 26: Default Transmitter/Sensor Coordinate Frames](#) and are measured with respect to rotations about the physical X, Y and Z axes of the sensor. The ANGLE ALIGN1 command allows you to mathematically change each sensor's X, Y and Z axes to an orientation that differs from that of the actual sensor.



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 | | | | BYTE # |
|-----|-----|-----|-----|-----|-----|----|----|-----------------|
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
| 0 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | #1 Command Byte |
| B7 | B6 | B5 | B4 | B3 | B2 | B1 | B0 | #2 LSbyte A |
| B15 | B14 | B13 | B12 | B11 | B10 | B9 | B8 | #3 MSbyte A |
| B7 | B6 | B5 | B4 | B3 | B2 | B1 | B0 | #4 LSbyte E |
| B15 | B14 | B13 | B12 | B11 | B10 | B9 | B8 | #5 MSbyte E |
| B7 | B6 | B5 | B4 | B3 | B2 | B1 | B0 | #6 LSbyte R |
| B15 | B14 | B13 | B12 | B11 | B10 | B9 | B8 | #7 MSbyte R |

See the ANGLES command for the format and scaling of the angle values sent

BORESIGHT

| | ASCII | HEX | DECIMAL | BINARY |
|--------------|-------|-----|---------|----------|
| Command Byte | u | 75 | 117 | 01110101 |



Note: Command not implemented at time of manual writing.

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 |

Sending the single byte BORESIGHT REMOVE command to the tracker causes the sensor's orientation outputs to revert to their values before you sent the BORESIGHT command. That is, if there were no ANGLE ALIGN values present, the sensor's orientation outputs will now be with respect to the sensor's physical orientation. If there were ANGLE ALIGN values present before the BORESIGHT command was given, then after the BORESIGHT REMOVE command is given, the sensor's orientation outputs will be with respect to this mathematically defined ANGLE ALIGNed sensor orientation.



Note: Command not implemented at time of manual writing.

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 a rear panel external button will be reported to the host computer (see [Table 4](#)). 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 = “0” or “1.”

For a description of the values that 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 (see [Table 4](#)). 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 CHANGE VALUE and EXAMINE VALUE PARAMETER numbers are listed in the table below. *Note: not all PARAMETERnumbers are CHANGEable, but ALL are EXAMINEable.*

Table 11 List of Examine Value Valid Parameters

| Parameter | | CHANGEable | CHANGE | EXAMINE | | Description |
|-----------|-----|------------|------------|---------|---------|------------------------------------|
| | | | Bytes Send | Bytes | | |
| | | | | Send | Receive | |
| Dec | Hex | | | | | |
| 0 | 0 | No | 0 | 2 | 2 | Tracker status |
| 1 | 1 | No | 0 | 2 | 2 | Software revision number |
| 2 | 2 | No | 0 | 2 | 2 | Tracker computer crystal speed |
| 3 | 3 | Yes | 4 | 2 | 2 | Position scaling |
| 4 | 4 | Yes | 4 | 2 | 2 | Filter on/off status |
| 5 | 5 | Yes | 16 | 2 | 14 | DC Filter constant table ALPHA_MIN |
| 8 | 8 | Yes | 3 | 2 | 1 | Data ready output character |
| 9 | 9 | Yes | 3 | 2 | 1 | Changes data ready character |
| 10 | A | No | 0 | 2 | 1 | Tracker outputs an error code |
| 12 | C | Yes | 16 | 2 | 14 | DC filter constant table Vm |
| 13 | D | Yes | 16 | 2 | 14 | DC filter constant table ALPHA_MAX |
| 14 | E | Yes | 3 | 2 | 1 | Sudden output change elimination |
| 15 | F | No | 0 | 2 | 10 | System Model Identification |
| 17 | 11 | Yes | 3 | 2 | 1 | XYZ Reference Frame |
| 20 | 14 | Yes | 3 | 2 | 1 | Filter line frequency |
| 22 | 16 | Yes | 4 | 2 | 2 | Change/Examine Hemisphere |
| 23 | 17 | Yes | 8 | 2 | 6 | Change/Examine Angle Align2 |

3D GUIDANCE TRAKSTAR 2™ INSTALLATION AND OPERATION GUIDE

| Parameter | | CHANGEable | CHANGE | EXAMINE | | Description |
|-----------|----|------------|------------|---------|---------|---------------------------------------|
| | | | Bytes Send | Bytes | | |
| | | | | Send | Receive | |
| 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 POsServer 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 POsServer Firmware Revision |

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

| MSB | | | | | | | LSB | | |
|--------|----|----|----|----|----|----|-----|------|-------------------|
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | BYTE | # |
| 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | #1 | Command Byte, 'P' |
| N7 | N6 | N5 | N4 | N3 | N2 | N1 | N0 | #2 | PARAMETERnumber |
| B7 | B6 | B5 | B4 | B3 | B2 | B1 | B0 | #3 | PARAMETERdata |
| LSbyte | | | | | | | | | |
| B7 | B6 | B5 | B4 | B3 | B2 | B1 | B0 | #4 | PARAMETERdata |
| MSbyte | | | | | | | | | |
| B7 | B6 | B5 | B4 | B3 | B2 | B1 | B0 | #N | PARAMETERdata |

Where N7-N0 represent a PARAMETERnumber (i.e. 00000011 or 00000100), and B7-B0 represent N-bytes of PARAMETERdata. If the PARAMETERdata is a word then the Least Significant byte (LSbyte) is transmitted before the Most Significant byte (MSbyte). If the PARAMETERdata is numeric, it must be in 2's complement format. You do not shift and add 'phasing' bits to the data.

The **EXAMINE VALUE** command must be issued to The Tracker in the following 2-byte sequence:

| MSB | | | | | | | LSB | | |
|-----|----|----|----|----|----|----|-----|------|-----------------|
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | BYTE | # |
| 0 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | #1 | Command Byte |
| N7 | N6 | N5 | N4 | N3 | N2 | N1 | N0 | #2 | PARAMETERnumber |

Where N7-N0 represent a PARAMETERnumber, i.e. 00000000 or 00000001, etc.

If the PARAMETERdata returned is a word then the Least Significant byte (LSbyte) is received before the Most Significant byte (MSbyte). If the PARAMETERdata is numeric, it is in 2's complement format. The PARAMETERdata received does not contain 'phasing' bits. The 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:



Tip: To retrieve the error code(s) indicated by this 'flag' (B13), send '10' as the parameter (see [ERROR CODE](#)).

Table 12 Tracker Status Bits

| Bit # | Description |
|--------|--|
| B15 | 1 – If Tracker is “Master” Tracker 0 – If Tracker is “Slave” Tracker |
| B14 | 1 – If Tracker has been initialized following Auto Config 0 – If Tracker has not been initialized |
| B13 | 1 – If errors exist in the SYSTEM ERROR register (error(s) detected) 0 – If no errors exist in the register (no errors detected) |
| B12 | 1 – If Tracker is RUNNING 0 – IF Tracker is not RUNNING |
| B6-B11 | Not currently used by trakSTAR |
| 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 QUANTERNION outputs selected 1000 If POSITION/QANTERNION outputs selected |
| B0 | 0 – If POINT mode selected 1 If STREAM mode selected |

Note: In STREAM mode, you cannot 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 table:

Table 13 Filter On/Off Status Bit Assignments

| Bit # | Description |
|--------|---|
| B7-B3 | 0 |
| B2 | 1 – If the AC NARROW notch filter is ON (default) 0 – If the AC NARROW notch filter is OFF |
| B1 | 1 – If the AC WIDE notch filter is ON (default) 0 – If the AC WIDE notch filter is OFF |
| B0 | 1 – If the DC (Adaptive) filter is ON (default) 0 – If the DC filter is OFF |
| B6-B11 | Not currently used by trakSTAR |

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 hertz. If your application requires minimum transport delay between measurement of the sensor's position/orientation and the output of these measurements, you may want to evaluate the effect on your application with this filter shut off and the AC NARROW notch filter on. If you are running the Tracker synchronized to a CRT, you can usually shut this filter off without experiencing an increase in noise.

The **DC** filter refers to an adaptive, low pass filter applied to the sensor data to eliminate high frequency noise. When the DC filter is turned on, you can modify its noise/lag characteristics by changing ALPHA_MIN and Vm.

To CHANGE the FILTER ON/OFF STATUS send The Tracker two bytes of 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.

Table 14 DC Filter Constant Table Default Values (Std Range Transmitter)

| Range (Inches) | ALPHA MIN (Decimal) |
|-------------------|------------------------|
| 0-17 | 0.02 = 028F (HEX) |
| 17-22 | 0.02 |
| 22-27 | 0.02 |
| 27-34 | 0.02 |
| 34-42 | 0.02 |
| 42-54 | 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.

Table 15 DC Filter Table Default Values (Dipole Transmitters)

| Range (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.

Table 16 DC Filter Constant Table Default Values (Dipole Transmitters)

| Range (Inches) | ALPHA MAX ((Integer) |
|----------------|----------------------|
| 0-17 | 0.9 |
| 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.

Table 17 System Model Identification

| Device Description String | Device |
|---------------------------|---|
| “6DFOB “ | Stand alone (SRT) |
| “6DERC “ | Extended Range Controller |
| “6DBOF “ | MotionStar (old name) |
| “6DBB4” | 3DGuidance driveBAY/trakSTAR/driveBAY 2 / trakSTAR 2 |
| “6DMC180-4” | 3D Guidance (4 sensor 3DGuidance medSAFE™) |
| “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:

Table 18 List of Hemisphere Definitions

| Hemisphere | HEMI AXIS | | HEMI SIGN | |
|------------|-----------|-----|-----------|-----|
| | ASCII | HEX | ASCII | HEX |
| Forward | nul | 00 | nul | 00 |
| Aft (Rear) | nul | 00 | soh | 01 |
| Lower | ff | 0C | nul | 00 |
| Upper | ff | 0C | soh | 01 |
| Right | ack | 06 | nul | 00 |
| Left | ack | 06 | soh | 01 |

Notes:

- 1) Please note that these are the same PARAMETERdata values as are used by the HEMISPHERE command 'L' (4C Hex). To CHANGE the Hemisphere, send 2 PARAMETERdata bytes as described above.
- 2) This command operates in exactly the same way as the HEMISPHERE command. The command is now included in the CHANGE/EXAMINE command set in order to allow users to examine the values which were previously inaccessible.
- 3) The values can only be EXAMINED with this command if they were previously CHANGED by this command.

ANGLE ALIGN

When PARAMETERnumber = 23 during EXAMINE, The Tracker will return 3 words (6 bytes) of data corresponding to the Azimuth, Elevation, and Roll angles used in the ANGLE ALIGN command. This command differs from the ANGLE ALIGN command only in that it allows both reading and writing of the angles. See ANGLE ALIGN for a full explanation of its use.

To CHANGE the angles send 6 bytes of PARAMETERdata after the 2 command bytes.

REFERENCE FRAME

When PARAMETERnumber = 24 during EXAMINE, The Tracker will return 3 words (6 bytes) of data corresponding to the Azimuth, Elevation and Roll angles used in the REFERENCE FRAME command.

See REFERENCE FRAME2 command for an explanation.

To CHANGE the angles send 6 bytes of PARAMETERdata after the 2 command bytes.

TRACKER SERIAL NUMBER

When PARAMETERnumber = 25 during EXAMINE, The Tracker will return a 1 word (2 byte) value corresponding to the Serial Number of the Tracker Electronics Unit.

Note: This number cannot be changed.

SENSOR SERIAL NUMBER

When PARAMETERnumber = 26, during EXAMINE, the Tracker will return a 1 word (2 byte) value corresponding to the Serial Number of 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:

Table 19 System Status Bit Descriptions

| Bit # | Description |
|-------|--|
| B7 | 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. |
| B6 | 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). |
| B5 | If 1, then there is a valid sensor at this address. If 0, then no sensor at this address was found. |
| B4 | If 1, then the transmitter is an ERT/WRT. If 0, transmitter is standard range. |
| B3 | If 1, ERT/WRT #3 is present. If 0, not present. |
| B2 | If 1, ERT/WRT #2 is present. If 0, not present. |
| B1 | If 1, ERT/WRT #1 is present. If 0, not present. |
| B0 | 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**, 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

Table 20 AUTOCONFIG Bit Definitions

| Byte | Description | |
|------|---|------------|
| 1 | FBB Configuration Mode: 0 - STANDALONE 1 - ONE TRANSMITTER/MULTIPLE SENSORS | |
| 2,3 | FBB Devices: | |
| | Bit | Definition |
| | 15 | 0 |

| Byte | Description | |
|------|-------------|---|
| | 14 | <p>If 1, device at address 14 is running If 0, device at address 14 is not running</p> <p>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 LED is blinking green and it has not been AUTO-CONFIGed or it has been AUTO-CONFIGed and it is ASLEEP.</p> |
| | 13 | <p>If 1, device at address 13 is running If 0, device at address 13 is not running</p> |
| | 1 | <p>If 1, device at address 1 is running If 0, device at address 1 is not running</p> |

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. E.g., if the first byte returned is 1 and the second byte is 2, the firmware revision number is 1.2.

MDSP FIRMWARE REVISION

When PARAMETERnumber = 131 during EXAMINE, the system will return 2 bytes indicating the revision number of the MDSP firmware. E.g., if the first byte returned is 1 and the second byte is 2, the firmware revision number is 1.2.

NONDIPOLE POSERVER FIRMWARE REVISION

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

FIVEDOF FIRMWARE REVISION

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

SIXDOF FIRMWARE REVISION

When PARAMETERnumber = 136 during EXAMINE, the system will return 2 bytes indicating the revision number of the 6DOF firmware. E.g., if the first byte returned is 1 and the second byte is 2, the firmware revision number is 1.2.

DIPOLE POSERVER FIRMWARE REVISION

When PARAMETERnumber = 137 during EXAMINE, the system will return 2 bytes indicating the revision number of the Dipole POserver firmware. E.g., if the first byte returned is 1 and the second byte is 2, the firmware revision number is 1.2.

HEMISPHERE

| | ASCII | HEX | DECIMAL | BINARY |
|--------------|-------|-----|---------|----------|
| Command Byte | L | 4C | 76 | 01001100 |

| | | |
|--------------|-----------|-----------|
| Command Data | HEMI_AXIS | HEMI_SIGN |
|--------------|-----------|-----------|

The shape of the magnetic field transmitted by the Tracker is symmetrical about each of the axes of the transmitter. This symmetry leads to an ambiguity in determining the sensor's X, Y, Z position. The amplitudes will always be correct, but the signs (\pm) may all be wrong, depending upon the hemisphere of operation. In many applications, this will not be relevant, but if you desire an unambiguous measure of position, operation must be either confined to a defined hemisphere or your host computer must 'track' the location of the sensor.

There is no ambiguity in the sensor's orientation angles as output by the ANGLES command, or in the rotation matrix as output by the MATRIX command.

The HEMISPHERE command is used to tell the Tracker in which hemisphere, centered about the transmitter, the sensor will be operating. There are six hemispheres from which you may choose: the forward, aft (rear), upper, lower, left, and the right. If no HEMISPHERE command is issued, the forward is used by default.

The two Command Data bytes, sent immediately after the HEMISPHERE command, are to be selected from these:

| Hemisphere | HEMI_AXIS | | HEMI_SIGN | |
|------------|-----------|-----|-----------|-----|
| ASCII HEX | ASCII | HEX | ASCII | HEX |
| Forward | nul | 00 | nul | 00 |
| Aft (Rear) | nul | 00 | soh | 01 |
| Upper | ff | 0C | soh | 01 |
| Lower | ff | 0C | nul | 00 |
| Left | ack | 06 | soh | 01 |
| Right | ack | 06 | nul | 00 |

The ambiguity in position determination can be eliminated if your host computer's software continuously 'tracks' the sensor location. In order to implement tracking, you must understand the behavior of the signs (\pm) of the X, Y, and Z position outputs when the sensor crosses a hemisphere boundary. When you select a given hemisphere of operation, the sign on the position axes that defines the hemisphere direction is forced to be positive, even when the sensor moves into another hemisphere. For example, the power-up default hemisphere is the forward hemisphere. This forces X position outputs to always be positive. The signs on Y and Z will vary between plus and minus depending on where you are within this hemisphere. If you had selected the lower hemisphere, the sign of Z would always be positive and the signs on X and Y will vary between plus and minus. If you had selected the left hemisphere, the sign of Y would always be negative, etc.

Regarding the default forward hemisphere, if the sensor moved into the aft hemisphere, the signs on Y and Z would instantaneously change to opposite polarities while the sign on X remained positive. To 'track' the sensor, your host software, on detecting this sign change, would reverse the signs on The Tracker's X, Y, and Z outputs. In order to 'track' correctly: You must start 'tracking' in the selected hemisphere so that the signs on the outputs are initially correct, and you must guard against the case where the sensor legally crossed the $Y = 0$, $Z = 0$ axes simultaneously without having crossed the $X = 0$ axes into the other hemisphere.

MATRIX

| | ASCII | HEX | DECIMAL | BINARY |
|--------------|-------|-----|---------|----------|
| Command Byte | X | 58 | 88 | 01011000 |

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

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

| | | | |
|----|----------|----------|----------|
| . | M (1, 1) | M (1, 2) | M (1, 3) |
| .. | M (2, 1) | M (2, 2) | M (2, 3) |
| .. | M (3, 1) | M (3, 2) | M (3, 3) |
| . | | | |

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

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

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

| | | | |
|---|----------------------------|----------------------------|------------------|
| . | COS (E) *COS (A) | COS (E) *SIN (A) | -SIN (E) |
| . | -COS (R) *SIN (A) | COS (R) *COS (A) | |
| . | +SIN (R) *SIN (E) *COS (A) | +SIN (R) *SIN (E) *SIN (A) | SIN (R) *COS (E) |
| . | SIN (R) *SIN (A) | -SIN (R) *COS (A) | |
| . | +COS (R) *SIN (E) *COS (A) | +COS (R) *SIN (E) *SIN (A) | COS (R) *COS (E) |

3D GUIDANCE TRAKSTAR 2™ INSTALLATION AND OPERATION GUIDE

The output record is in the following format for the eighteen transmitted bytes:

| MSB | | | | LSB | | | | BYTE # | |
|-----|-----|-----|-----|-----|-----|-----|----|--------|---------------|
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | | |
| 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

WARNING: 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, i.e. 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 SENSOR |
|--------------|---|

Normally the position outputs from the **trakSTAR** represent the x, y, z position of the center of the sensor with respect to the origin of the Transmitter. The **OFFSET** command allows the user to specify a location that is offset from the center of the sensor. Figure 43 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 43, the x, y, z offset distances you supply with this command are measured in the reference frame attached to the sensor and are measured from the sensor center to the desired position (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 were 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:

$$\text{Xoffset} = 4915 = 5.4 * 32768 / 36.$$

$$\text{Yoffset} = 63625 = 65536 - 1911$$

$$\text{Zoffset} = 1183$$

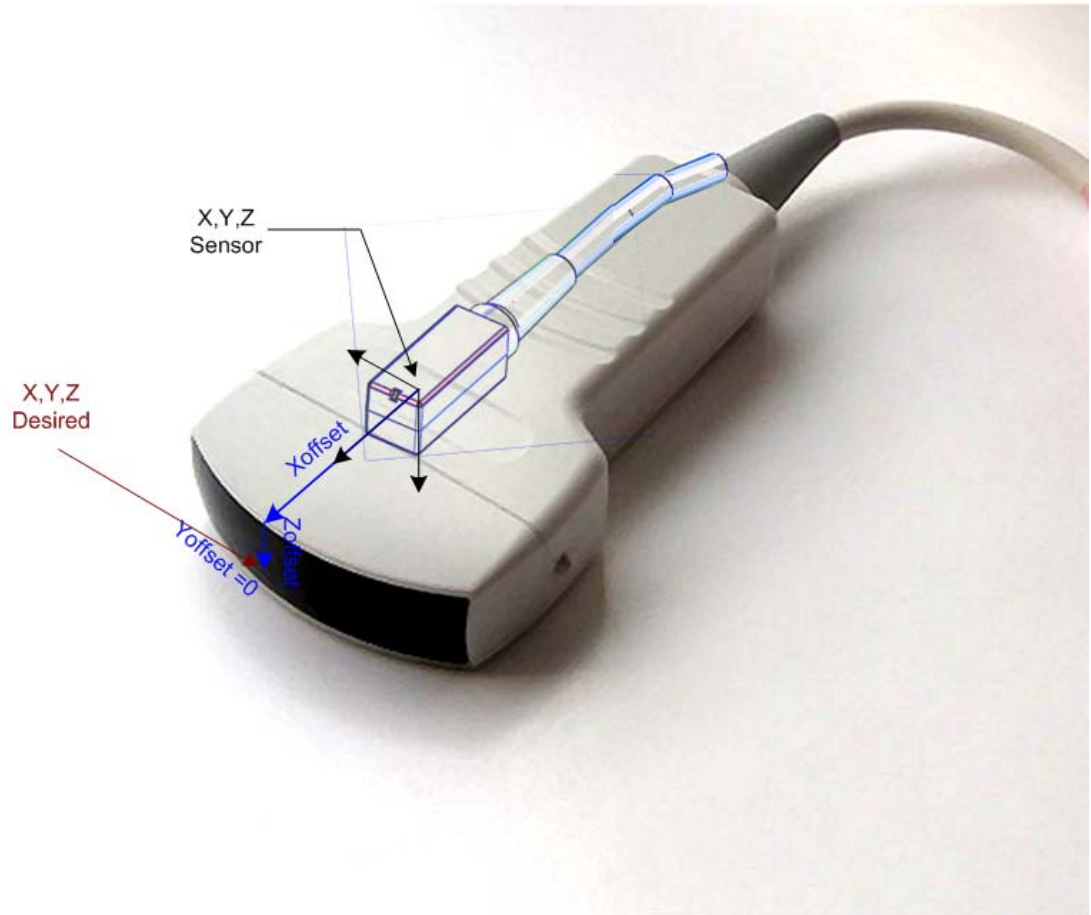


Figure 43 Using the OFFSET command

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 7 | 6 | 5 | 4 | 3 | 2 | 1 | LSB 0 | BYTE # |
|---|----------|-----|-----|-----|-----|-----|----|----------|-------------|
| 1 | X8 | X7 | X6 | X5 | X4 | X3 | X2 | | #1 LSbyte X |
| 0 | X15 | X14 | X13 | X12 | X11 | X10 | X9 | | #2 MSbyte X |
| 0 | Y8 | Y7 | Y6 | Y5 | Y4 | Y3 | Y2 | | #3 LSbyte Y |
| 0 | Y15 | Y14 | Y13 | Y12 | Y11 | Y10 | Y9 | | #4 MSbyte Y |
| 0 | Z8 | Z7 | Z6 | Z5 | Z4 | Z3 | Z2 | | #5 LSbyte Z |
| 0 | Z15 | Z14 | Z13 | Z12 | Z11 | Z10 | Z9 | | #6 MSbyte Z |

The X, Y, and Z values vary between the binary equivalent of \pm MAX inches. Where MAX = 36" or 72". The default positive X, Y, and Z directions are shown in Figure 26 [Default Reference Frame](#)

Scaling of each position coordinate is full scale = MAX inches. That is, +MAX = 7FFF Hex, 0 = 0 Hex, -MAX = 8000 Hex. Since the maximum range (Range = square root($X^2 + Y^2 + Z^2$)) from the Transmitter to the sensor is limited to MAX inches, only one of the X, Y, or Z coordinates may reach its full scale value. Once a full scale value is reached, the positional coordinates no longer reflect the correct position of the sensor.

To convert the numbers into inches first cast it into a signed integer. This will give you a number from +/- 32767. Second multiply by 36 or 72. Finally divide the number by 32768 to get the position in inches. The equation should look something like this:

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

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

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 | | |
|-----|-----|-----|-----|-----|-----|-----|----|-----|------|-------------|
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | | BYTE | # |
| 1 | X8 | X7 | X6 | X5 | X4 | X3 | X2 | | #1 | LSbyte X |
| 0 | X15 | X14 | X13 | X12 | X11 | X10 | X9 | | #2 | MSbyte X |
| 0 | Y8 | Y7 | Y6 | Y5 | Y4 | Y3 | Y2 | | #3 | LSbyte Y |
| 0 | Y15 | Y14 | Y13 | Y12 | Y11 | Y10 | Y9 | | #4 | MSbyte Y |
| 0 | Z8 | Z7 | Z6 | Z5 | Z4 | Z3 | Z2 | | #5 | LSbyte Z |
| 0 | Z15 | Z14 | Z13 | Z12 | Z11 | Z10 | Z9 | | #6 | MSbyte Z |
| 0 | Z8 | Z7 | Z6 | Z5 | Z4 | Z3 | Z2 | | #7 | LSbyte Zang |
| 0 | Z15 | Z14 | Z13 | Z12 | Z11 | Z10 | Z9 | | #8 | MSbyte Zang |
| 0 | Y8 | Y7 | Y6 | Y5 | Y4 | Y3 | Y2 | | #9 | LSbyte Yang |
| 0 | Y15 | Y14 | Y13 | Y12 | Y11 | Y10 | Y9 | | #10 | MSbyte Yang |
| 0 | X8 | X7 | X6 | X5 | X4 | X3 | X2 | | #11 | LSbyte Xang |
| 0 | X15 | X14 | X13 | X12 | X11 | X10 | X9 | | #12 | MSbyte Xang |

See POSITION mode and ANGLE mode for number ranges and scaling.

POSITION/MATRIX

| | ASCII | HEX | DECIMAL | BINARY |
|--------------|-------|-----|---------|----------|
| Command Byte | Z | 5A | 90 | 01011010 |

In the POSITION/MATRIX mode, the outputs from the POSITION and MATRIX modes are combined into one record containing the following twenty four bytes:

| MSB | | | | | | | | LSB | | | | 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 | 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, q0, q1, q2, and q3 that describe the orientation of the sensor with respect to the Transmitter. The output record is in the following format for the fourteen transmitted bytes:

| MSB | 7 | 6 | 5 | 4 | 3 | 2 | 1 | LSB | 0 | BYTE # | |
|-----|-----|-----|-----|-----|-----|-----|----|-----|---|--------|-----------|
| 1 | X8 | X7 | X6 | X5 | X4 | X3 | X2 | | | #1 | LSbyte X |
| 0 | X15 | X14 | X13 | X12 | X11 | X10 | X9 | | | #2 | MSbyte X |
| 0 | Y8 | Y7 | Y6 | Y5 | Y4 | Y3 | Y2 | | | #3 | LSbyte Y |
| 0 | Y15 | Y14 | Y13 | Y12 | Y11 | Y10 | Y9 | | | #4 | MSbyte Y |
| 0 | Z8 | Z7 | Z6 | Z5 | Z4 | Z3 | Z2 | | | #5 | LSbyte Z |
| 0 | Z15 | Z14 | Z13 | Z12 | Z11 | Z10 | Z9 | | | #6 | MSbyte Z |
| 0 | B8 | B7 | B6 | B5 | B4 | B3 | B2 | | | #7 | LSbyte q0 |
| 0 | B15 | B14 | B13 | B12 | B11 | B10 | B9 | | | #8 | MSbyte q0 |
| 0 | B8 | B7 | B6 | B5 | B4 | B3 | B2 | | | #9 | LSbyte q1 |
| 0 | B15 | B14 | B13 | B12 | B11 | B10 | B9 | | | #10 | MSbyte q1 |
| 0 | B8 | B7 | B6 | B5 | B4 | B3 | B2 | | | #11 | LSbyte q2 |
| 0 | B15 | B14 | B13 | B12 | B11 | B10 | B9 | | | #12 | MSbyte q2 |
| 0 | B8 | B7 | B6 | B5 | B4 | B3 | B2 | | | #13 | LSbyte q3 |
| 0 | B15 | B14 | B13 | B12 | B11 | B10 | B9 | | | #14 | MSbyte q3 |

See POSITION mode and QUATERNION mode for number ranges and scaling.

QUATERNION

| | ASCII | HEX | DECIMAL | BINARY |
|--------------|-------|-----|---------|----------|
| Command Byte | \ | 5C | 92 | 01011100 |

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

| | MSB | | | | | | | LSB | |
|---|-----|-----|-----|-----|-----|-----|----|-----|-----------------|
| | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | BYTE # |
| 1 | B8 | B7 | B6 | B5 | B4 | B3 | B2 | | #1 LSbyte q_0 |
| 0 | B15 | B14 | B13 | B12 | B11 | B10 | B9 | | #2 MSbyte q_0 |
| 0 | B8 | B7 | B6 | B5 | B4 | B3 | B2 | | #3 LSbyte q_1 |
| 0 | B15 | B14 | B13 | B12 | B11 | B10 | B9 | | #4 MSbyte q_1 |
| 0 | B8 | B7 | B6 | B5 | B4 | B3 | B2 | | #5 LSbyte q_2 |
| 0 | B15 | B14 | B13 | B12 | B11 | B10 | B9 | | #6 MSbyte q_2 |
| 0 | B8 | B7 | B6 | B5 | B4 | B3 | B2 | | #7 LSbyte q_3 |
| 0 | B15 | B14 | B13 | B12 | B11 | B10 | B9 | | #8 MSbyte q_3 |

Scaling of the quaternions is full scale = +.99996 = 7FFF Hex, 0 = 0 Hex, and
-1.0 = 8000 Hex.

REFERENCE FRAME

| | ASCII | HEX | DECIMAL | BINARY |
|--------------|-------|-----|---------|----------|
| Command Byte | r | 72 | 114 | 01110010 |

| | |
|--------------|---------|
| Command Data | A, E, R |
|--------------|---------|

By default, The Tracker's reference frame is defined by the Transmitter's physical X, Y, and Z axes.(see [Figure 26](#)) In some applications, it may be desirable to have the orientation measured with respect to another reference frame. The REFERENCE FRAME command permits you to define a new reference frame by inputting the angles required to align the physical axes of the Transmitter to the X, Y, and Z axes of the new reference frame. The alignment angles are defined as rotations about the Z, Y, and X axes of the Transmitter. These angles are called the, Azimuth, Elevation, and Roll angles.

The command sequence consists of a Command Byte and 6 Command Data bytes. The Command Data consists of the alignment angles Azimuth (A), Elevation (E), and Roll (R).

If you immediately follow the REFERENCE FRAME command with a POINT or STREAM mode data request you may not see the effect of this command in the data returned. It may take one measurement period before you see the effect of the command.

The Command Byte and Command Data must be transmitted to The Tracker in the following seven-byte format:

| MSB | | | | LSB | | | | BYTE # |
|-----|-----|-----|-----|-----|-----|----|----|-----------------|
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
| 0 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | #1 Command Byte |
| B7 | B6 | B5 | B4 | B3 | B2 | B1 | B0 | #2 LSbyte A |
| B15 | B14 | B13 | B12 | B11 | B10 | B9 | B8 | #3 MSbyte A |
| B7 | B6 | B5 | B4 | B3 | B2 | B1 | B0 | #4 LSbyte E |
| B15 | B14 | B13 | B12 | B11 | B10 | B9 | B8 | #5 MSbyte E |
| B7 | B6 | B5 | B4 | B3 | B2 | B1 | B0 | #6 LSbyte R |
| B15 | B14 | B13 | B12 | B11 | B10 | B9 | B8 | #7 MSbyte R |

See the ANGLES command for the format and scaling of the angle values sent.

REPORT RATE



Note: For alternate Report Rate settings, use the **CHANGE/ EXAMINE** command for this function. See [REPORT RATE](#) PARAMETERnumber=29.

| Measurement Rate Divisor Command | ASCII | HEX | DECIMAL | BINARY |
|--|-------|-----|---------|----------|
| 1 | Q | 51 | 81 | 01010001 |
| 2 | R | 52 | 82 | 01010010 |
| 8 | S | 53 | 83 | 01010011 |
| 32 | T | 54 | 84 | 01010100 |

If you do not want a Tracker data record output to your host computer every 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 (SENSOR) ADDR | ≡ | F0 | 240 | 11110000 | + FBB |

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 3DGuidance systems, which support multiple sensors. The pass through command permits selection and configuration of sensors beyond the first sensor.



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

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 commands, 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 |

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



Tip: To maximize the life of your system, issue the SLEEP command when you are not using the tracker, or configure the Sleep on Reset setting in Default Configuration (see [Table 1](#)).

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.

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.

- Examine Parameter 10 – retrieves single byte error code and removes it from the SYSTEM ERROR buffer.
- 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.

Table 21 Error Notification Options

| Error Notification | Description |
|---------------------|--|
| Error Flag | <p>Monitor the ERROR bit (B13) in the two byte BIRD STATUS register.</p> <ol style="list-style-type: none"> 1. Send EXAMINE VALUE command with PARAMETERnumber = 0 <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> <ol style="list-style-type: none"> 2. Send the EXAMINE VALUE command with PARAMETERnumber = 10 <p>This returns the earliest Error sent to the buffer and clears it.</p> <div style="border: 1px solid black; padding: 5px;"> <p>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.</p> </div> |
| SYSTEM ERROR | <p>Alternatively, the user can query the SYSTEM ERROR buffer directly.</p> <ol style="list-style-type: none"> 1. Send the EXAMINE VALUE command with PARAMETERnumber = 10 <p>This returns the earliest Error sent to the buffer. and clears it.</p> <ol style="list-style-type: none"> 2. Additional queries will return and clear the next Error code in the buffer, until the buffer. is empty. When the buffer is empty, the query will return '0'. <div style="border: 1px solid black; padding: 5px;"> <p>Note: If the query returns '45', then the 16-byte(16 error codes) buffer has overflowed, and the newest codes generated will be lost.</p> </div> |

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 |

| Code | Error Description |
|------|--|
| | 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. |

Troubleshooting, Maintenance, and Repair

*Taking care of your **trakSTAR 2** is easy. For best results over time, please treat all tracker components like delicate scientific instruments. Sensors, the transmitter, and their cables are typically handled most and are subject to the most wear and tear. Handle them with care and they will operate as specified for many years to come.*

User Maintenance

trakSTAR 2 requires minimal maintenance. Here are a few things that you can do to maximize good performance:

Maintenance Prior to Each Use

1. Periodically check the transmitter and sensor cables for nicks and cuts in the insulation.
2. Inspect both the component connectors and receptacles for bent or damaged pins.
3. Inspect the transmitter for cracks and exterior damage. If transmitter is cracked or interior of the transmitter is exposed in some way, the component should be replaced after proper disposal.

Periodic Maintenance (As needed)

1. Inspect serial port connections. Tighten as necessary to maintain positive contact.
2. Check that transmitter, sensor, and electronics unit mounting provisions are secure, and are as recommended in the [Mounting the Hardware](#) section of this Guide.

Proper Handling of Sensor and Cable

Since sensors and cables move through space and are often attached to flexible tools and instruments, they are subject to frequent and continuing stress. Here are a few things you can do to minimize problems:

- Don't over flex or twist the sensor cable.
- Prevent any part of the sensor from being crushed. The connectors can become warped if stepped on; the internal wires in the sensor cable may break or become weakened if pinched; and the sensor head may be damaged if trapped under something heavy.
- Don't drop or whack the sensor head against a hard surface. This will cause accuracy errors.
- If the sensor head is mounted on a tool, implement a strain relief where the sensor cable exits the tool to distribute forces over a region of the cable.

Cleaning and Disinfecting

To clean the equipment (electronics unit, transmitter, sensor, and cables) wipe with a cloth dampened with a cleaning solution such as mild soap and water, isopropyl alcohol or a similar solution according to your organization's standards. After each use, thoroughly clean equipment. In environments where the equipment may come in contact with biological fluid or tissue, be sure to follow your organization's procedures for proper cleaning and disinfection. The electronics unit, transmitters and sensors cannot be subject to autoclaving or gamma radiation. Sensors will withstand ETO. Do not immerse the electronics unit, transmitter, sensor, or cables in liquids, as they are not waterproof.



Note: The electronics unit, transmitters and sensors cannot be subject to autoclaving or gamma radiation. Sensors will withstand ETO or Sterrad processing. Do not immerse the electronics unit, transmitter, sensor, or cables in liquids, as they are not waterproof.

Firmware Updates

As new features or updates become available for **trakSTAR 2**, you may find it necessary to update the firmware stored in the Electronic Unit's memory. The driveBAY2 Utility included on your CD-ROM allows you to do this without opening the Electronics Unit or returning it to Ascension. See [Appendix II, driveBAY 2 Utility](#).

Disposal

The European Union has issued a directive, known as the WEEE (Waste Electrical and Electronic Equipment) Directive, to protect the quality of our environment by reducing the amount of electrical equipment waste buried in landfills. WEEE focuses on the recycling and reuse of “equipment that depends on an electronic current or an electromagnetic field to operate and as equipment for the generation, transfer and measurement of such currents and fields.” Although none of the **trakSTAR 2**’s components are hazardous materials, proper disposal is important, especially, in the European Union where these components cannot be consigned to a landfill. Wherever available, tracker components should be brought to centralized recycling and collection points. Please contact Ascension Technical Support for further instructions on the correct disposal procedures in your country. If you have biologically contaminated components, please refer to your organization’s standard operating procedures for proper disposal.

Troubleshooting

Most installation and tracking problems are easy to fix. Consult the troubleshooting table for common problems and their solutions. If you continue to experience problems, contact Ascension for technical support.

Table 22 Troubleshooting Symptoms, Possible Causes, and Solutions

| Symptom | Possible Causes | Solution |
|---|---|---|
| No front panel LED illumination | No Power Fuse(s) blown | Check AC connections to power supply Reset hardware by cycling AC power Check fuse drawer for blown fuse. Replace with same rating/type (see Fuses). |
| Front panel LED blinking red/yellow. | No valid transmitter attached. | Attach transmitter Reseat transmitter connector See Table 23 for LED state definitions |
| Front panel LED slow blinking green/orange. | Electronics not yet 'programmed' for the attached Transmitter | Let the unit continue reading and programming the calibration information from the transmitter. When the programming is complete, the LED will switch to slow blink green. |
| Demo Utility doesn't run | No serial communication Software Installation unsuccessful | Check the suggestions outlined in 'Not able to communicate' below Re-initialize the HOST PC and run the installation again |
| Power-up defaults did not change after configuring with the utility | Configuration download interrupted Tracker did not reset(restart) after burning the Flash settings | Re-start the trakSTAR 2 and the utility and set the defaults again. Be sure to click APPLY to send the settings to the Electronics Cycle power to the trakSTAR 2 , and re-check the settings |
| Not able to communicate with the system using RS-232 | Baud rate incorrect COM port incorrect RS232 cable disconnected System in RESET | Only supported Baud Rate for the trakSTAR 2 system is 115200. Specify to your HOST PC the COM port that the tracker is connected to. Check that the serial connections at both ends of the cable are fully seated. Check that host isn't controlling CTS/RTS. If enabled, and you are running Windows, either disable it or ensure that your application takes control of the COM port. |

| Symptom | Possible Causes | Solution |
|---|---|---|
| Not able to communicate with the system using USB | Re-initialize USB Driver not installed | Unplug and replug USB cable on trakSTAR 2 . Cycle power on trakSTAR 2 Check installation/status of trakSTAR 2 USB driver in Windows Device Manager. Re-install if necessary |
| No Data | Sensor is saturated Transmitter is OFF or disconnected Component connections faulty | Check the error codes for a sensor a saturation condition. Move the sensor farther away from the Transmitter. Check status of the Transmitter. Turn ON using SELECT_TRANSMITTER parameter or RUN command (ATCRS232 protocol). Check that Sensor/Transmitter connectors are correctly installed. Inspect pins for wear or damage. Contact Ascension for assistance |
| Data is too noisy | Filters OFF Low Signal External noise in environment Changing Hemisphere Line frequency value set incorrectly. | Check FILTER STATUS for present state of filter configuration. Decrease distance from sensor to Transmitter. Be sure that the sensor is not located near the Tracker's power supply or other electronic devices or cables. See section on Reducing Noise in Chapter 3 If the signs of the X, Y or Z position outputs suddenly change you may have crossed a hemisphere boundary. Use the HEMISPHERE command to rectify. Determine correct frequency (50 Hz in Europe, 60 Hz in North America) and set to correct value. |
| Poor accuracy | Metal in tracking environment. Damaged equipment. Sensor or transmitter connector not properly inserted. A software application error. | Check all around the transmitter to the furthest distance from the center of the transmitter to the maximum distance the sensor is used. Move or replace metal, or reposition trakSTAR 2 system. Check for damage. Correct by unplugging and plugging the components back into the board. Verify formulas and scale factors. |

Error Codes

Chapter 5 : 3DGuidance API Reference provides an explanation and listing of all [ERRORCODE](#) via the USB interface.

Chapter 6: Ascension RS232 Interface Reference provides a complete [Error Code Listing](#) for the RS232 protocol.

LED Definitions

Table 23 LED Definitions

| LED State | Description of Board Status | Possible Error Conditions resulting in LED state: |
|--|--|---|
| OFF | Power applied to board. | PLD un-programmed PLD error LED faulty Board error Board un-powered. |
| SOLID RED | PLD loaded and configured. | Flash memory un-programmed POSERVER DSP error Board error |
| SOLID ORANGE | POSERVER DSP - boot loader successfully loaded and running. All preliminary tests completed. | Multi-Boot Loader test failed POSERVER main application not found Code corrupt? DSP error Board error |
| SLOW BLINKING RED (< 1Hz) | POSERVER DSP - application loaded, running and Command Handler executing | MDSP application not found in Flash Communications failure Failed starting MDSP |
| FAST BLINKING RED (> 2Hz) | MDSP DSP - code downloaded by POSERVER | MDSP failed to respond to messages MDSP code corrupt MDSP error Board error |
| SLOW BLINKING ORANGE (< 1Hz) | POSERVER successfully communicated with MDSP | System EEPROM error PLD version error MDSP fails to respond after tests MDSP test failed |

| LED State | Description of Board Status | Possible Error Conditions resulting in LED state: |
|--|--|---|
| FAST BLINKING ORANGE (> 2Hz) | All of POSERVER initialization and power on test is complete | Transmitter not present Transmitter EEPROM error Transmitter hardware failure |
| SLOW BLINKING GREEN/ORANGE (< 1Hz) | Unit is re-programming internal memory for the attached Non-Dipole transmitter. | Normal – no user action necessary. When a NonDipole (Flat or MAGnet) transmitter and electronics unit are ‘paired’ together for the first time, the electronics unit must update internal memory with the new transmitter’s calibration information. When the unit completes the transfer of information from the transmitter’s memory (up to 2 minutes), it will turn the LED to the Ready state (slow blinking or solid green). Note that this re-programming occurs whenever the non-dipole transmitter has changed from that which was previously connected. If on power-up the unit detects that the same transmitter is still attached, no re-programming is necessary and the standard boot time (~12sec) will be observed. |
| SLOW BLINKING GREEN (< 1Hz) | System is fully functional. The transmitter is <u>not</u> being energized during acquisition. (ASLEEP) | System ASLEEP – normal If auto-config issued then abnormal |
| SOLID GREEN | System is fully functional. The transmitter <u>is</u> being energized during the acquisition. (AWAKE) | System AWAKE – normal If SLEEP command issued then abnormal. |
| FAST BLINKING RED/ORANGE (> 2Hz) | Run-time Diagnostic Error | One of the run-time diagnostics has failed. Primary suspect is a removed transmitter. (This shows up as a TX Temp Sense error.) Secondary suspects might be TX Temp. Sense or EU Temp Sense (over-heating problem) |

Contacting Ascension Technology Corporation for Repair

Fuses

A fuse drawer is located on the rear panel, next to the AC power inlet. Both power entry conductors are protected by size 5 x 20 mm fuses:

T1AL250V : 1.0 A ; 250V; T (Time Lag /SLO-BLO)

Other

There are no user serviceable parts inside the **trakSTAR 2** Electronics, Transmitters, or Sensors.

In addition to this Guide, we can provide personal support by contacting us at any of the following ways:

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

E-mail: support@ascension-tech.com

Telephone: Call (802) 893-6657 9 a.m.-- 5 p.m. U.S. Eastern Standard Time, Monday through Friday.

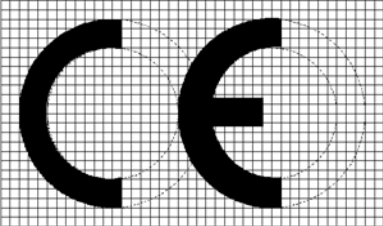
Fax: (802) 893-6659

Warranty

Ascension warrants that its products are free from defects in material and workmanship for a period of one (1) year from date of delivery, providing the products are not subject to misuse, neglect, accident, incorrect installation, or improper care. If any Ascension products fail due to no fault of the buyer, Ascension will (at its option) either repair the defective product and restore it to normal operation without charge for parts and labor or provide a replacement in exchange for the defective product. Repair work shall be warranted for the remainder of the unexpired warranty period or for a period of 60 days, whichever is longer. This warranty is the exclusive warranty given in lieu of any other express or implied warranty. Ascension disclaims any implied warranties of merchantability and fitness to a particular purpose. Warranties are voided if the buyer utilizes a power supply that does not strictly adhere to Ascension's electrical power requirements, changes the configuration of a tracker, (such as, adding extensions to cables or modifying boards), or mishandles Sensors or cables. To avoid warranty issues, customers should carefully adhere to all product advisories. Transmitter, Sensors, cables and connectors are sensitive electronic components and should be treated with care. Do not drop, pull, twist, or mishandle cables.

Regulatory Information and Specifications

Mid-Range, Short Range, Flat, and MAGnet Transmitter Configurations:

| | |
|--|---|
|  | <p>In accordance with EN60601-1 (Medical electrical equipment – general requirements for safety), this equipment is classified as follows:</p> <p>Class I</p> <ul style="list-style-type: none"> – Type B Applied Part – Not AP/APG |
|--|---|

Class I: Non-invasive electric/electronic equipment without a monitoring function, which has a reliable ground and thus provides the type of protection against electric shock as defined by EN60601-1.



Type B Applied Part: An applied part (sensors) complying with the specified requirements of EN 60601-1 to provide protection against electric shock, particularly regarding allowable leakage current. Type B specifies the degree of electric shock protection provided by the unit.

Not AP/APG means unsuitable for use in the presence of flammable gases.

Modification or use of the equipment in any way that is not specified by Ascension Technology Corporation may impair the protection and accuracy provided by the equipment.



The lightning flash arrow symbol within an equilateral triangle is intended to alert the user to the presence of uninsulated dangerous voltage within the product's enclosure. That voltage may constitute a risk of electric shock to persons.



The exclamation point within an equilateral triangle is intended to alert the user to the presence of important operating and maintenance (servicing) instructions in the appliance literature.



Waste Electrical and Electronic Equipment compatible. European Union – do not place in landfill.



Warning: This tracker does not have approval from the FDA for patient contact applications

EC Declaration of Conformity

EC Declaration of Conformity

Issued by

Ascension technology corporation
PO Box 527
Burlington, VT 05402 USA
802-893-6657

| | |
|-------------------------------|---|
| Equipment Description: | 3D Guidance trakSTAR 2™ Tracking System (w/Mid-Range, Short-Range, Flat, and MAGnet Transmitters) 100-240VAC 50/60Hz; 70VA |
| Applicable Directive: | 93/42/EEC, Medical Devices |
| Applicable Standards: | IEC 60601-1 Ed. 3 2005 Medical Electrical Equipment: Part 1: General Requirements for Safety IEC 60601-1-2 Ed. 3 2007 Medical Electrical Equipment: Part 1: General Requirements for Safety –2. Collateral Standard: Electromagnetic Compatibility- Requirements and Test |

FCC Compliance Statement

Radio and television interference

Warning: Changes or modifications to this unit not expressly approved by the party responsible for compliance could void the user's authority to operate the equipment.

Note: This equipment has been tested and found to comply with the limits for a Class B digital device, pursuant to Part 15 of the FCC rules. These limits are designed to provide reasonable protection against harmful interference in a residential installation. This equipment generates, uses, and can radiate radio frequency energy and, if not installed and used in accordance with the instructions, may cause harmful interference to radio communications. However, there is no guarantee that interference will not occur in a particular installation. If this equipment does cause harmful interference to radio or television reception, which can be determined by turning the equipment off and on, the user is encouraged to try and correct the interference by one or more of the following measures:

- Reorient or relocate the receiving antenna.
- Increase the separation between the equipment and receiver.
- Connect the equipment into an outlet on a circuit different from that to which the receiver is connected.
- Consult the dealer or an experienced radio/TV technician for help.

Declaration of Conformity

Model Number: 3D Guidance **trakSTAR 2™**
Trade Name: Ascension
Responsible Party: Ascension Technology Corporation
Address: P.O. Box 527
 Burlington, Vermont 05402
Telephone Number: (802) 893-6657

This device complies with Part 15 of the FCC rules.

Operation is subject to the following two conditions:

1. This device may not cause harmful interference, and
2. This device must accept any interference received, including interference that may cause undesired operation.

Wide-Range Transmitter Configurations:

FCC Regulations

Warning: Changes or modifications to this unit not expressly approved by the party responsible for compliance could void the user's authority to operate the equipment.

NOTE: This equipment has been tested and found to comply with the limits for a Class A digital device, pursuant to Part 15 of the FCC Rules. These limits are designed to provide reasonable protection against harmful interference when the equipment is operated in a commercial environment. This equipment generates, uses, and can radiate radio frequency energy and, if not installed and used in accordance with the instruction manual, may cause harmful interference to radio communications. Operation of this equipment in a residential area is likely to cause harmful interference in which case the user will be required to correct the interference at the user's expense.

EC Declaration of Conformity

EC Declaration of Conformity

Issued by

ASCENSION TECHNOLOGY CORPORATION
PO Box 527
Burlington, VT 05402 USA
802-893-6657

| | |
|-------------------------------|---|
| Equipment Description: | 3D Guidance trakSTAR 2™ Tracking System (w/Wide-Range Transmitters) 100-240VAC 50/60Hz; 70VA |
| Applicable directive: | 73/23/EEC, Low Voltage Directive 89/336/EEC, EMC Directive |
| Applicable standards: | EN 61010-1: 1995 Safety Requirements for Electrical Equipment for Measurement, Control and Laboratory (MCL) Use-Part 1. General Requirements EN 61326-1: 1998 Electrical Equipment for Measurement, Control and Laboratory Use - Electromagnetic Compatibility Requirements |

Product Specifications

Performance

| | |
|---------------------------------|--|
| Degrees of freedom: | Six (position and orientation) |
| Translation range: | |
| Short Range Transmitter: | ±45 cm in any direction |
| Mid Range Transmitter: | ±76 cm in any direction |
| Wide Range Transmitter: | ±2.1m in any direction |
| Angular range: | All attitude: ±180 deg azimuth and roll, ±90 deg elevation |
| Static accuracy: | <p>Mid-Range, Short-Range, Flat, and MAGnet Transmitters: (see note below) 1.4 mm RMS position 0.5 degree RMS orientation</p> <p>Wide Range Transmitters: 3.8mm RMS position 0.5 degree RMS orientation</p> |
| Update rate: | <p>Short, mid, and wide range transmitters up to 600 updates/second.</p> <p>Flat – 378 updates/sec (System Measurement Rate: 42Hz)</p> <p>MAGnet – 504 updates/sec (System Measurement Rate: 42Hz)</p> |
| Position: | <p>Because of symmetries in the transmitted field for short and mid-range transmitters, operation of the sensor with these transmitters shall be confined to one of six hemispheres of operation. Operation within a given hemisphere requires that position sign of the axis of rotation for that hemisphere does not change sign. In order to meet accuracy specifications, the system must operate in the forward (positive X) hemisphere.</p> |
| Outputs: | X, Y, Z positional coordinates, orientation angles, orientation matrix and quaternion. |
| Interface | USB / RS232 |

Physical

| | |
|--------------|---|
| Size: | <p>Mid-Range Transmitter: 3.75" (9.6cm) cube with 10' (3.05m) cable</p> <p>Short-Range Transmitter: 2.09"(5.3cm) x 2.09"(5.3cm) x 2.71"(6.9cm)</p> <p>Wide Range Transmitter: 12" (30.5cm) cube with 20' (6.15m) cable</p> <p>Flat Transmitter:</p> <p>MAGnet Transmitter:</p> <p>Model 800 Sensor: Sensor: 8 mm x 8 mm x 20 mm (0.31 inch x 0.31 inch x 0.78 inch) Cable O.D.: 3.8mm Cable Length Options: 3.3m (10.8ft) 9.1 m (30 ft) cable</p> <p>Model 180 Sensor: Sensor Housing: Outside Diameter: 2 mm (0.07 inch) Length: 9.7 mm (0.38 inch) Sensor Cable: 3.3 m (10.8 ft)</p> <p>Model 130 Sensor: Sensor Housing: Outside Diameter: 1.5 mm (0.05 inch) Length: 7.7 mm (0.30 inch) Sensor Cable: 3.3 m (10.8 ft)</p> |
|--------------|---|

3D GUIDANCE TRAKSTAR 2™ INSTALLATION AND OPERATION GUIDE

| | |
|---------------------------------|--|
| | <p>Model 90 Sensor:</p> <p>Sensor Housing:</p> <p>Outside Diameter: 0.9 mm (0.035 inch)</p> <p>Length: 8.8 mm (0.35 inch)</p> <p>Sensor Cable: 3.3 m (10.8 ft)</p> <p>Model 55 Sensor:</p> <p>Sensor Housing:</p> <p>Outside Diameter: 0.55 mm –0/+0.01mm (inch)</p> <p>Length: X.X mm (0.XX inch)</p> <p>Sensor Cable: X.X m (XX.X ft)</p> <p>Electronics Unit</p> <p>Dimensions (L x W x H): 18.4 cm x 29.2cm x 6.4 cm</p> <p>Weight: 1.28 Kg</p> |
| Power: | The unit's internal supplies will operate from 100 to 240V, at 50/60 Hz. Power consumption is 70 VA. |
| Operating temperature: | 59°F to 95°F (15°C to 40°C), 95% non-condensing humidity. |
| Warm up: | System shall meet accuracy specifications after 5 mins |
| Note on static accuracy: | Accuracy is defined as the RMS position error of the magnetic center of a single sensor with respect to the magnetic center of a single transmitter over the Performance Motion Box . Accuracy will be degraded if there are interfering electromagnetic noise sources or metal in the operating environment. |

Appendix I: winBIRD

Running the RS232 Demo Utility

The winBIRD demo utility is available for download from the Ascension ftp site.

<ftp://ftp.ascension-tech.com/PRODUCTS/UTILITIES/>

Use the demo utility to become familiar with the sensor's motion region, and the tracker's capabilities. If the utility does not run, or the 3DGuidance™ does not operate as described, please consult the troubleshooting table in Chapter 7 of this manual for assistance.

1. Run the winBIRD Installer prior to running the winBIRD demo program. Browse to the \3D GuidanceXXXX\WinBIRD subdirectory (either on the CD-ROM or in your \Program Files\Ascension\3D GuidanceXXXX\ subdirectory), and double-click winBIRD.msi.

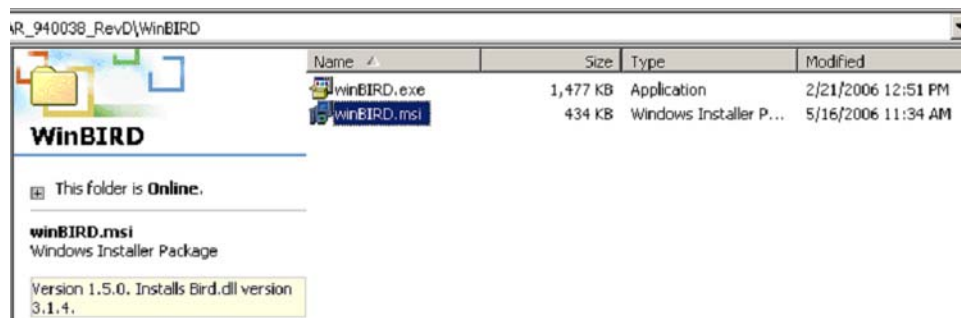


Figure 44 Accessing the winBIRD Installer

2. Follow the setup wizard prompts to install winBIRD.
3. Start the demo utility by selecting winBIRD from the Ascension Technology program group in the Windows® Start menu.

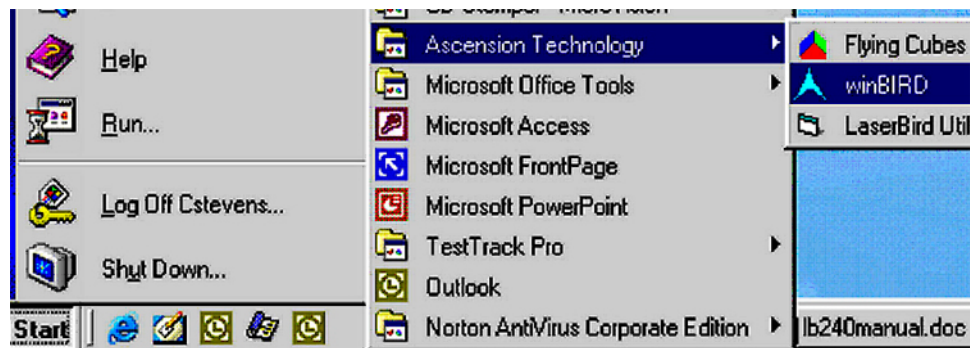


Figure 45 Opening the winBIRD Demo Program

4. winBIRD opens with the top of the screen containing the menu bar and toolbars, providing links to main winBIRD functions. Most are initially disabled with the menus and toolbars becoming available as procedures are fulfilled.
5. Click **Setup** on the menu bar, and select **RS232**.



Figure 46 Setting up for RS232

6. . Set the **Baud Rate** for the COM port to 115200.



Figure 47 Settingthe RS232 Baud Rate

7. Enter the total number of sensors in the **Number of Devices** field and click **OK**. (See Note if only one sensor is used.)



Note: 'Not Standalone' or 'Group' mode' is used for a 'group' of birds or multi-sensor systems

8. In Device 1 message box, enter the Com Port for the 3DGuidance™ and ensure the Direct radio button is clicked (for enabling Device 1).

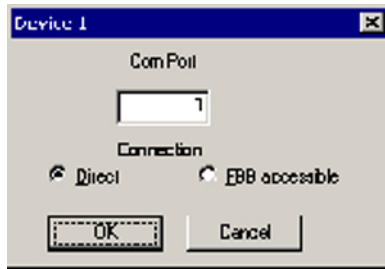


Figure 48 Setting Sensor 1 (Device 1) Parameters

9. For the remaining devices (sensors), enable the 'FBB accessible' radio button and click **OK**.

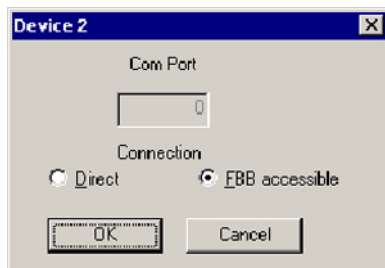


Figure 49 Setting Remaining Sensors Parameters

10. The main window will display “**Waking up bird**” indicating the utility is establishing communication with the system. “**Setup complete**” displays when communication is established and several of the icons on the toolbar will be enabled.
11. Select '**Take Data**' from the menu bar and choose the option you would like for displaying the data.



Figure 50 Selecting Animate from Take Data Drop-Down Menu

Select **Animate** to display a set of tri-color axes. (one axis per sensor) This axis graphically represents the sensor's translations and rotations in the motion region.

If **Stream** is selected, the display will continuously update two rows of data for each sensor. The three values in the first row of each sensor block represent sensor position, in inches, relative to the Transmitter. The three values in the second row give sensor orientation in degrees.

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

Appendix II: driveBAY 2 Utility

Running the driveBAY 2 Utility

Setup

Before changing settings or beginning an upgrade, setup the tracker with either the RS232 or USB interface.



Note: This Utility can be run with either the driveBAY 2 or **trakSTAR 2** systems.

1. Connect the RS232 or USB cable and the Power cable to the rear panel of the tracker.



Note: The Utility can be run using either the RS232 (COM port capable of 115.2Kbaud) or USB interfaces.

2. Connect the other end of the RS232 or USB cable to an open COM port or USB port on the host PC (PC that will run the utility)
3. Power on the Tracker.

Run the Utility

1. Start the utility by running the 'trakSTAR 2 Utility.exe' file located on the CD-ROM. This opens the interface configuration window of the Utility.

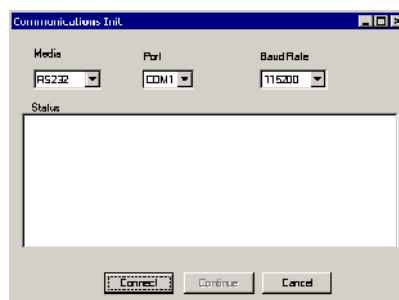


Figure 51 driveBAY 2 Utility Communication Initialize Window

2. If using the RS232 interface, select the COM port from the pull down menu, and click Connect.



Note: Current operation supports 115200 baud only.

3. If using the USB interface, select **USB** from the **Media** pull down menu and click **Connect** to establish communication with the tracker and initiate reading the current configuration. Progression of the reading is shown in the Status window.
4. When the reading has completed, select **Continue** to open the **Utility** default **Settings** tab.

The screenshot shows the 'driveBAY2 Utility 35.0.22.1' window with the 'Dipole Settings' tab selected. The window contains several configuration sections:

- System Settings:** Includes fields for IP Address (92.168.200.51), Mask (255.255.0.0), Port (6000), Baud Rate (115200), Sync mode (Internal), Rate (Hz) (240), Line Freq (60), Report (24), Scale (72), MultiUnit ID (0), Full POST on reset (unchecked), Sleep on reset (checked), and Enable LED (checked). Sensors IDs are listed as 4, 3, 2, 1.
- Transmitter Settings:** Includes 'Enable Reference Frame' (unchecked) and 'Reference Frame(degrees)' with X (0), Y (90), and Z (90) values.
- Sensor Settings:** A tabbed interface for Sensor 1 through Sensor 4. Sensor 1 is selected, showing 'Hemisphere' (FRONT), 'Data Format' (POS/ANG), 'Sensor Offsets(inches)' (X: 0, Y: 0, Z: 0), 'Angle Align(degrees)' (Az: 0, El: 0, Rl: 0), and a table of filter settings.

The filter settings table for Sensor 1 is as follows:

| | Alpha | 0.02 | 0.02 | 0.02 | 0.02 | 0.02 | 0.02 | 0.02 | 0.02 | 0.02 | 0.02 |
|-----------------|-------|------|------|------|------|------|------|------|------|------|------|
| AC Wide ON | Alpha | 0.9 | 0.9 | 0.9 | 0.9 | 0.9 | 0.9 | 0.9 | 0.9 | 0.9 | 0.9 |
| AC Narrow ON | Alpha | 2 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 |
| Adaptive filter | VM | | | | | | | | | | |

At the bottom, there are buttons for 'Set To Defaults', 'Read From Tracker', 'Apply to Tracker', 'Save to File...', and 'Read from File...'.

Figure 52 driveBAY 2 Utility Window (Dipole Default Settings Tab Displayed)

5. To upgrade Flash Sectors, click the 'Flash Maintenance' tab to show the contents of the flash memory device.

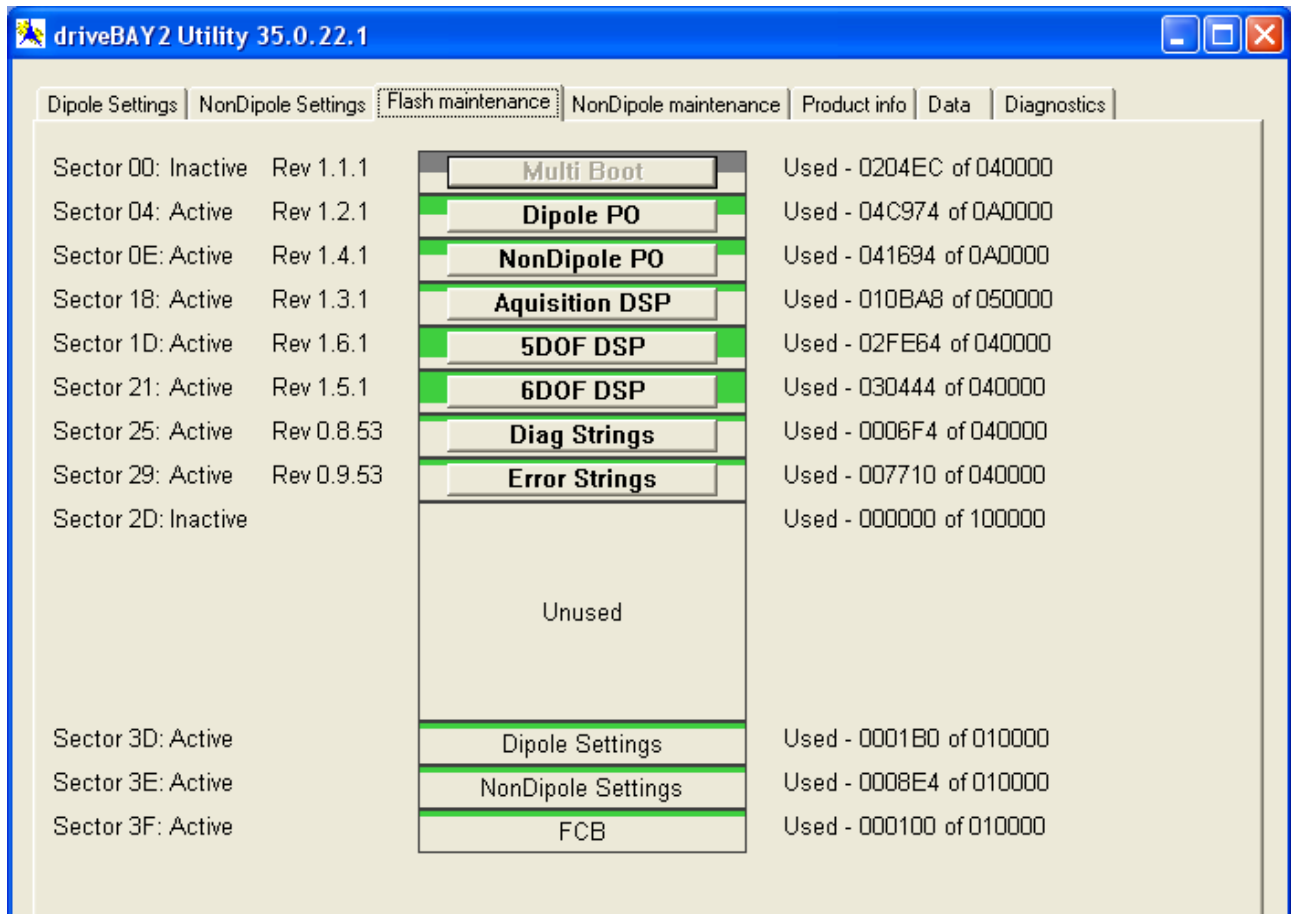


Figure 53 driveBAY 2 Utility Flash Maintenance

- Click the flash sector to be upgraded. Select the new loader file and click 'Open'. This will begin the upgrade of the flash sector. A progress bar will indicate status. When the upgrade of the sector is complete, the new contents of the Flash will be displayed in the 'Rev' field next to each sector
- Close the Utility ('X' in title bar) and cycle the power on the tracker.

Appendix III: Application Notes

Computing Stylus Tip Coordinates

In many applications that require a sensor mounted on a tool or pointing device, the position of the tip is required. This type of pointing device is generically referred to as a stylus.

The sensor position and orientation values are presented with respect to the center of the sensor. The corresponding X, Y, Z coordinates at the tip of the stylus may be easily calculated knowing the tip offset from the sensor center.

The stylus coordinates can be computed from the following:

$$\begin{aligned} X_S &= X_B + X_O * M(1, 1) + Y_O * M(2, 1) + Z_O * M(3, 1) \\ Y_S &= Y_B + X_O * M(1, 2) + Y_O * M(2, 2) + Z_O * M(3, 2) \\ Z_S &= Z_B + X_O * M(1, 3) + Y_O * M(2, 3) + Z_O * M(3, 3) \end{aligned}$$

Where:

X_B, Y_B, Z_B are the X, Y, Z position outputs from the 3DGuidance™ sensor with respect to the transmitter's center.

X_O, Y_O, Z_O are the offset distances from the sensor's center to the tip of the stylus.

X_S, Y_S, Z_S are the coordinates of the stylus's tip with respect to the transmitter's center.

$M(i, j)$ are the elements of the rotation matrix.

Often the values of X_O, Y_O, Z_O are not known ahead of time and must be calculated. This may be done by placing the tip of the stylus at a set location, collecting data of the stylus being repositioned with the tip fixed, and solving for X_O, Y_O, Z_O . Since the tip location (X_S, Y_S, Z_S) is fixed, and the 3DGuidance™ sensor position (X_B, Y_B, Z_B) and orientation ($M(i, j)$) are reported by the system, solving for X_O, Y_O, Z_O may be solved.

Collect many measurement points over a large range of angles and rotations for maximum accuracy.

Explaining Measurement Rate vs. Update Rate

Measurement Rate is the rate at which the tracking system acquires a complete measurement set from all available axes in the transmitter.

Update Rate is the rate at which a new (unique) position and orientation solution for the sensor is computed by the tracker and available to the user.

Older Systems:

In previous generation ATC trackers, a position and orientation solution would only be available after *all* transmitter axes had been energized. Thus the Update rate was equal to the system Measurement rate.

$$UpdateRate = \frac{1}{a * t_A} Hz$$

Where:

a = Number of transmitter axes

$$t_A = \text{Axis cycle time} = \frac{1}{\text{Measurement Rate} * a} \text{ (sec)}$$

For example, for an older tracker configured to run at a Measurement rate of 80Hz with a standard mid-range transmitter (3-axis transmitter), the Update rate would be computed as:

$$UpdateRate = \frac{1}{3 * 4.167mS} = 80Hz$$

3DGuidance Systems

Next generation systems have the capability to compute a new position and orientation solution at the end of each transmitter axis cycle (see Fig 1 below). Both [dipole](#) and [non-dipole](#) transmitter based systems use proprietary algorithms for advancing the solution in time so that there is very little skew of the data. This yields an Update rate that is:

$$\begin{aligned} UpdateRate &= \frac{1}{t_A} \\ &= a * \text{Measurement Rate} \end{aligned}$$

Where again:

a = Number of transmitter axes

Using the same example of a system running at a Measurement Rate of 80Hz with a standard mid-range transmitter (3-axis), the Update Rate would be:

$$\text{Update Rate} = 3 * 80\text{Hz} = 240\text{Hz}$$

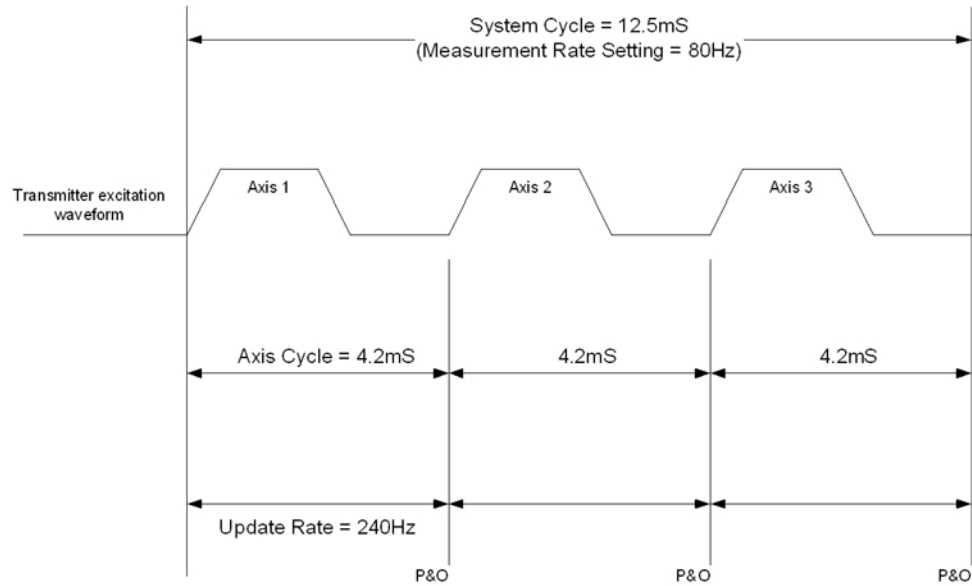


Figure 54 Transmitter Excitation Waveform