3D Guidance driveBAY 2™

Installation and Operation Guide

www.ascension-tech.com

3D Guidance driveBAY 2™

Installation and Operation Guide

REVISION NUMBER:
940051 Rev 3
08/30/2012

107 Catamount Drive
Milton, VT 05468

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

Table of Contents

List of Figures

List of Tables

Introduction

Congratulations on your purchase of our Ascension Technology Corporation (ATC) 3DGuidance driveBAY 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.

driveBAY 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 interface.

This User's Guide will help you set up and install the 3DGuidance driveBAY 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 driveBAY 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 seven 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 driveBAY 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: 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 7: 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 6. 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

Chapter 1

Preparing for Setup

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

System Requirements

Intended Use Statement

The 3DGuidance driveBAY 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 driveBAY 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 is 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: driveBAY 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.

Power: driveBAY 2 operates from an internal hard disk drive (HDD) power cable that provides +5V and +12V.

- + 12V: 1.6A nominal, 2.9A max
- + 5V: 1.0A nominal, 1.2A max

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 driveBAY 2 will arrive in one or two shipping boxes (depending on the configuration). Make sure you have received the following items before proceeding to the setup and checkout section:

Electronics Unit:

Figure 1 driveBAY 2 Electronics Unit

Sensor(s):

Valid sensor options for the driveBAY 2 include:

6DOF: Model 800, Model180, Model 130, Model 90, and Model 55.

5DOF versions of many sensors are also now supported via a Junction Box (see below).

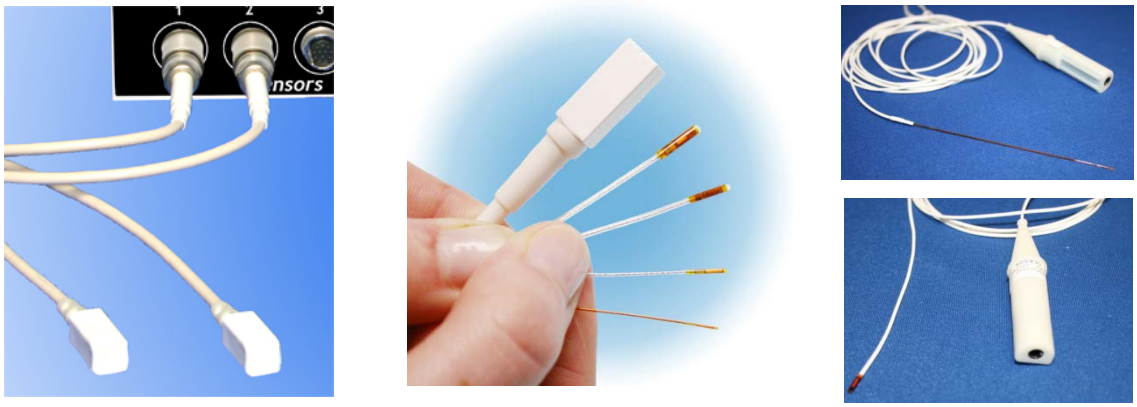
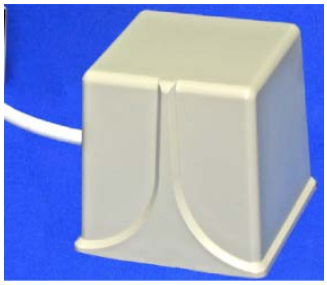


Figure 2 driveBAY 2 Sensors

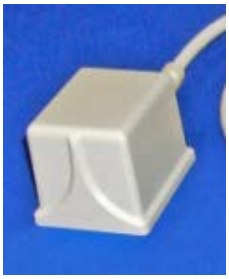
One or more of the following transmitters:

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

6DOF Transmitters:
These transmitters support operation with 6DOF sensors



Mid-Range Transmitter



Short-Range Transmitter

Figure 3 driveBAY 2 6DOF Transmitters

5DOF/6DOF Transmitters:
These transmitters support operation with 6DOF sensors AND 5DOF sensors

5DOF/6DOF Flat Transmitter

5DOF/6DOF MAGnet Transmitter

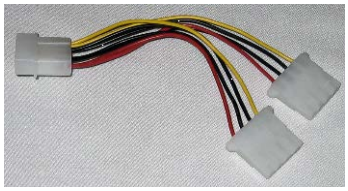
5DOF/6DOF miniMAG Transmitter

Figure 4 driveBAY 2 5DOF/6DOF Transmitters

Junction Box (Required for 5DOF Sensor connections):

Figure 5 driveBAY 2 5DOF Junction Box

Cables:



DC 'Y' Power Cable

Internal USB Cable

Hexagonal connector on one end and header connector on the other

Figure 6 driveBAY 2 Cables

driveBAY 2 CD-ROM:

Figure 7 3D Guidance driveBAY 2 CD-ROM

Four Mounting Screws: These are used to attach the driveBAY electronics unit directly to your PC chassis or to connect the mounting brackets.

Quick Setup Guide: Condensed list of procedures for quick and easy setup.

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

Additional Items

You Will Need For Setup

- Small Philips head screwdriver for fastening the EU mounting screws.

Depending on Your PC Configuration, You May Also Need the Following:

NOTE: Ascension does not provide the items listed below:

1 USB Cable (hexagonal connector on one end and rectangular connector on the other)

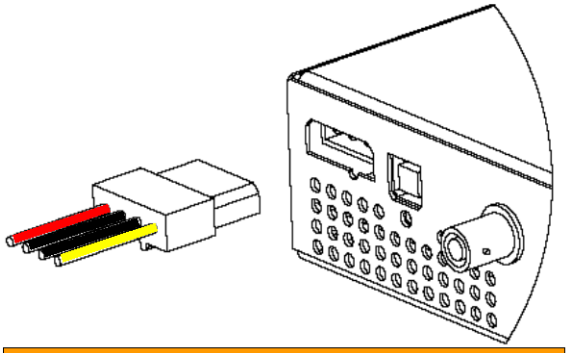
1 USB PCI Card: To make available an internal USB port when a header is not available on the motherboard. (Picture below is an example)

1 Set of Mounting Brackets: Usually included with your PC or customized for your PC (picture below is an example)

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.

- Caution - Read this Guide. This symbol cautions you to please refer to the safety-related notes in this manual
- Keep your computer turned off while installing the driveBAY2!



⚠WARNING

Check orientation of power harness before plugging in. Yellow wire on inside (right); Red wire on outside (left) – as shown
INCORRECT CONNECTION WILL DAMAGE THE UNIT AND VOID YOUR WARRANTY

- 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.
- Never power up the system or place the transmitter in an explosive atmosphere.
- 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), 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.

- 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.
- For **Periodic Maintenance and Inspection** recommendations, be sure to checkout the User Maintenance section (in Chapter 6)
- 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. See Cleaning and Disinfecting (in Chapter 6) for additional details
- Keep the transmitter, sensors, and cables away from sources of heat.
- If mounting the transmitter inside an enclosure, be sure to provide adequate ventilation.
- To extend tracker life, be sure to shut down the transmitter when not in use. You can do this in several ways:
    i. Select "No Transmitter" by setting the System parameter: SELECT_TRANSMITTER value to **'–1'** in the 3D Guidance API.
    ii. Recycle the power on the electronic unit (if independent access is available to the unit's power).
    iii. Disconnect and reconnect the USB cable.
- 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 driveBAY 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 driveBAY 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%

Chapter 2

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 driveBAY 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 driveBAY 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.

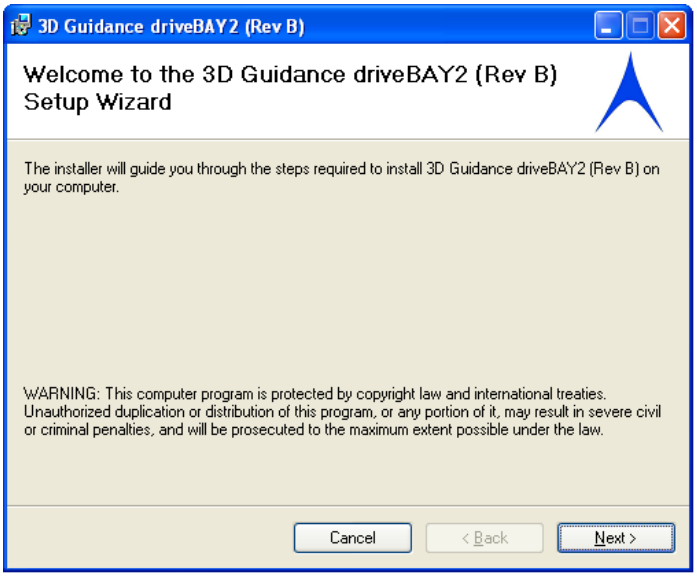


Figure 8 3D Guidance Setup Wizard

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

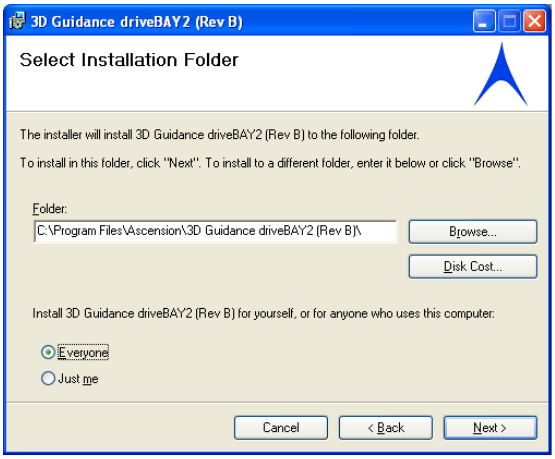


Figure 9 3D Guidance Setup Wizard – Confirming Target Folder

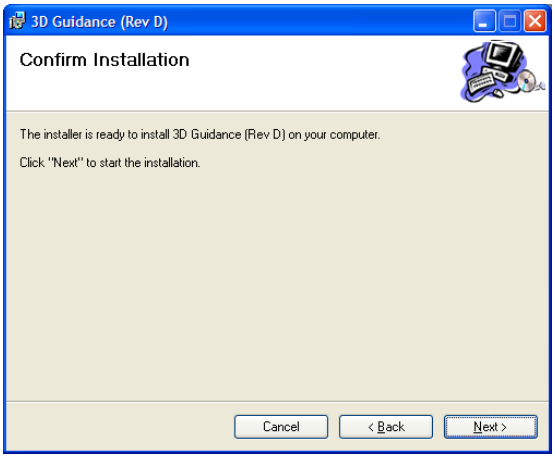


Figure 10 3D Guidance Setup Wizard – Confirming Installation

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 11 3D Guidance Setup Wizard – Installing USB Driver

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

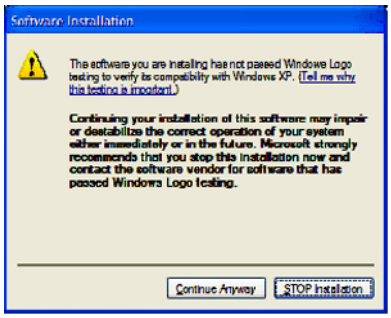


Figure 12 Windows Logo Testing Message Box

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

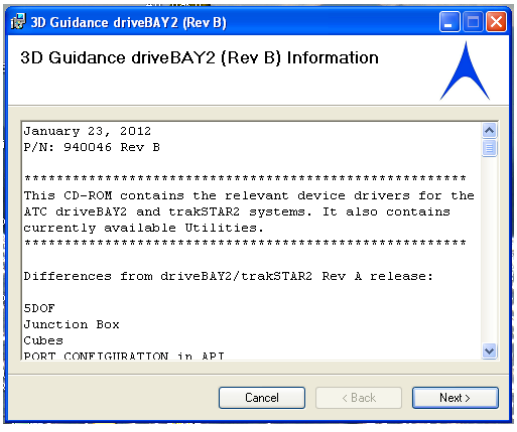


Figure 13 3D Guidance Readme File

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.

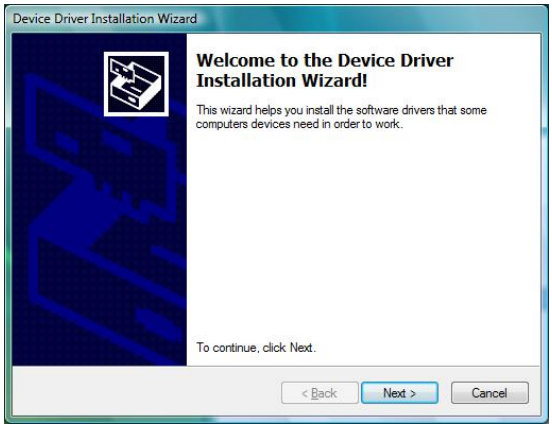


Figure 14 Windows-Vista Driver Installation Wizard

Installing the Hardware

The driveBAY 2 hardware can be installed and checked out in 4 easysteps:

- Installing the Electronics Unit.
- Attaching the External Cables.
- Installing the Drivers.
- Checking the system by running the demo.

Before You Begin

To install the driveBAY 2, you must open your computer, slide the driveBAY 2 into an open bay, and connect it to your computer's motherboard. This can be a tricky process, so it helps to be prepared by reading these instructions.

Follow these preliminary steps to make sure you understand the entire installation process and have everything you need to install the driveBAY 2 *before* opening your computer. This helps ensure that you can complete the installation all at once (and not get stuck part way through).

Check these steps off as you go.

- ❑ Read through all the steps in this chapter so that you understand and can picture the entire installation process.
- ❑ Unpack the driveBAY 2 and make sure you have everything you need to install it into your computer. See Unpacking the System.
- ❑ Locate your computer's guide. Make sure it describes how to open your computer as well as the layout of the computer's motherboard.
- ❑ Your computer must be running the Windows 7, Vista, or Windows XP operating system. Contact Ascension for the latest information on other supported operating systems.
- ❑ The driveBAY 2's dimensions are: 14.7 centimeters (cm) wide by 4.1 cm high by 17.7 cm deep. Make sure you have a desktop model computer with an open bay big enough for installing the driveBAY 2. (You cannot install the electronics unit in a portable or laptop computer.) You should be able to easily install the driveBAY 2 into your computer, although there is a chance that you might need separate mounting slides.
- ❑ Your computer's motherboard must have a USB header port. You should be able to determine this from your computer's guide.

One final word before you begin. Different computer models have different ways of opening and have different layouts on their motherboards. Because of this, these instructions can only guide you through the process of installing the driveBAY 2 into your computer — in other words, this part of the instructions are not specific to your particular computer. Your computer's guide should have specific instructions on how to open your computer and how to locate the USB header port on your computer's motherboard and the internal power cable. It also helps to be a bit resourceful. If unsure of what to do, contact us for immediate help.

Installing the Electronics Unit

This section explains how to install the driveBAY 2 electronics into your computer.

1. **Turn OFF your computer, but leave it plugged in.**

   Shut down your computer. Leave the power cord plugged into the computer and into the external power source (such as the wall outlet, power strip, or surge protector). Leaving the computer plugged in helps dissipate any static electricity in your hands.

2. **Open up your computer's chassis.**

   To open most computers, you must remove a side panel. You might also have to remove the computer's front panel (Figure 13). Refer to your computer's guide for specific instructions.

Figure 15 An open computer without panels

3. **Make sure your computer can accept the driveBAY 2.**

   Check the open bay to make sure that it is large enough and deep enough to accept the driveBAY2. Make sure the area behind the open bay is free of any wires and cables.

   Find a 4-conductor hard disk drive (HDD) power cable to plug into the driveBAY 2 (Figure 14). There might be a free cable that you can plug directly into the driveBAY 2. If not, you will have to install our HDD power cable Y-adapter.

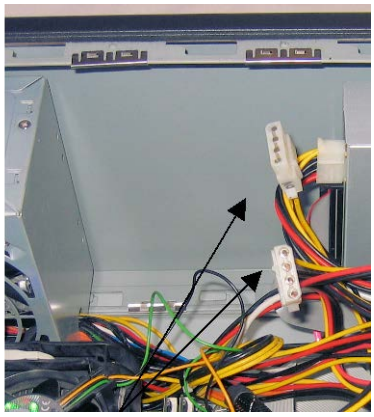


Figure 16 HHD power cables

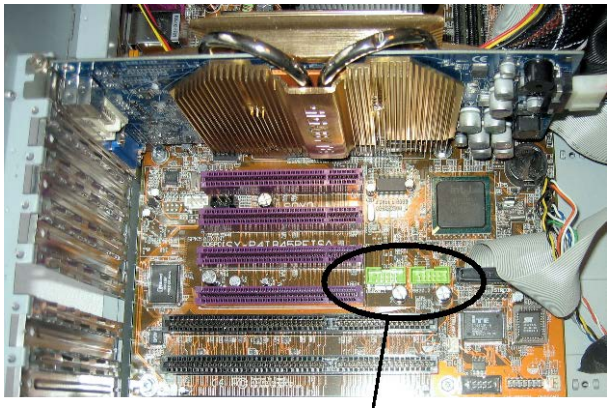


Figure 17 Two USB header ports (2 x 5 configurations) on your computer's motherboard

Locate the USB header port on the motherboard. The word 'USB' might be stamped on the motherboard to identify the ports (Figure 15). The USB header port is a rectangular port with pins in either a 2x4 or 2x5 layout: in other words, 2 rows of 4 pins in each row, or 2 rows of 5 pins in each row.

Again, refer to your computer's guide to help locate these components. If the open bay is not large or deep enough or the motherboard does not have a USB port, you cannot install the driveBAY 2. Close up your computer, and call Ascension technical support.

4. Gain access to the open bay.

Remove any dummy faceplates and metal shielding components from the open bay. Make sure you can access the open bay through your computer's front panel. You might also have to reposition any internal wires or cables to create the space necessary for the driveBAY (Fig 16).

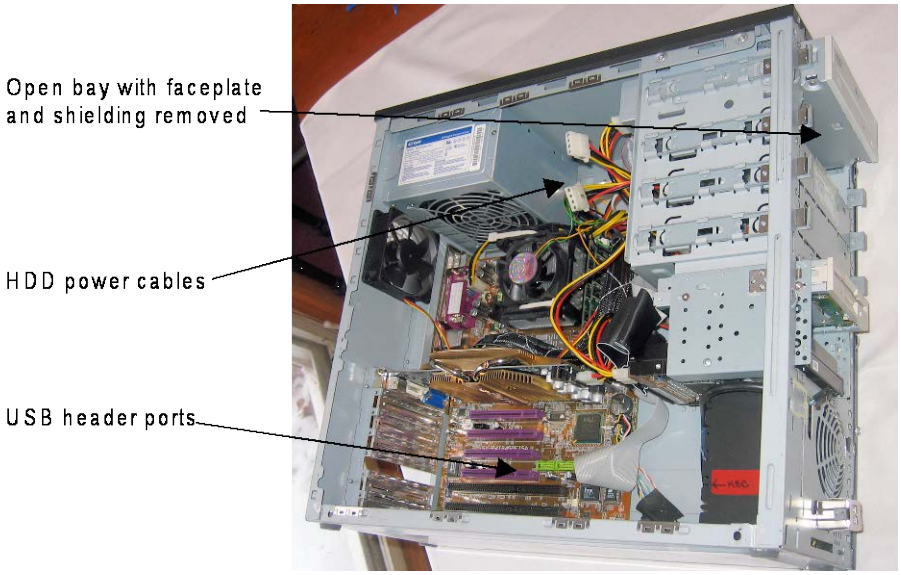


Figure 18 An inside look at a computer

5. Insert the driveBAY into the open bay.

Secure the driveBAY 2 onto the computer using the connector holes on the driveBAY 2's side (Figure 2-5).

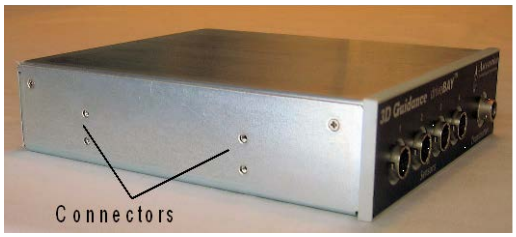


Figure 19 For connecting to the computer's chassis

Align the driveBAY 2 and slide it into the open bay until the driveBAY 2 is fully inserted into your computer (Figure 18).

Figure 20 Align and slide in driveBAY 2

Computers usually have a variety of holes and slots for securing devices into their bays. Align two of these holes or slots with the threaded holes on the side of the driveBAY 2. You might have to adjust the angle of the driveBAY 2 slightly to gain a perfect alignment. Using the Phillips screwdriver, insert the mounting screws through the holes or slots and into the driveBAY 2; tighten the screws until they are just snug. Avoid over-tightening the screws (Figure 19).

Figure 21 Screw in the driveBAY 2

Some computers use levers, detents, or snaps to secure devices into their bays (rather than holes or slots for screws). In these cases, you only have to slide the driveBAY 2 into the open bay until it clicks securely into these levers, detents, or snaps.

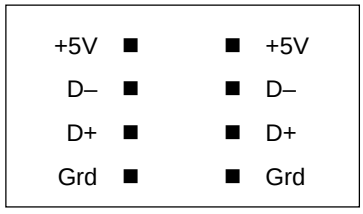
There is a slight possibility that you might not be able to slide the driveBAY 2 into your computer chassis without first attaching mounting slides to the sides of the driveBAY 2. In these cases, you need to provide your own mounting slides; some computers store mounting slides on the inside of its case. Align the mounting slides with the holes on the sides of the driveBAY 2, screw them into place, then slide the driveBAY 2 into your computer chassis; it should snap into place. Your computer guide should be able to provide more details.

6. Connect the USB cable to the motherboard.

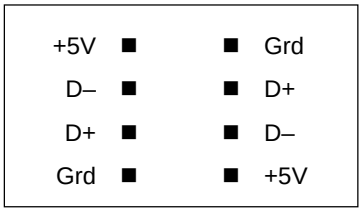
Plug the wired end of the internal USB cable into the USB header port on the motherboard.

Warning! Improperly connecting the internal USB cable to the USB header port can permanently and fatally damage your motherboard, the driveBAY 2, and any other peripheral connected to the motherboard. Make sure you carefully follow these instructions to avoid any problems.

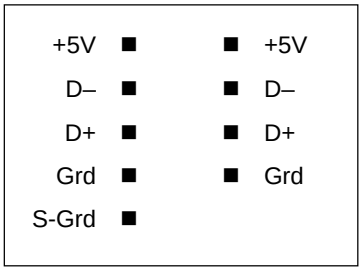
These four diagrams depict the four possible layouts of the pins in the USB header port:



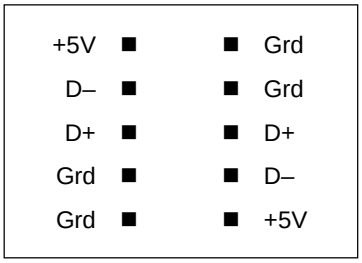
A 2x4 layout with pins in the same direction



A 2x4 layout with pins in the reverse direction



A 2x4 layout with pins in the same direction with an extra ground pin



A 2x4 layout with pins in the reverse direction with two extra ground pins

Figure 22 USB Header Port Layouts

Check with your computer's guide to determine the pin layout of the USB header ports on your motherboard. These USB header pins must align with the wired end of the USB cable.

Figure 21 shows a close up of the wired end of the internal USB cable that you use to connect the USB header port on your motherboard to the driveBAY. These colored wires must align correctly with the pins in your USB header.

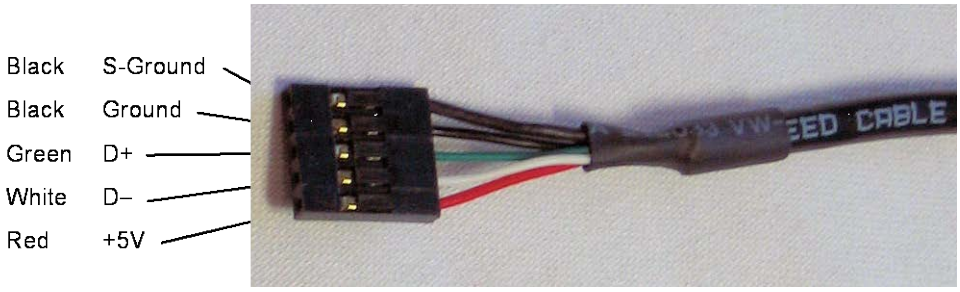


Figure 23 Color coded wire end of USB Cable

To correctly plug in the USB cable, align the red wired pin on the USB cable's wired end with the +5V pin on the motherboard's USB connector. All the remaining wires plug in correctly (Figure 22).

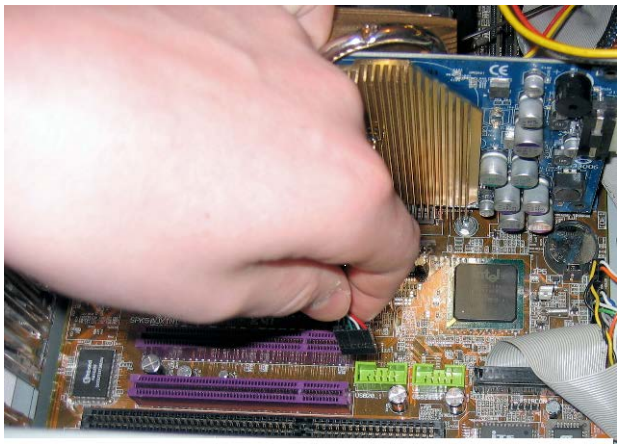


Figure 24 Plugging the wired end of the cable into the USB header port

7. Attach the power cable to the driveBAY 2.

Attach an unused HDD power cable into the back of the driveBAY 2 (Figure 23).

If there are no unused power cables, then you must first install the HDD power cable Y-adapter:

a. Unplug a HDD power cable from another device inside your computer.

b. Plug this HDD power cable into the bottom of the HDD power cable Y-adapter.

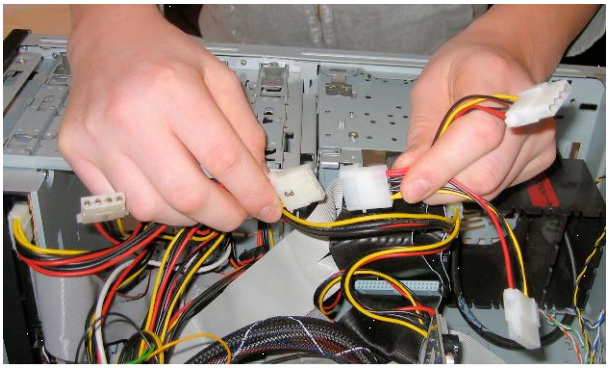


Figure 25 Plug the Y adapter into the HDD power cable

c. Plug one end of the Y-adapter into the device you just removed the cable from.

d. Plug the other end of the Y-adapter into the back of the driveBAY 2 (Figure 24).

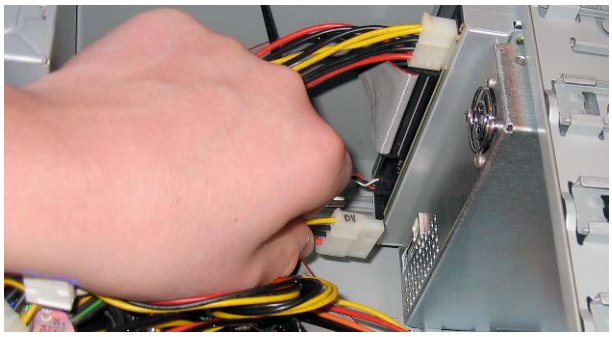
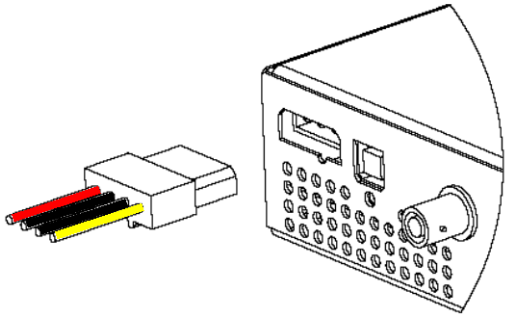


Figure 26 Plug the HDD power cable or the Y adapter into the back of the driveBAY 2



⚠WARNING

Check orientation of power harness before plugging in. Yellow wire on inside (right); Red wire on outside (left) – as shown
INCORRECT CONNECTION WILL DAMAGE THE UNIT AND VOID YOUR WARRANTY

8. Connect the USB cable to the driveBAY 2.

Connect the Type B end of the USB cable to the back of the driveBAY 2 (Figure 25).

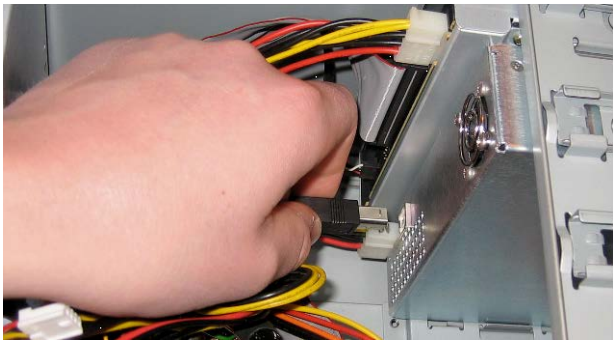


Figure 27 Connect the YSB cable's Type B end to the back of the driveBAY 2

9. Close up the computer.

Position the panels back onto your computer and secure them into place. If necessary, make sure the front panel of your computer is aligned and snug with the front of the driveBAY 2(Figure 26).

Figure 28 driveBAY 2 Installed

Attaching the External Cables

Now attach the transmitter and sensors to the front of the driveBAY.

1. **Connect a Transmitter –**Plug the transmitter cable into the corresponding connector on the right 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 29 Connecting the Transmitter

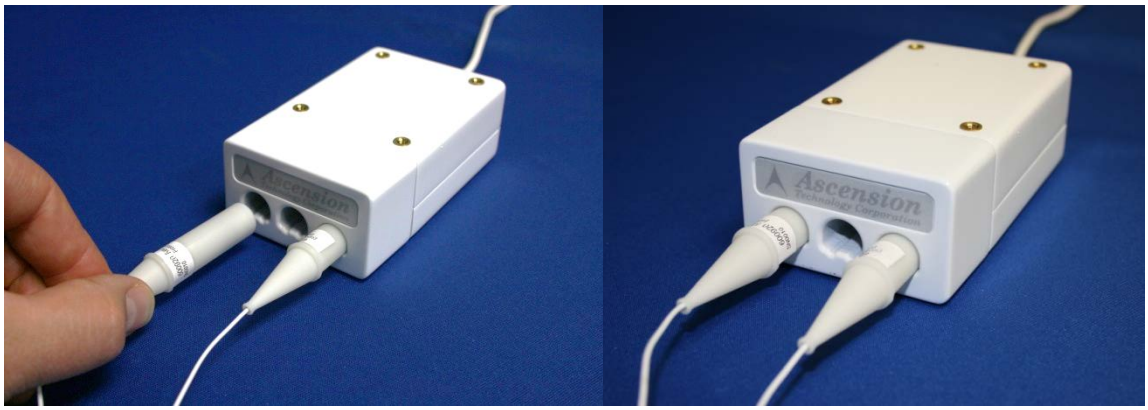
2. **Connect the 6DOF Sensor(s) to the Electronics Unit –** Plug the connector of each 6DOF sensor into a receptacle on the electronics unit. Be sure to fully insert the connector, until audible 'click' is heard.

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 30 Connecting the 6DOF Sensors

3. **Connect the Junction Box to the Electronics Unit –** If you're system came with a Junction Box and 5DOF sensors, plug the metal connector of the Junction Box into any of the 4 'Sensor' receptacles on the electronics unit (labeled '1','2','3', or '4'). Be sure to fully insert the connector, until audible 'click' is heard.

    a. **Connect the 5DOF sensor(s) to the Junction Box –** Plug the plastic 5DOF sensor connector into the front of the Junction Box.



Note: driveBAY2 electronics units are pre-configured for 6DOF operation. If you are connecting 5DOF sensors (via the Junction Box), there are a few power-up settings you'll need to change after you get the system running. See Configuring for 5DOF operation in Chapter 3.

4. **Turn on your computer.**
   Because the driveBAY 2 is connected to your computer's power supply, the driveBAY 2 starts up when you turn on your computer.

Note: When power is applied, the driveBAY 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 driveBAY 2 until the front panel LED is blinking GREEN - (see the LED Definitions table in Chapter 6 for more information).

Installing the Driver

The **New Hardware Wizard** displays after turning on the computer 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 31 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*.

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

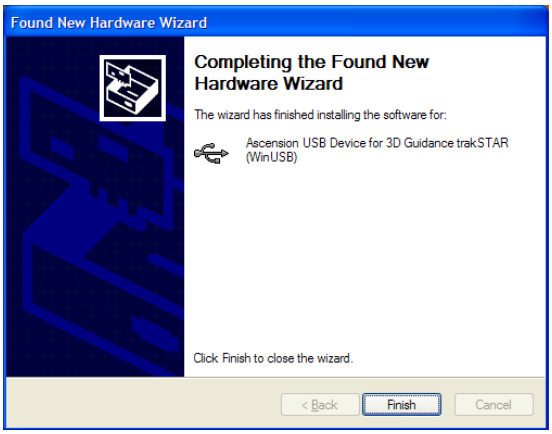


Figure 32 Found New Hardware (Closing Wizard)

System Checkout: Running the USB Demo Software

With the driveBAY 2 running and the drivers and utilities installed, you are ready to run the demo software and checkout the operation of your tracker.

Note: For 5DOF sensor operation you'll need to first change your front panel port configuration. See Configuring for 5DOF Operation in Chapter 3.

Note: While the Cubes demo utility is an excellent way to become familiar with the features of the tracker, it has some limitations as a data collection tool. For example, it is not possible to obtain the full update rate from the tracker (typical update rates from Cubes 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.

1. Select *CUBES* from the **Ascension Program Group** in the *Windows® Start* menu to start the Demo Utility.

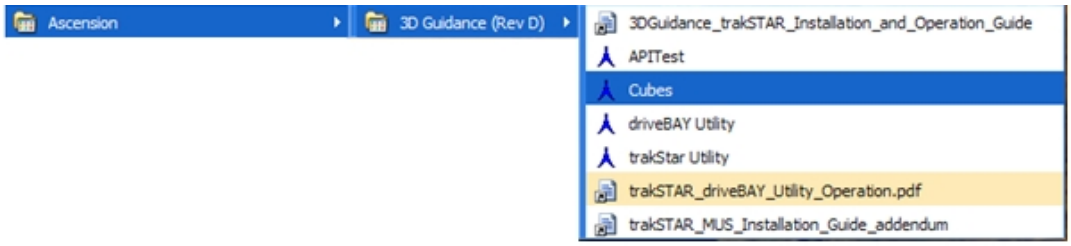


Figure 33 Location of Demo Utility (CUBES)

After a brief pause (~20sec), the CUBES window displays

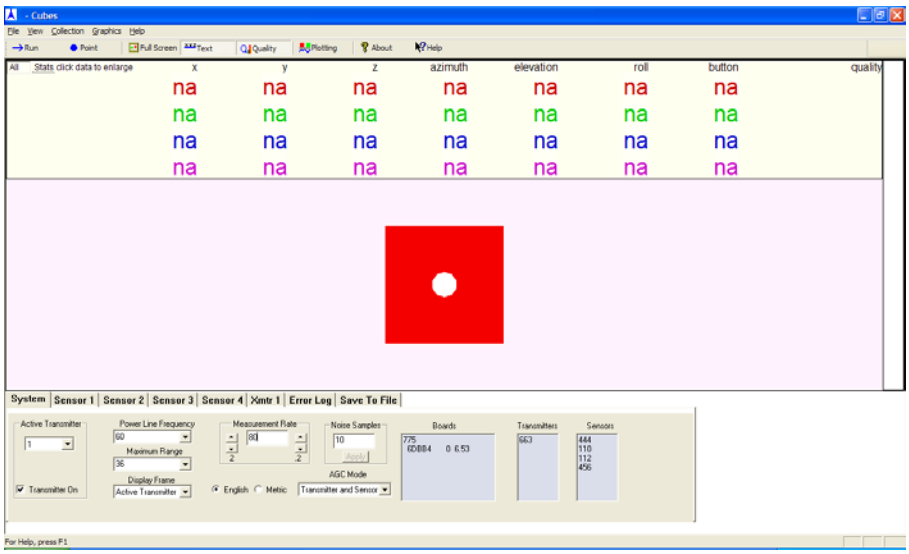


Figure 34 CUBES Window

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

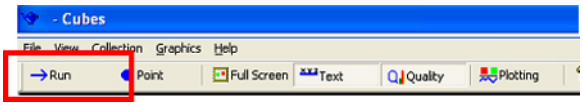


Figure 35 Location of CUBES Run Button

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.

NOTE that if a data record for a sensor is not displayed, Cubes will display a message in that sensor's row which reflects the Run Time Device Status Code being returned by the tracker. In Figure 36 for example:

- The sensor plugged into Port 2 is the wrong type for the unit's current Port setting (i.e 5DOF sensor in 6DOF port). See Configuring for 5DOF Operation. The status code returned is 'INVALID_DEVICE'.
- The sensor in Port 3 does not have enough data yet to satisfy the Position/Orientation algorithm. The status code returned is 'ALGORITHM_INITIALIZING'.

See the Status Codes paragraph in the Quick Reference section.

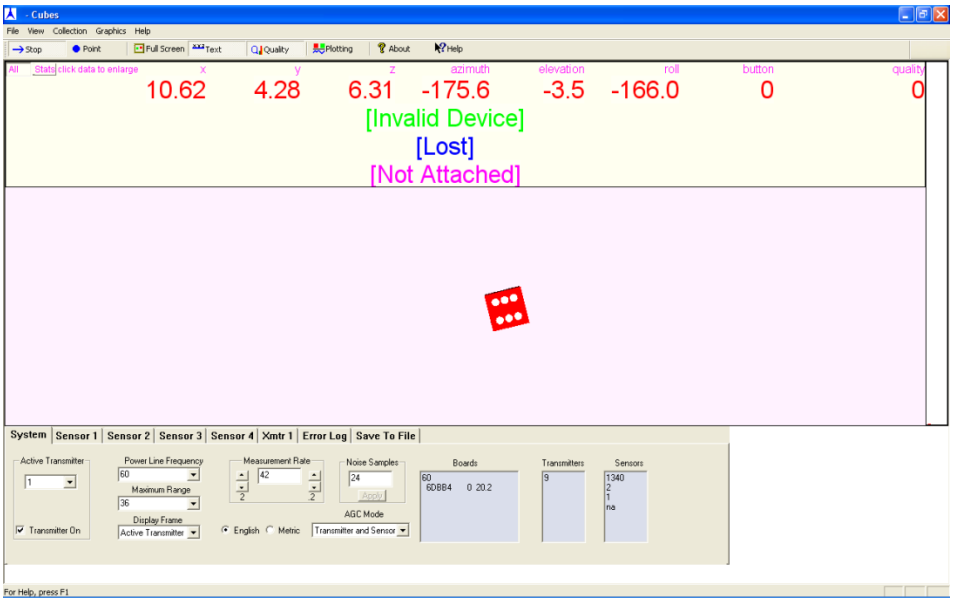


Figure 36 Sensor Device Status Code (run-time) displayed in Cubes

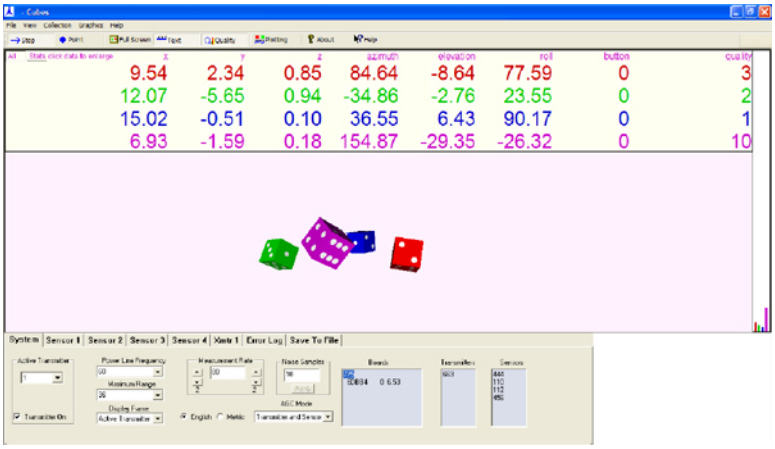


Figure 37 Typical Data Collection Position and Orientation Display

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 (trakSTAR configurations only).

The last column in each row (and the vertical bar on the far right) 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.

Note: The value of the Quality number displayed in this Cubes demo utility has been scaled down by 256 from the value returned by the API.

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 both the color of the numbers shown in the rows of sensor data, and the color of the quality indicator on the right.

The bottom of the window includes a 'System' tab that allows you to configure and adjust all System parameters (Measurement Rate, Metric units, Max Range Scale Factor, Power Line Frequency), turn on and off the transmitter, and to display the serial number of all connected components.

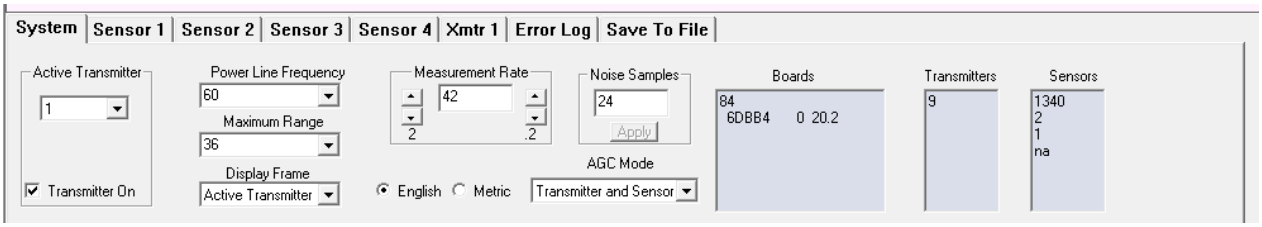


Figure 38 Cubes System Tab

It also includes tabs providing information and access to current parameter settings for the individual components – sensors and transmitters.

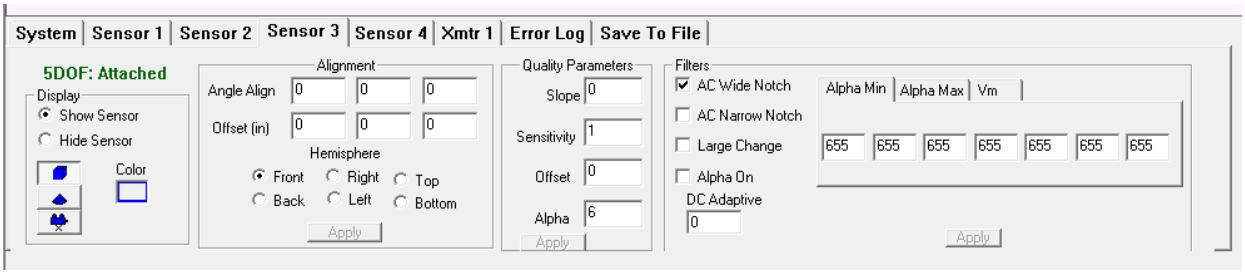


Figure 39 Cubes Sensor Tab

Note: To enable 5DOF sensor operation, you must first change your front panel port configuration. See Configuring for 5DOF Operation in Chapter 3.

MORE SENSORS:

If you've connected more than one 5DOF sensor per front panel port (i.e. two or more in a Junction Box) and you'd like to have Cubes display ALL connected sensors, you'll need to do two things:

1) Configure the corresponding front panel ports of your electronics unit for 5DOF Operation. See Configuring for 5DOF Operation

2) Set the AUTOCONFIG parameter to '12'. See AUTOCONFIG.

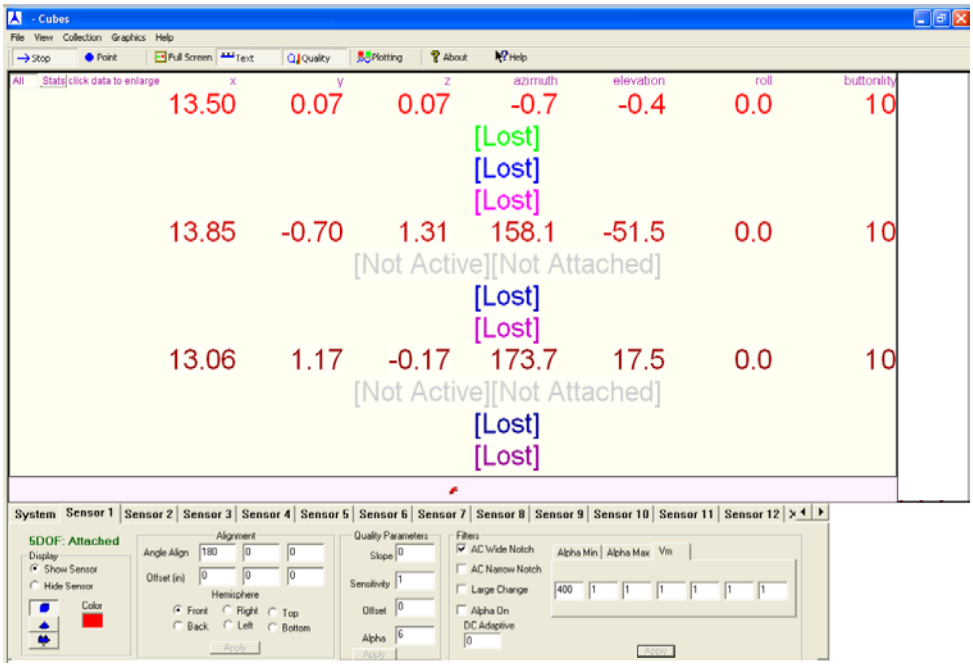


Figure 40 Running Cubes with AUTOCONFIG set for 12 Sensors

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 Cubes demo utility is an excellent way to become familiar with the features of the tracker, it has some limitations as a data collection tool. For example, it is not possible to obtain the full update rate from the tracker (typical update rates from Cubes 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.

Chapter 3

Configuration and Basic Operation

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

When the driveBAY 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:

3. Review the list of configurable default settings.
4. Determine which (if any) of the settings you would like to change.
5. 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), and **Mid-Range** (MRT) 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), **5DOF/6DOF MAGnet** (MAG), **5DOF/6DOF miniMAG** (miniMAG) transmitters.

Table 1 Power Up Default Dipole Settings:

Feature	Setting	Utility	SW call
System Settings:			
Baud Rate: (RS232 only)	115200		
Sync Mode:	Internal	✓	
Multi-Unit ID	0	✓	
Measurement Rate:	80.0 Hz **Note: that the Available Update Rate is always X times the measurement rate, where X is 3 for the SRT and MRT transmitters.**	✓	✓
Scale:	SRT and MRT Systems: 36 inches	✓	✓
Sleep on Reset:	Enabled	✓	
Full POST on Reset:	Disabled	✓	
Enable LED:	Enabled	✓	
Power Line Freq:	60Hz	✓	✓
Report Rate:	1	✓	
Commutation:	Disabled	✓	✓
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, 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 **Note:** **The Available Update Rate for Flat9 transmitter is always: 42 x9 = 378 updates/sec** **Available Update Rate for MAGnet and miniMAG transmitters is always: 42x12 = 504 updates/sec**	✓	✓
Scale:	36 inches	✓	✓
Sleep on Reset:	Enabled	✓	
Full POST on Reset:	Disabled	✓	
Enable LED:	Enabled	✓	
Power Line Freq:	60Hz	✓	✓
Report Rate:	1	✓	
Commutation:	Disabled	✓	✓
Transmitter Settings:			
Reference Frame (deg):	az = 0, el = 0, rl = 0	✓	✓
Enable XYZ Reference Frame	Disabled	✓	✓
Sensor Settings (per port)			
Algorithm	6DOF **Note:** **This setting must be changed to 5DOF to utilize sensors connected to the 5DOF Junction Box. See below**	✓	
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	✓	✓

Feature	Setting	Utility	SW call
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, 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	Establishes the default RS232 communication rate for power-up. Only supported Baud Rate for the system is 115200. **Note: Currently the only supported Baud Rate for RS232 communication is 115200. RS232 interface available on trakSTAR 2 systems only.**
Sync Mode	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. **Note: Multi-Unit synchronization (MUS) not supported at this time.**
Multi-Unit ID	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.
Measurement Rate	Sets the acquisition rate for the system. This can be altered to optimize susceptibility to distortion from certain metals. See Performance Factors section below.

Setting	Description
Scale	Sets the scale factor used by the tracker to report the position of the sensor with respect to the transmitter. Valid values of 36, 72, and 144 represent the full-scale position output in inches.
Sleep on Reset	When enabled, this setting will cause the driveBAY 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.
Full POST on Reset	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 InitializeBIRDSystem (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. 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 6 for more information
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.
Commutation	This is an alternate acquisition mode utilized by the tracker when running integrated sensor modules (i.e. M1525). This mode should be disabled unless running these integrated sensor types. Note that the system's sensitivity to conductive metal will increase slightly when it is being used.
Reference Frame	The default driveBAY 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.

Setting	Description
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.
SENSOR SETTINGS	
Algorithm **Note: This setting must be changed for 5DOF sensor operation. See Configuring for 5DOF Operation below.** .	When using a non-dipole (flat) transmitter, each sensor port can either track a single 6DOF sensor or up to three 5DOF sensors, connected to the unit via a Junction Box. The tracking mode is chosen at startup based on this setting. There are three options: • 6DOF - the port always computes a 6DOF tracking solution; A solution with six degrees of freedom (6DOF), includes three positional coordinates (X,Y,Z) and three orientation angles o Azimuth – rotation about Z sensor axis o Elevation – rotation about Y sensor axis o Roll – rotation about X sensor axis • 5DOF – the port always computes a 5DOF tracking solution; A solution with five degrees of freedom (5DOF), includes three positional coordinates (X,Y,Z) and two orientation angles o Azimuth – rotation about Z sensor axis o Elevation – rotation about Y sensor axis • 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. Once the port is set on power-up, it cannot be changed without restarting the tracker.
Hemisphere	This parameter is used to tell the driveBAY 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 (Flat, MAGnet, miniMAG) only support the FRONT hemisphere.

Setting	Description
Sensor Offsets	Default position outputs from the driveBAY 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 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. Many data formats are available from the tracker – See Data Format Types for details
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 driveBAY 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. For Dipole transmitter configurations, this filter is usually 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. **NOTE:** Non-Dipole transmitter configurations (Flat9, MAGnet, miniMAG) should not be run with this filter enabled, as the Kalman Filter utilized in the Non-Dipole solution has similar properties and enabling this Adaptive filter there will only add latency.
Kalman Filter	These parameters are used to tune the static and dynamic tracking performance when using non-dipole transmitters. Most of these settings should 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

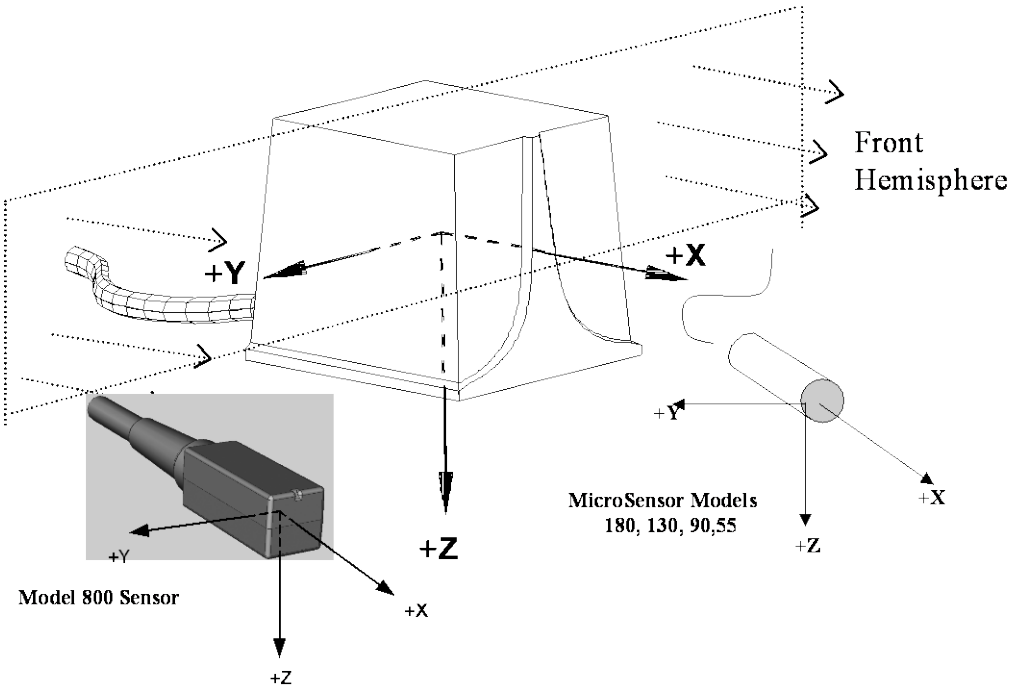


Figure 41 Default Transmitter/Sensor Coordinate Frames

NOTE: The origin of the Mid and Short 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

Flat Transmitter

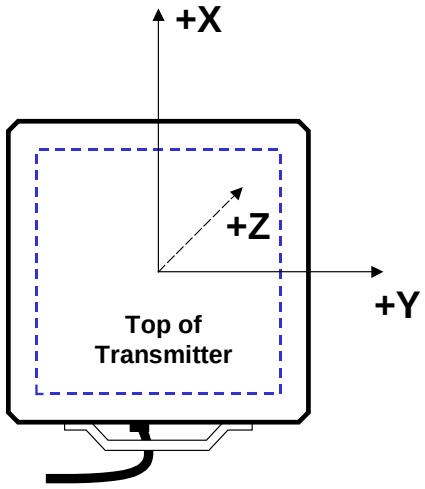


Figure 42 Flat Transmitter Coordinate Frame

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

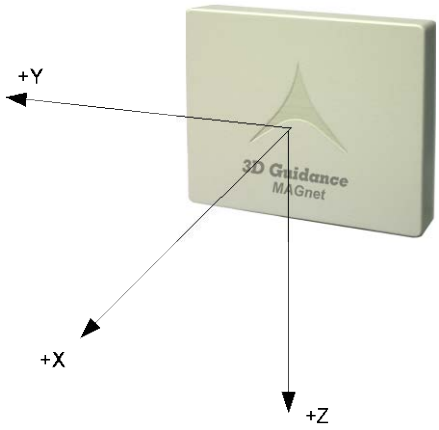


Figure 43 MAGnet Transmitter Coordinate Frame

5DOF/6DOF MAGnetTransmitter Origin:
In the middle of the square that defines the top surface

- 4.0 ” from left/right edge
- 4.0” from top/bottom edge
- 0 ” from the top face

miniMAG Transmitter

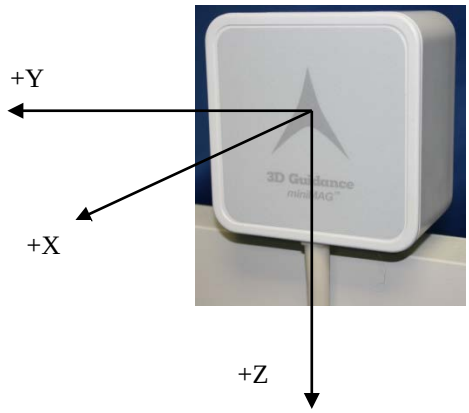


Figure 44 miniMAG Transmitter Coordinate Frame

5DOF/6DOF MAGnetTransmitter Origin:

- 4.0 ” from left/right edge
- 4.0” from top/bottom edge
- Approx. -1.0 ” from the top face (inside the transmitter)

Changing Your Settings

The *driveBAY2 Utility* may be used to alter power-up defaults for driveBAY 2 and trakSTAR 2 systems. See Appendix I: driveBAY 2 Utility for full details on using the Utility.

Configuring for 5DOF Operation

The default configuration for the driveBAY 2 and trakSTAR 2 front panel sensor ports is to return a 6DOF solution (for 6DOF sensors). However, the units can also be setup to run 5DOF sensors as follows:

1) Power on the tracker, connect to your host PC, and run the ***driveBAY2 Utility***.

2) On the ‘**NonDipole Settings’** tab, find the Sensor tab for the front panel connector that you’d like to use for your 5DOF sensor connections (i.e. Sensor Port 1,2,3, or 4).

3) Change the following setting to utilize your 5DOF sensor (see Figure 45 below):
   a. Algorithm : **5DOF**

4) Click ‘Write to Tracker’ to change this power-up default setting, and re-boot the tracker.

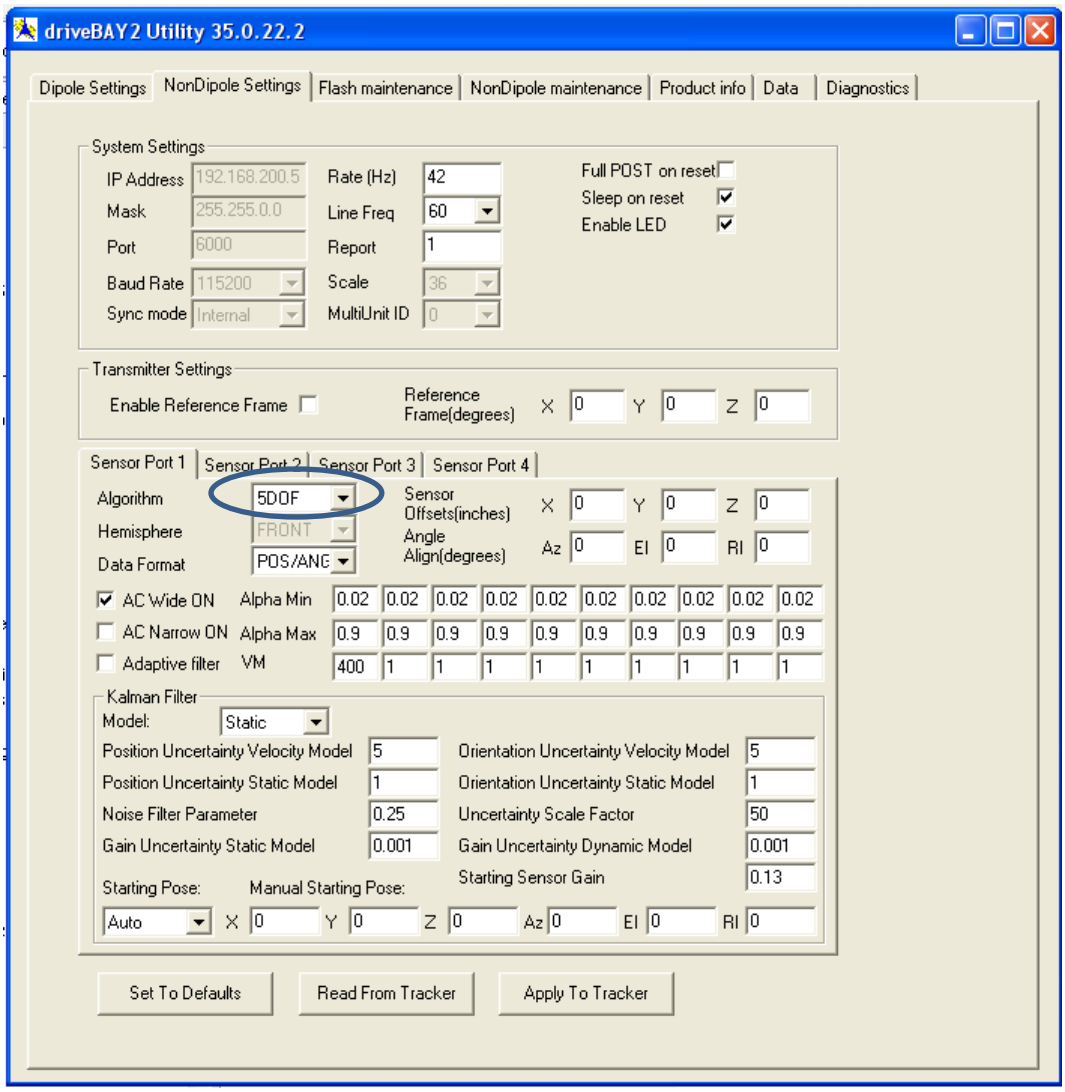


Figure 45 Changing settings for 5DOF operation on Sensor Port1

5) If you'll be running more than one 5dof sensor in the same port (i.e. 3-sensor Junction Box), then you'll also want to change the default AUTOCONFIG parameter. Read the next sections Sensor Mapping and AUTOCONFIG for details.

Sensor Mapping

The electronics unit of the tracking system has four sensor ports (connectors) on the front panel. On each of these, a user can pre-configure the the sensor port to optionally connect:

- A single six degree of freedom (6DOF) sensor
- A single, double or triple five degree of freedom (5DOF) sensor
- A 5DOF Junction Box (containing one, two or three 5DOF sensors)
- No sensor –connector port left blank

This allows a total from 1 to 12 tracked object channels to be supported by the system.

The question is how does an application make the association between the physical tracking device (the sensor) and the logical channel number?

The tracking system addresses this by assigning **Fixed Sensor ID** numbers to the tracking device(s) plugged into a connector. This 'sensor mapping' is as shown in Figure 46 below.

The figure shows three scenarios

- Scenario#1:
    - All sensor ports have been pre-configured for 6DOF sensors;
    - All ports are full.

- Scenario#2:
    - All sensor ports have been pre-configured for 5DOF sensors;
    - 2 contain 5DOF sensors that share a single front panel connector;
    - 2 contain a 5DOF Junction Box with three 5DOF sensors plugged into each slot.

- Scenario#3:
    - Sensor ports 1 – 3 have been pre-configured for 6DOF sensors;
    - Sensor port 4 has been preconfigured for 5DOF sensors
    - Sensor port 3 is empty
    - Sensor port 4 contains a 5DOF Junction Box with 2 5DOF sensors plugged into the first and third slot.

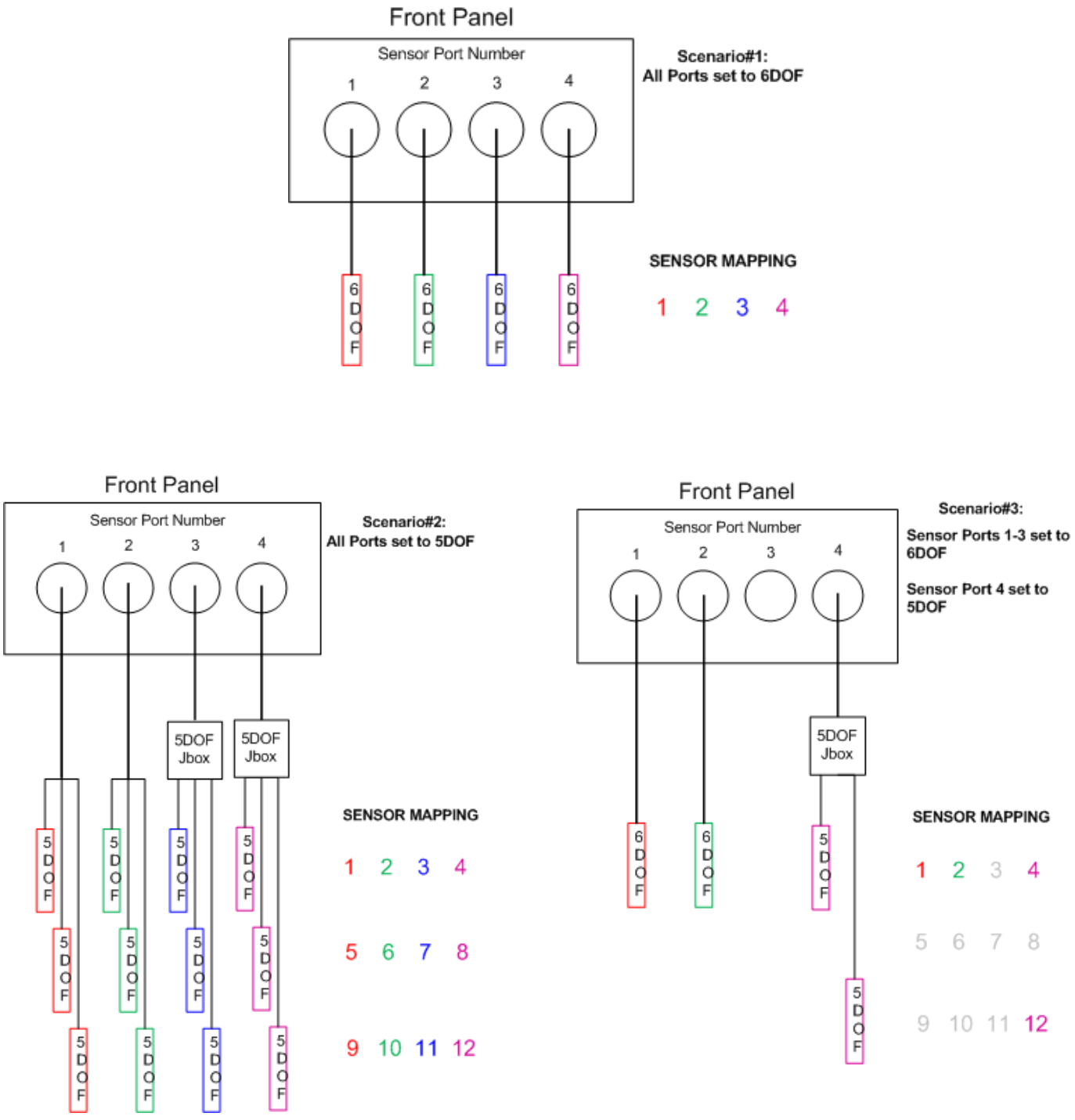


Figure 46 Three scenarios demonstrating the Fixed ID sensor mapping

AUTOCONFIG

The AUTOCONFIG parameter allows the host software to specify the number of sensors the tracking system should monitor and report data for when queried. It is set to '4' by default, but can easily be changed using either of the two methods listed below.

WHEN TO USE:

The scenarios depicted in the previous section on Sensor Mapping (Figure 46), demonstrate that the host software may elect to set the AUTOCONFIG parameter differently depending on what connections are expected in the application. In Scenario #1 for example, leaving the AUTOCONFIG parameter set to '4' will provide access to all possible sensors. For Scenario#2 and Scenario#3 however, the AUTOCONFIG parameter must be set to '12' to access all possible sensors being tracked by the system.

For Non-Dipole configurations, 4-12 are valid values for this parameter.

When running Dipole configurations, any setting of the AUTOCONFIG parameter greater than 4 will be ignored by the system.

HOW TO CHANGE IT:

1) Change the parameter prior to an InitializeBIRDSystem() function call, using the SetSystem Parameter Command and the AUTOCONFIG SYSTEM_PARAMETER_TYPE.

Note: See the Pre-Init Commands section for details of additional parameters that can be changed prior to Initialization.

Tip: Start the APITest.exe utility (from the Ascension Program Start Menu) to quickly change the AUTOCONFIG parameter.

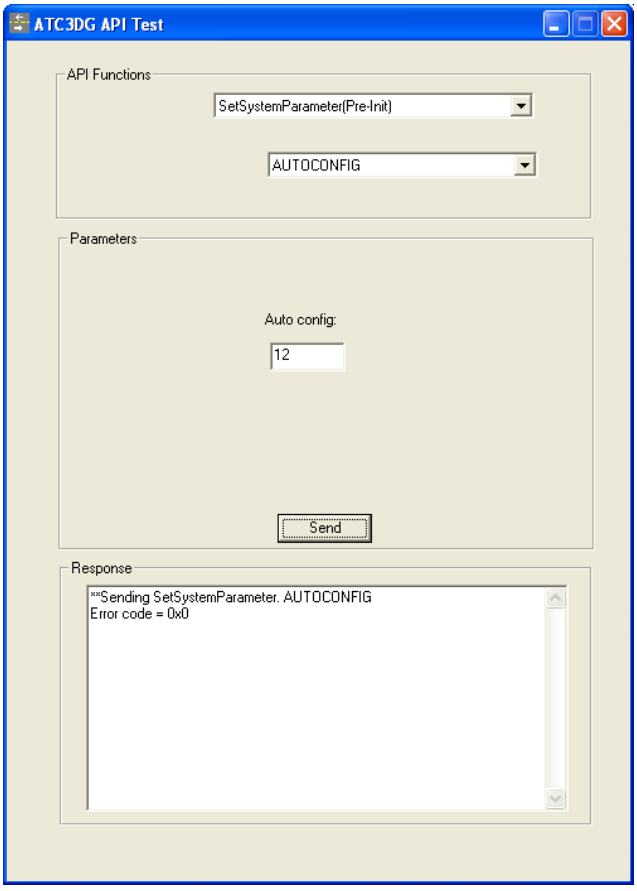


Figure 47 Setting the AUTOCONFIG parameter to '12' using the APITest Utility

OR:

2) Change the parameter directly in the ATC3DG.ini file:

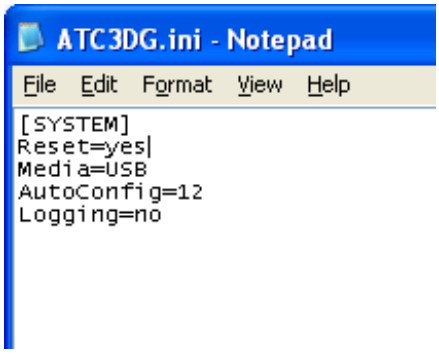


Figure 48 Setting the AUTOCONFIG parameter to '12' directly in the ATC3DG.ini file

Per Microsoft's intent, the 3DGuidance API now places the ATC3DG.ini file in the %APPDATA%\Ascension\ATC3DG\ folder which is typically:

C:\Documents and Settings\[username]\Application Data

or

C:\Users\[username]\AppData\Roaming\Ascension\ATC3DG\

This is a hidden, but user-writable, folder. The %APPDATA% folder can be easily opened by selecting Run... from the Start Menu and entering "%APPDATA%".

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.

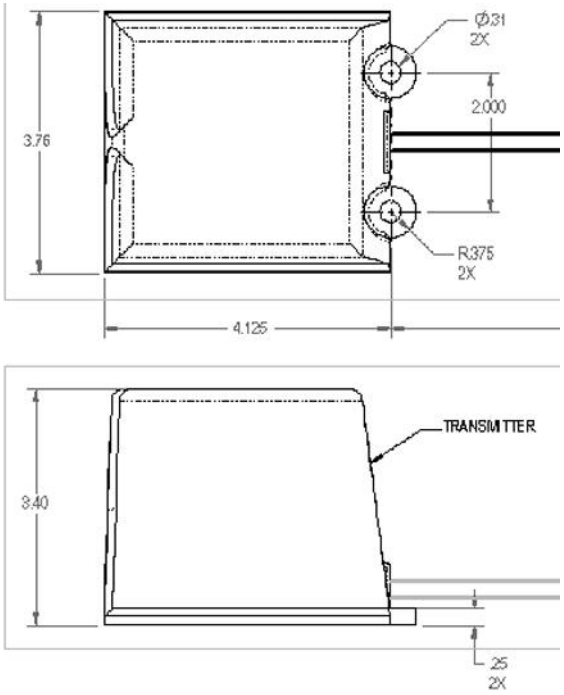


Figure 49 Mid-Range Transmitter Mounting Dimensions (inches) -Top and Side Views

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.

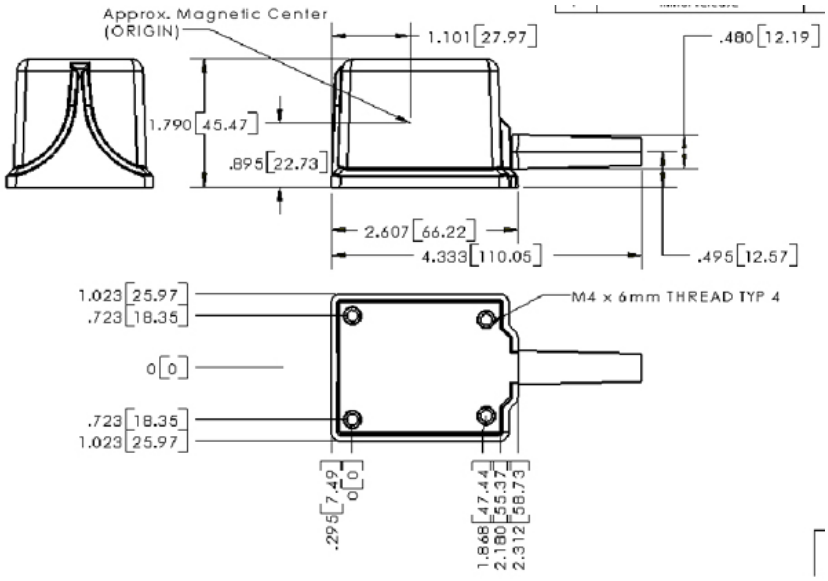


Figure 50 Short-Range Transmitter Mounting Dimensions (inches) Top and Side Views

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.

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 and miniMAG Transmitter Mounting

Mount the MAGnet and miniMAG transmitters to a non-metallic surface such as wood or plastic, using non-metallic bolts or 300-series stainless steel bolts. All mounting inserts sized for **M4** thread. For optimum performance, keep the transmitter at least twenty-four inches away from the electronics unit.

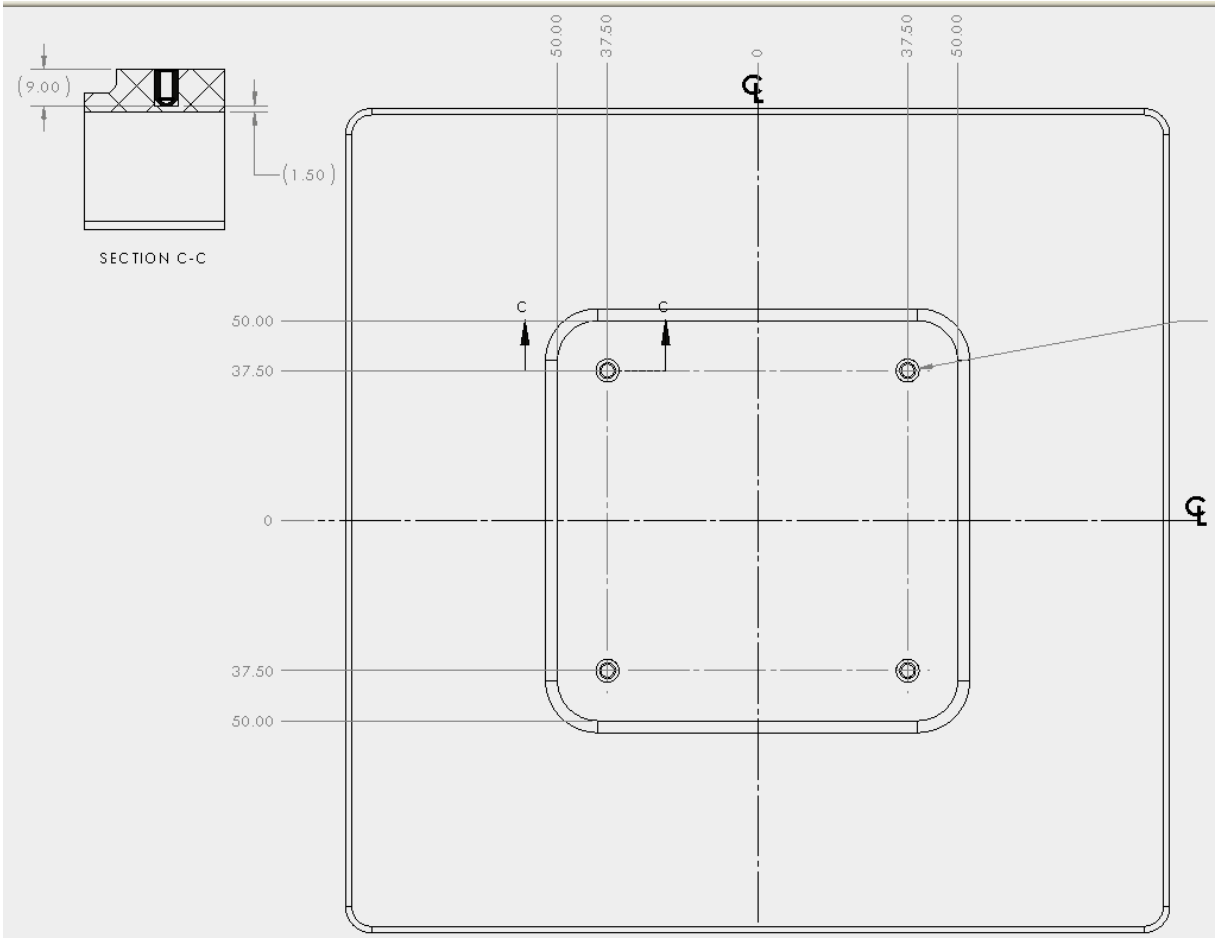


Figure 51 MAGnet Transmitter Mounting Hole Dimensions (mm)

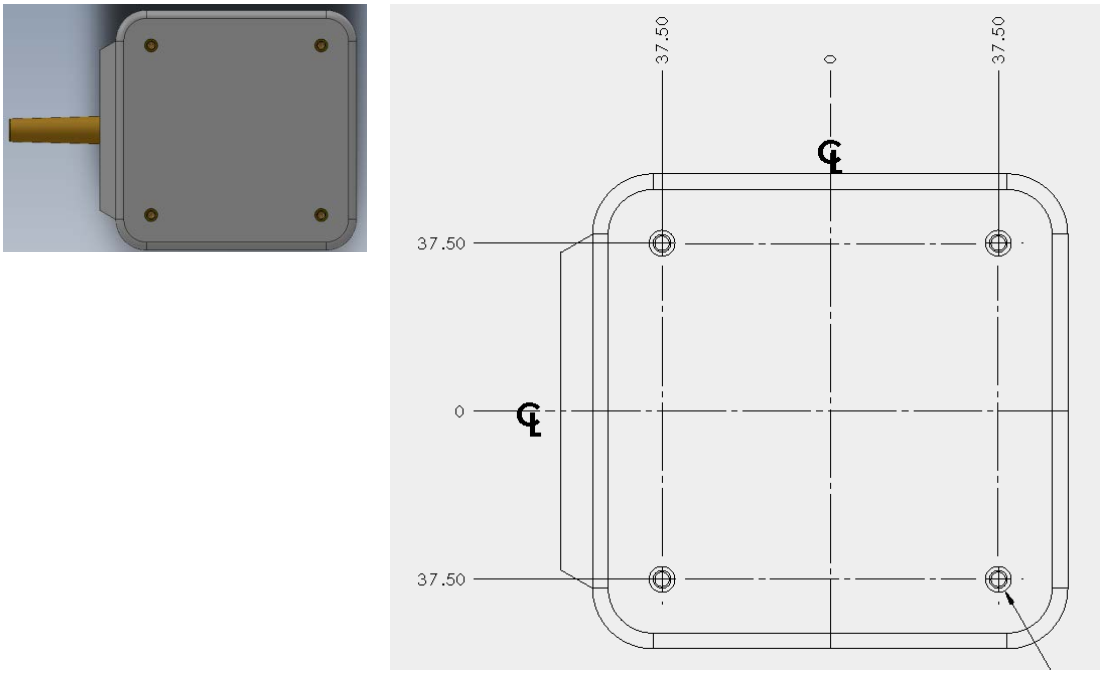
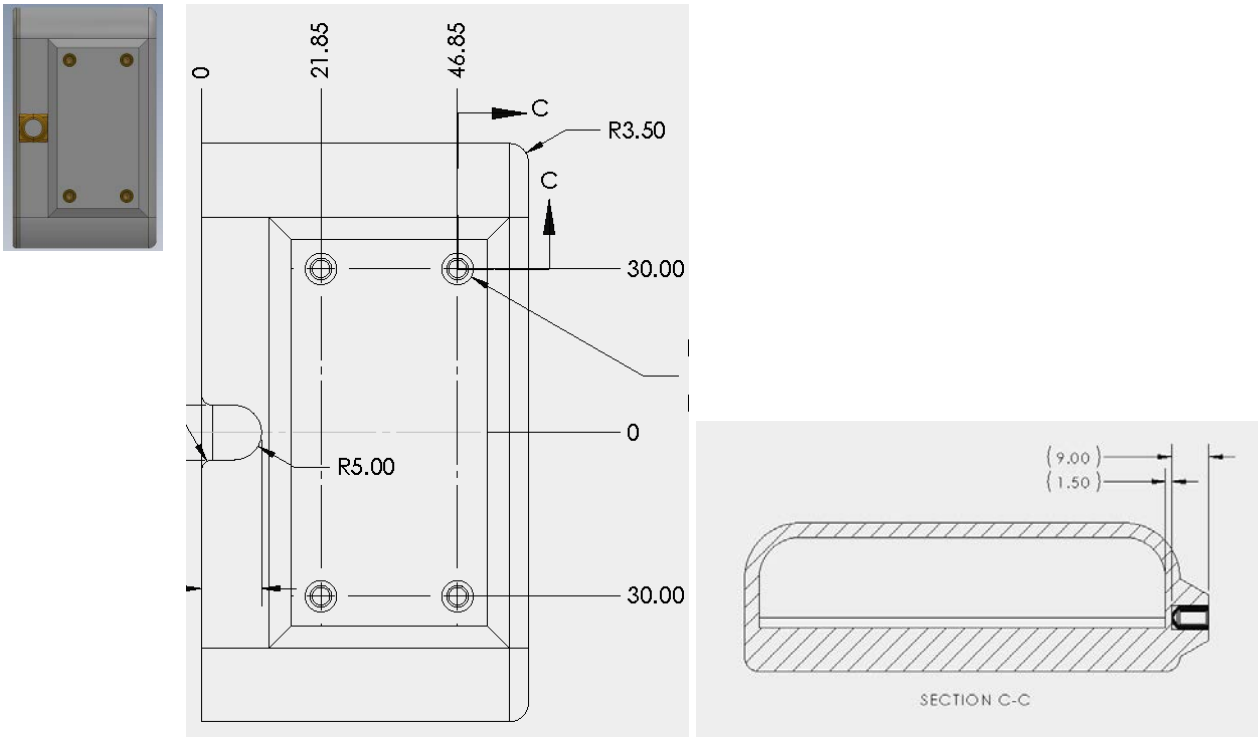
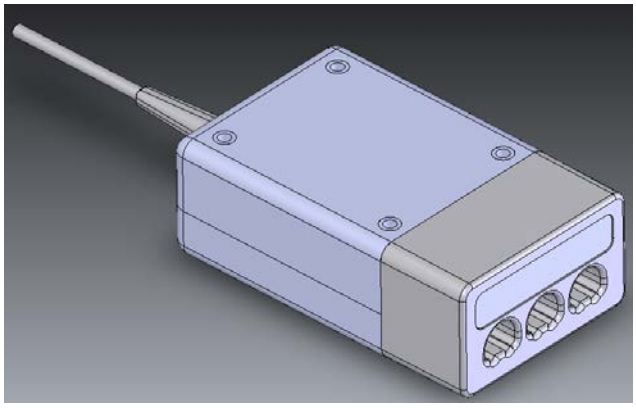


Figure 52 miniMAGTransmitter Mounting Hole Dimensions (mm);
Rear (upper) and Bottom (lower) mounting surfaces shown



Junction Box Mounting



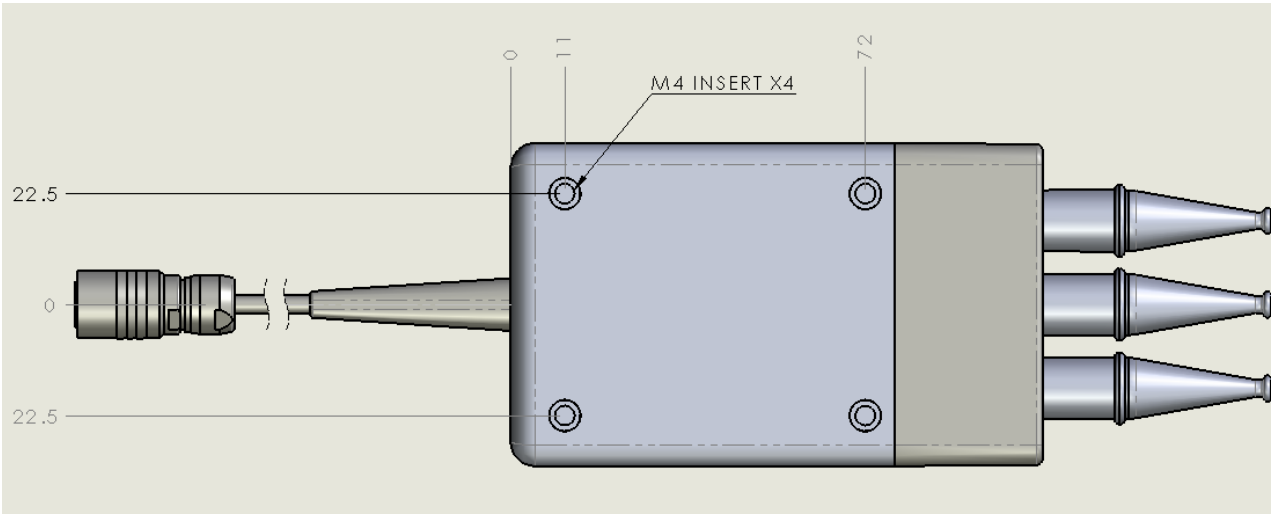


Figure 53 5DOF Junction Box Mounting Dimensions (mm)

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.

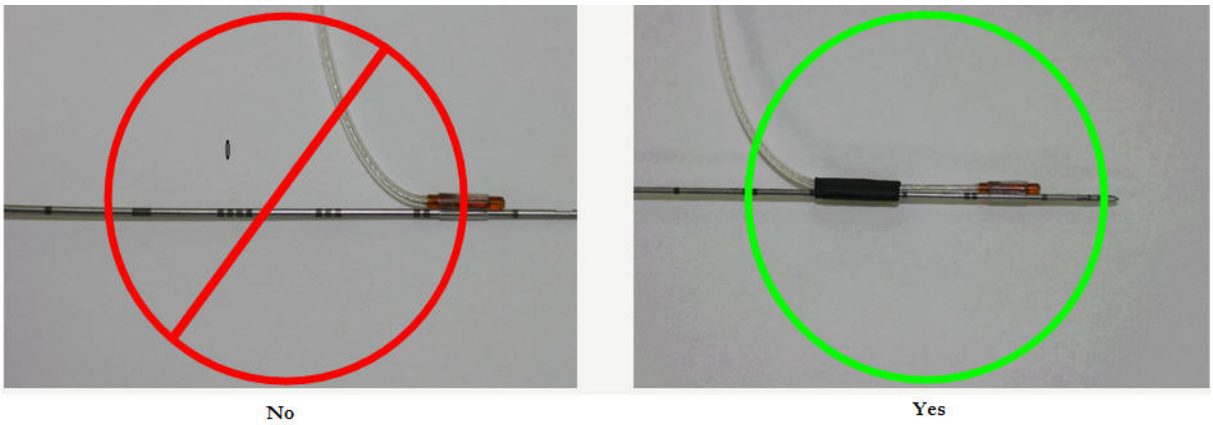


Figure 54 Cable Mounting Requirements

Electronics Unit Mounting

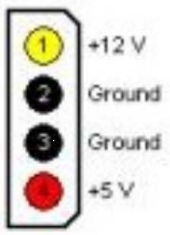
The driveBAY 2 electronics unit is designed for mounting in the optical drive bay of a PC. Be sure to secure the unit with appropriate mounting hardware as recommended by your PC manufacturer, such that a tug on the cables will not allow the unit to pull out and fall on the floor.

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

Rear Panel Connectors

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

Table 4 driveBAY 2 Rear Panel Connections

Name	Description
Power	The power connector is a standard disk drive power connector. Pin 1: +12V: • 1.6A nominal with 4 sensors loaded, 2.9A max Pin 2: Ground Pin 3: Ground Pin 4: +5V: • 1.0A nominal; 1.2 A max 
USB	The USB 2.0 connector is a standard USB TYPE B -Female for peripheral devices. PIN: 1: VCC +5VDC PIN: 2: DATA- PIN: 3: DATA+ PIN: 4: GROUND  B
UNIT SYNC	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.  **Note: Multi-Unit Sync (MUS) not active at this time**

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 driveBAY 2 tracker is designed to work with two basic types of transmitters: Dipole and NonDipole.

Dipole Transmitters

Three Dipole transmitter options are available with driveBAY 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 and miniMAG) 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.

> NOTE: The construction of the MAGnet and miniMAG transmitters 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, 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:

- POServer: 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 II: 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. driveBAY 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 drivEBAY 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 drivEBAY 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. Dimensions of the Performance Motion Box for each transmitter sensor combination are listed below.

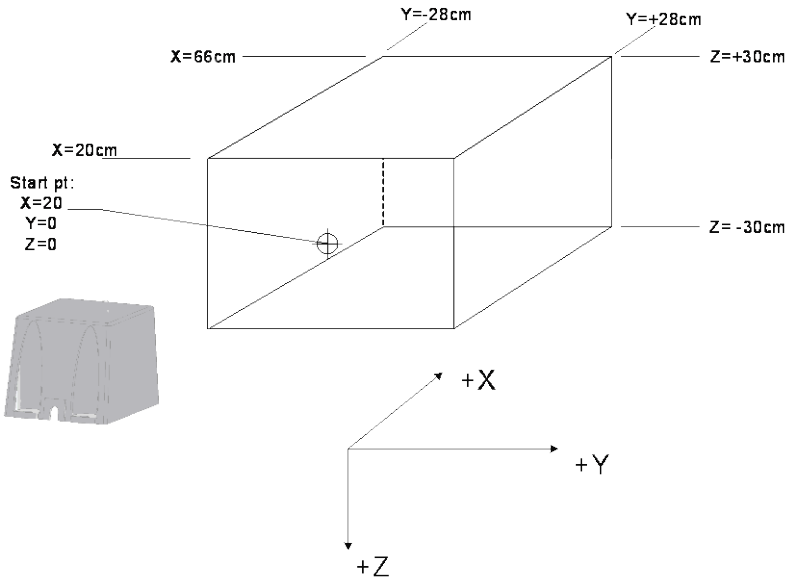
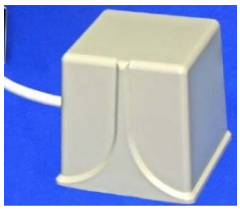
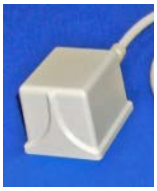


Figure 55 Performance Motion Box – MRT and Model 800 (8 mm) Sensors

Dipole Transmitters

Table 5: Performance Motion Box for Dipole Transmitters

6DOF Sensor Options	DIPOLE TRANSMITTER OPTIONS – FOR 6DOF SENSORS **PERFORMANCE MOTION BOX** (Translation Range for Spec Performance) (Static Accuracy: Position = 1.4mm RMS/Orientation = 0.5° RMS) Volumes are in the FORWARD Hemisphere only Dimensions listed are from the Transmitter Origin (center)	
	MID-RANGE TRANSMITTER MRT 	**SHORT-RANGE TRANSMITTER SRT** 
MODEL 55	X: 16-32cm Y: ±10cm Z: ±10cm *Note –Recommended max range for the Model 55 sensor with the MRT is 25cm (10”)	Not recommended
MODEL 90	X: 20-36cm Y: ±20cm Z: ±20cm *Note –Recommended max range for the Model 90 sensor with the MRT is 36cm (14”)	Not recommended
MODEL 130	X: 20-36cm Y: ±20cm Z: ±20cm	Not recommended
MODEL 180	X: 20-51cm Y: ±23cm Z: ±15cm	Not recommended
MODEL 800	X: 20-66cm Y: ±28cm Z: ±30cm	X: 15-41cm Y: ±12cm Z: ±12cm

NonDipole Transmitters

Table 6: Performance Motion Box for NonDipole Transmitters

	NONDIPOLE TRANSMITTER OPTIONS – FOR 6DOF AND 5DOF SENSORS		
	PERFORMANCE MOTION BOX (Translation Range for Spec Performance) (Static Accuracy: Position = 1.4mm RMS/Orientation = 0.5° RMS)		
6DOF SENSOR OPTIONS	**FLAT 9-COIL TRANSMITTER** **FLAT9** 	**MAGNET 12-COIL TRANSMITTER** **MAGNET** 	**MINIMAG 12-COIL TRANSMITTER** **MINIMAG** 
MODEL 55	10-20 cm off surface ±15 cm other 2 axes (10 cm off-set from surface)	10-20 cm from surface ±10 cm other 2 axes (10 cm off-set from surface)	5-12 cm off surface ±10 cm other 2 axes (5 cm off-set from surface)
MODEL 90	10-30 cm off surface ±15 cm other 2 axes (10 cm off-set from surface)	10-30 cm from surface ±15 cm other 2 axes (10 cm off-set from surface)	5-18 cm from surface ±10 cm other 2 axes (5 cm off-set from surface)
MODEL 130	10-40 cm off surface ±20 cm other 2 axes (10 cm off-set from surface)	10-40 cm from surface, ±20 cm other 2 axes (10 cm off-set from surface)	10-28 cm from surface ±15 cm other 2 axes (10 cm off-set from surface)
MODEL 180	10-40 cm off surface ±20 cm other 2 axes (10 cm off-set from surface)	10-40 cm from surface ±20 cm other 2 axes (10 cm off-set from surface)	10-33 cm from surface ±15 cm other 2 axes (10 cm off-set from surface)
MODEL 800	10-45 cm off surface ±20 cm other 2 axes (10 cm off-set from surface)	10-45 cm from surface ±20 cm other 2 axes (10 cm off-set from surface)	10-38 cm from surface ±15 cm other 2 axes (10 cm off-set from surface)

5DOF SENSOR OPTIONS	**FLAT9**	**MAGNET**	**MINIMAG**
MODEL 55	10-30 cm off surface ±15 cm other 2 axes (10 cm off-set from surface	10-30 cm off surface ±15 cm other 2 axes (10 cm off-set from surface)	10-20 cm off surface ±13 cm other 2 axes (10 cm off-set from surface)
MODEL 90	10-30 cm off surface ±15 cm other 2 axes (10 cm off-set from surface)	10-30 cm off surface ±15 cm other 2 axes (10 cm off-set from surface)	10-20cm off surface ±13 cm other 2 axes (10 cm off-set from surface)
MODEL 130	10-35cm off surface ±20 cm other 2 axes (10 cm off-set from surface)	10-35cm off surface ±20 cm other 2 axes (10 cm off-set from surface)	10-20cm off surface ±13 cm other 2 axes (10 cm off-set from surface)

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 with a given sensor will only be achieved over the motion boxes listed in the Table. The full Motion Box of the transmitter is:

- 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 volumes listed in the Table.

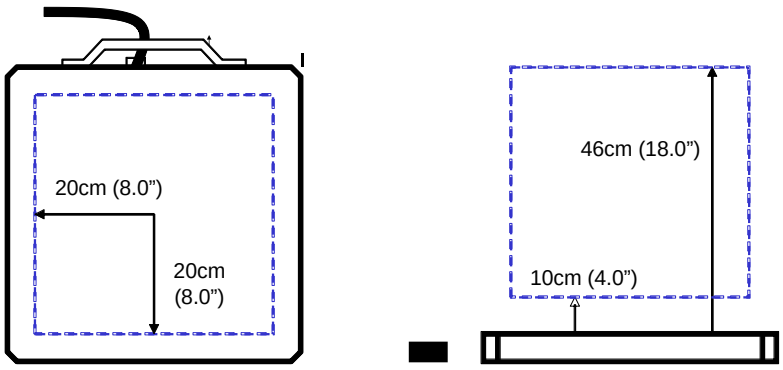


Figure 56 Full Motion Box – Flat Transmitter

Chapter 4

Software Operation - Tools for Successful Tracking

This chapter outlines the interface and method for communicating with the 3DGuidance tracker and lists the sample programs available for the USB interface protocol.

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 7 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)
Cubes.exe	A Demo Windows application (32-bit) that displays tracking data (both test and graphical representation) for all sensors connected to the system. Supports the following functions: • 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 ***NOTE: Cubes is a demo program and should NOT be used when fast data collection is required.***	Program Files\Ascension\3D GuidanceXXXX\Applications\ Cubes.exe
APITest.exe	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: • 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). NOTE: It's important to know how commands are defined and the sequence of commands to send, for this tool to be used effectively.	Program Files\Ascension\3D GuidanceXXXX\Applications\ APITest.exe

Table 8 Location of Sample Utility Programs

Sample Code	Description	Location
Sample	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: • Initializes the Tracker ▪ 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. 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.	Program Files\Ascension3D GuidanceXXXX\API Developer\Samples\ Sample\ Sample.vcproj
Sample2	This C++ project also contains sample code for a very simple console application using the 3DGuidance™ API. It is more comprehensive than "Sample", in that it shows how to use each of the GETXXX/SETXXX calls appropriately. These calls give access to all configurable tracker parameters. An additional feature: also shows command for saving configuration settings to an .ini file. 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.	Program Files\Ascension3D GuidanceXXXX\ API Developer \Samples\ Sample2\Sample2.vcproj

Sample Code	Description	Location
GetSynchronous RecordSample	This C++ project is a version of the Sample project described above, which has been modified to collect as much data from the tracker as possible using the GetSynchronousRecord() call. Specifically, it: • 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. TX Off - Before closing the application, shows how to turn off the TX 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.	Program Files\Ascension\3D GuidanceXXXX\ API Developer \Samples\ GetSynchronousRecord Sample\ GetSynchronousRecordSample.vcproj

Chapter 5

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

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)
- Auxiliary Port: None selected
- Commutation Mode Disabled
- Communications USB
- Reset on initialization True (resets tracker to a known state)
- Logging False (log file not generated)
- Auto Configuration 4 (4 sensors)

Note: Transmitter AGC not used in 3DGuidance systems.

- **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.

Note: Transmitter AGC not used in 3DGuidance systems.

- **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 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.

- **AUXILIARY PORT**

For 3DGuidance trackers equipped with one or more auxiliary sensor ports, this system parameter allows selection between the front panel port and its paired auxiliary port. Only one or the other can be selected at a time. All sensor operations and data requests will be referred to the selected port. In Multi-Unit Sync configurations, auxiliary ports can only be selected for the first board.

- **COMMUTATION MODE**

Integrated Sensor Modules (i.e. M1525) available for oem custom installations of the 3DGuidance tracker require this alternate data acquisition mode for optimum operation. Note that this system parameter should only be enabled when sensors of this type are in use, as the system's sensitivity to conductive metal will increase slightly when this mode is being used.

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 senor data to eliminate signals with a frequency between 30 and 72 Hz. Note: for this filter to work properly the system parameter: POWER_LINE_FREQUENCY must be correctly initialized using the SetSystemParameter function call.
- **Sudden Output Change Filter**
  - If 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.

Note: Sudden Output filter not implemented at time of manual writing.

- **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 driveBAY2 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 for details.

- **VITAL PRODUCT DATA STORAGE**

  Note: VPD storage not available in Junction Box 5DOF sensors

  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 overfill 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, a dipole system will transmit 80 *3 =240 Updates/sec (1 record every 4mS) for each sensor. Changing the REPORT_RATE setting to 4 will reduce the number of records to 240/4 = 60 Updates/sec (1 record every 17mS) for each sensor. The default REPORT_RATE setting of 1 makes all outputs computed by the tracker available. See the Configurable Settings section in Chapter 3 for details on changing the default setting.

  As with the GetAsynchronous call, a record containing data from all sensors can be obtained by setting the sensor ID to ALL_SENSORS. For legacy tracker users, this is the equivalent of GROUP Mode.

BOARD

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 tranmsitter. 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 priviledges.

- **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 InitializeBIRDSystem() function call. The Media Type is set to USB by default.

- **LOGGING**

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

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

- **RESET**

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

  NOTE: This parameter can be used prior to a InitializeBIRDSystem() 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 needed with 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. See the AUTOCONFIG section in Chapter 3 for additional details.

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
      REFERENCE_FRAME,        // command parameter type
      &frame,                 // address of data source buffer
      sizeof(frame)           // size of source buffer
      );
if(errorCode!=BIRD_ERROR_SUCCESS) errorHandler(errorCode);
      // user must provide an error handler

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

Acquiring Tracking Data

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 reponse 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 3DGuidance 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

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 n
);
```

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 *bufferSize* 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 *bufferSize* 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 transmitterrIDs 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 *bufferSize* 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 *bufferSize* 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 then the size of the parameter being passed or the function will read beyond the end of the buffer into user memory with indeterminate results.

bufferSize

[in] Contains the size of the buffer whose address is passed in *pBuffer*. Note the *bufferSize* value must match the size 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 *bufferSize* 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 3DGuidance system.

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

Parameters

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

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

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

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

Return Values

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

Value	Meaning
BIRD_ERROR_SUCCESS	No errors occurred. Call completed successfully
BIRD_ERROR_SYSTEM_UNINITIALIZED	The 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 *bufferSize* 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 then the size of the parameter being passed or the function may read beyond the end of the buffer into user memory with indeterminate results.

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

Return Values

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

Value	Meaning
BIRD_ERROR_SUCCESS	No errors occurred. Call completed successfully
BIRD_ERROR_SYSTEM_UNINITIALIZED	The 3DGuidance hardware and system has not been initialized yet. The InitializeBIRDSystem function must be called first.
BIRD_ERROR_INCORRECT_PARAMETER_SIZE	The value of the *bufferSize* 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 *bufferSize* 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 *recordSize* 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 *recordSize* 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 overfill 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 tracker will transmit 80 *3 =240 Updates/sec (1 record every 4mS) for each sensor. Changing the Report Rate setting to 4 will reduce the number of records to 240/4 = 60 Updates/sec (1 record every 17mS) for each sensor. The default Report Rate setting of 1 makes all outputs computed by the tracker available. See the 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 *errorCode*.

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	The system has detected that the connected sensor does not match the port configuration (i.e. 5DOF sensor in 6DOF port).				
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. This set in Non-Dipole configurations when the algorithm is lost.			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 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	The system has detected that the connected sensor does not match the port configuration (i.e. 5DOF sensor in 6DOF port).				
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. This set in Non-Dipole configurations when the algorithm is lost.			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 InitializeBIRDSystem has been called
2) Calling the function with an invalid transmitter ID.

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

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

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

Requirements

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 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	The system has detected that the connected sensor does not match the port configuration (i.e. 5DOF sensor in 6DOF port).				
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	AIGORITHM_INITIALIZING	The sensor has not acquired enough raw magnetic data to compute an accurate P&O solution. This set in Non-Dipole configurations when the algorithm is lost.			x	

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 InitializeBIRDSystem has been called

2) Calling the function with an invalid board ID.

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

1) If the function InitializeBIRDSystem has not been called then the “UnInitialized” bit will be set.

2) If this function call was made with an invalid (out of range) board ID then the “Non-Existent” bit will be set.

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

Requirements

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	The system has detected that the connected sensor does not match the port configuration (i.e. 5DOF sensor in 6DOF port).				
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. This set in Non-Dipole configurations when the algorithm is lost.			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 InitializeBIRDSystemfunction 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 SaveSystemConfiguration() to automatically save a correctly formatted initialization file.
BIRD_ERROR_MISMATCHED_DATA	Data item in the initialization file does not match a system parameter. For example the initialization file states the system has 3 boards (NumberOfBoards=3) but the system initialization routine – InitializeBIRDSystem() only detected two.
BIRD_ERROR_CONFIG_INTERNAL	Internal error in configuration file handler. Report to vendor.

Remarks

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

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

NOTE: if the full pathname specification is not provided then the default search path is in the 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.

Data record structures:

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

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 model string of the board in an 11 byte NULL terminated character string. For example the driveBAY/trakSTAR systems supporting 6DOF operation will have the string "6DBB4 ". 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

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/Setsystem parameter will only access those diagnostic tests on the first unit (Multi-Unit ID =0). The DIAGNOSTIC_TEST_PCB board parameter type should be used with the Get/SetBoardParameter() calls, for accessing these tests across multiple units.

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

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

```
typedef struct tagDIAGNOSTIC_TEST_PARAMETERS{
   UCHAR             suite;
   UCHAR             test;
   UCHAR             suites;
   UCHAR             tests;
   PUCHAR      *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 9.

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 9.

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**
GetSystem Parameter ()	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.
GetSystem Parameter ()	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.
GetSystem Parameter ()	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.
SetSystem Parameter ()	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. *
SetSystem Parameter ()	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. *
SetSystem Parameter ()	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 9 Diagnostic Test Usage

* NOTE: A character string describing the error condition can be obtained by calling GetErrorText() 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 priviledges (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

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.

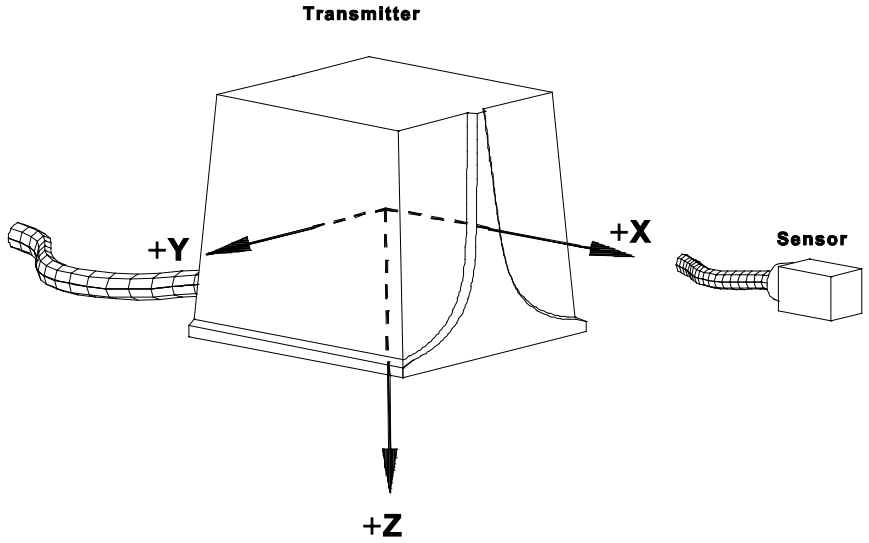


Figure 57 : Measurement Reference Frame (Mid-Range Transmitter)

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:

```
Angle = (signed int * 180) / 32768;
```

Requirements

Windows NT/2000: Requires Windows 2000 or later.
Header: Declared in ATC3DG.h
Library: Use ATC3DG.lib

See Also

SHORT_MATRIX_RECORD

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

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

Members

s[3][3]

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

Remarks

The MATRIX mode outputs the 9 elements of the rotation matrix that **define the orientation of the sensor's X, Y, and Z axes with respect to the transmitter's X, Y, and Z axes**.

Each row of the matrix describes the orientation of one of the sensor's axis in the transmitter's frame of reference. So if the first row of the matrix was [.75, .10, .66], it would mean that in the transmitter's coordinate frame, the sensor's x-axis is a vector in the direction <.75, .10, .66>. The second row is the sensor's y-axis, the third row is the z-axis. Each row (and column) will have a length of 1. Each row will be orthogonal (90 degree difference) to every other row (and every column will be orthogonal to every other column).

In graphics applications the rotation matrix is used to rotate the 3-D coordinates of the item being drawn. To do this, **the transpose of the Ascension matrix is used (the rows become the columns)**.

Say you want your 3D model to follow the orientation of the sensor. Multiply the transpose of the Ascension rotation matrix by the 3D position of each point in the model. newPt = M.transpose() * pt. Where pt is a 3x1 vector containing the object's point.

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

M(1,1)	**M(1,2)**	**M(1,3)**
M(2,1)	**M(2,2)**	**M(2,3)**
M(3,1)	**M(3,2)**	**M(3,3)**

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

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

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

COS(E)*COS(A)	COS(E)*SIN(A)	-SIN(E)
-(COS(R)*SIN(A)) +(SIN(R)*SIN(E)*COS(A))	(COS(R)*COS(A)) +(SIN(R)*SIN(E)*SIN(A))	SIN(R)*COS(E)
(SIN(R)*SIN(A)) +(COS(R)*SIN(E)*COS(A))	-(SIN(R)*COS(A)) +(COS(R)*SIN(E)*SIN(A))	COS(R)*COS(E)

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

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

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.

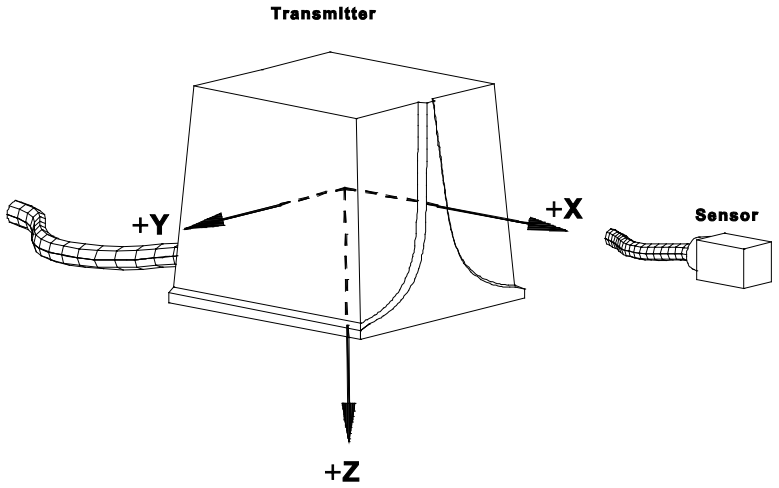


Figure 58 Measurement Reference Frame (Mid-Range Transmitter)

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**.

Each row of the matrix describes the orientation of one of the sensor's axis in the transmitter's frame of reference. So if the first row of the matrix was [.75, .10, .66], it would mean that in the transmitter's coordinate frame, the sensor's x-axis is a vector in the direction <.75, .10, .66>. The second row is the sensor's y-axis, the third row is the z-axis. Each row (and column) will have a length of 1. Each row will be orthogonal (90 degree difference) to every other row (and every column will be orthogonal to every other column).

In graphics applications the rotation matrix is used to rotate the 3-D coordinates of the item being drawn. To do this, **the transpose of the Ascension matrix is used (the rows become the columns)**.

Say you want your 3D model to follow the orientation of the sensor. Multiply the transpose of the Ascension rotation matrix by the 3D position of each point in the model. newPt = M.transpose() * pt. Where pt is a 3x1 vector containing the object's point.

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

M(1,1)	**M(1,2)**	**M(1,3)**
M(2,1)	**M(2,2)**	**M(2,3)**
M(3,1)	**M(3,2)**	**M(3,3)**

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

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

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

COS(E)*COS(A)	**COS(E)*SIN(A)**	**-SIN(E)**
-(COS(R)*SIN(A)) +(SIN(R)*SIN(E)*COS(A))	**(COS(R)*COS(A)) +(SIN(R)*SIN(E)*SIN(A))**	**SIN(R)*COS(E)**
(SIN(R)*SIN(A)) +(COS(R)*SIN(E)*COS(A))	**-(SIN(R)*COS(A)) +(COS(R)*SIN(E)*SIN(A))**	**COS(R)*COS(E)**

The matrix elements take values between the binary equivalents of +.99996 and -1.0. Element scaling is +.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 scaler component, have been extracted from the MATRIX output using the algorithm described in "Quaternion from Rotation Matrix" by Stanley W. Sheppard, Journal of Guidance and Control, Vol. 1, May-June 1978, pp. 223-4.

Requirements

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 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_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 (where applicable - 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 (where applicable - 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 3DGuidance trackers

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 InitializeBIRDSystem 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: 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 – InitializeBIRDSystem() only detected two.
BIRD_ERROR_CONFIG_INTERNAL	Internal error in configuration file handler. Report to

	vendor.
BIRD_ERROR_UNRECOGNIZED_MODEL_STRING	The firmware is reporting a model string which is unrecognized by the API dll. This could be due to a hardware failure causing the model string data to be corrupted or it may be caused by a corrupted board EEProm or the board installed is of a type not recognized by the API dll. If the error is repeatable return to vendor.
BIRD_ERROR_INCORRECT_SENSOR	An invalid sensor type has been attached to this card.
BIRD_ERROR_INCORRECT_TRANSMITTER	An invalid transmitter type has been attached to this card.
BIRD_ERROR_ALGORITHM_INITIALIZATION	The position and orientation algorithm used for the non-dipole transmitters has not correctly initialized.
BIRD_ERROR_LOST_CONNECTION	USB connection to the tracker has been lost. This may be due to usb cable removal, or removal of power to the electronics unit.
BIRD_ERROR_INVALID_CONFIGURATION	The system has queried the Multi-Unit ID settings for each of the attached electronics units, and found that the current configuration is not supported. Valid MUS configurations include: 0,1 (8 sensors), 0,1,2 (12 sensors), 0,1,2,3 (16 sensors). Further details can be found in the MUS Installation Guide Addendum.
BIRD_ERROR_TRANSMITTER_RUNNING	A request to read or write to the VPD component memory storage area was made while the transmitter was still running. De-select (turn off) the transmitter prior to a read/write to VPD memory.
BIRD_ERROR_MAXIMUM_VALUE	Final error code place holder

Requirements

Windows NT/2000: Requires Windows 2000 or later.
Header: Declared in ATC3DG.h
Library: Use ATC3DG.lib

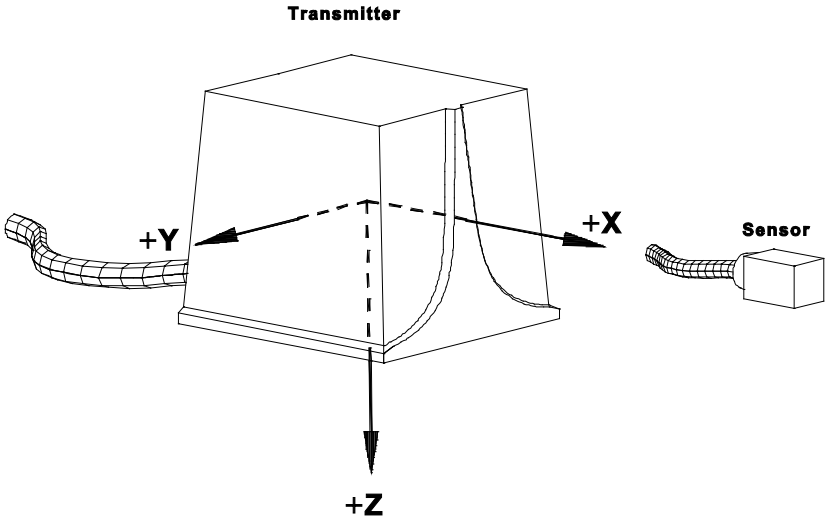
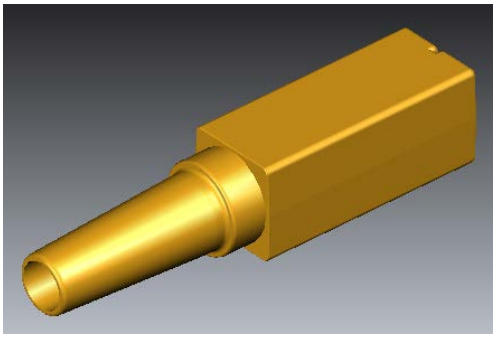
See Also

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,
   PORT_CONFIGURATION
};
```

Enumerator Value	Meaning
DATA_FORMAT	See the Data Format Structures section for details
ANGLE_ALIGN	By default, the angle outputs from the Tracker are measured in the coordinate frame defined by the transmitter's X, Y and Z axes, as shown in Figure 59 and are measured with respect to rotations about the physical X, Y and Z axes of the sensor Figure 60. The ANGLE ALIGN parameter allows you to mathematically change the sensor's X, Y and Z axes to an orientation which differs from that of the actual sensor. For example: Suppose that during installation you find it necessary, due to physical requirements, to rotate the sensor, resulting in its angle outputs reading Azim = 5 deg, Elev = 10 and Roll = 15 when it is in its normal "resting" position. To compensate, use the ANGLE_ALIGN parameter, passing as values 5, 10 and 15 degrees. After this sequence is sent, the sensor outputs will be zero, and orientations will be computed as if the sensor were not misaligned. ANGLE_ALIGN parameters are not meaningful for 5DOF sensors. ANGLE_ALIGN parameters applied to a 5DOF sensor will be ignored.

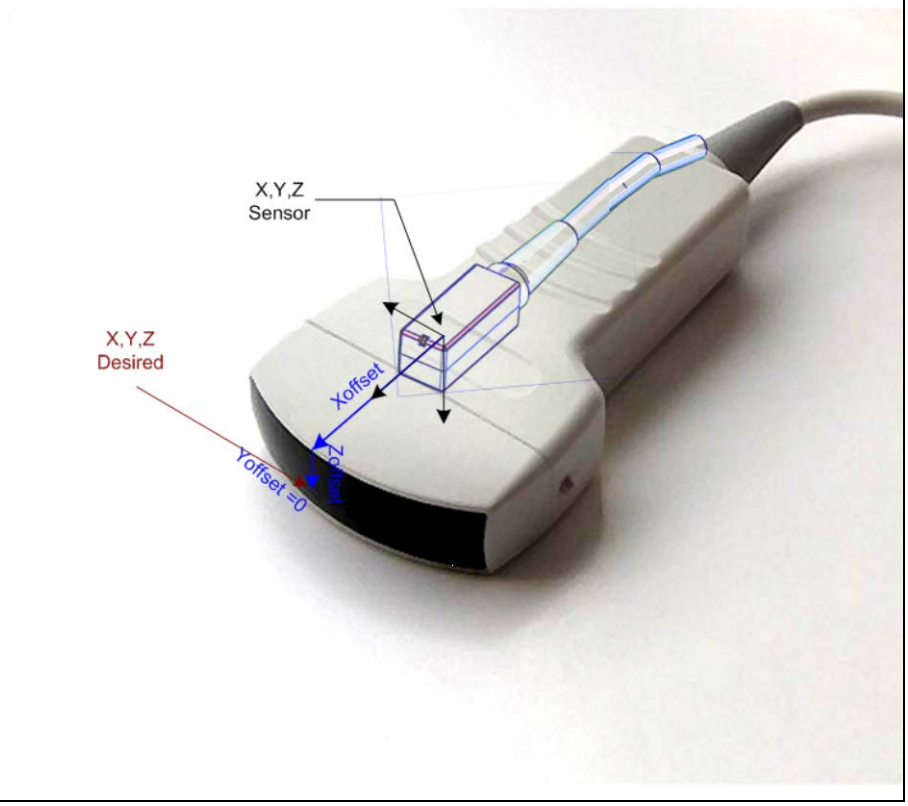
ANGLE_ALIGN, continued	 **Figure 59 Measurement Reference Frame**  **Figure 60 Receiver Zero Orientation (8mm Sensor)**
HEMISPHERE	The shape of the magnetic field transmitted by the Tracker is symmetrical about each of the axes of the transmitter. This symmetry leads to an ambiguity in determining the sensor's X, Y, Z position. The amplitudes will always be correct, but the signs (±) 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 (±) of the X, Y, and Z position outputs when the sensor crosses a hemisphere boundary. When you select a given

	hemisphere of operation, the sign on the position axes that defines the hemisphere direction is forced to positive, even when the sensor moves into another hemisphere. For example, the power-up default hemisphere is the front hemisphere. This forces X position outputs to always be positive. The signs on Y and Z will vary between plus and minus depending on where you are within this hemisphere. If you had selected the bottom hemisphere, the sign of Z would always be positive and the signs on X and Y will vary between plus and minus. If you had selected the left hemisphere, the sign of Y would always be negative, etc. Regarding the default front hemisphere, if the sensor moved into the back hemisphere, the signs on Y and Z would instantaneously change to opposite polarities while the sign on X remained positive. To 'track' the sensor, your host software, on detecting this sign change, would reverse the signs on The Tracker's X, Y, and Z outputs. In order to 'track' correctly: You must start 'tracking' in the selected hemisphere so that the signs on the outputs are initially correct, and you must guard against the case where the sensor legally crossed the Y = 0, Z = 0 axes simultaneously without having crossed the X = 0 axes into the other hemisphere.
FILTER_AC_WIDE_NOTCH	The AC WIDE notch filter refers to a eight tap FIR notch filter that is applied to the sensor data to eliminate sinusoidal signals with a frequency between 30 and 72 hertz. If your application requires minimum transport delay between measurement of the sensor's position/orientation and the output of these measurements, you may want to evaluate the effect on your application with this filter shut off and the AC NARROW notch filter on.
FILTER_AC_NARROW_NOTCH	The AC NARROW notch filter refers to a two tap finite impulse response (FIR) notch filter that is applied to signals measured by the Tracker's sensor to eliminate a narrow band of noise with sinusoidal characteristics. Use this filter in place of the AC WIDE notch filter when you want to minimize the transport delay between Tracker measurement of the sensor's position/orientation and the output of these measurements. The transport delay of the AC NARROW notch filter is approximately one third the delay of the AC WIDE notch filter.

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

	too noisy during rapid sensor movement. The 7 words that make up the Vm table values represent the expected noise that the DC filter will measure. By changing the table values, you can increase or decrease the DC filter's lag as a function of sensor range from the transmitter. The DC filter is adaptive in that it tries to reduce the amount of low pass filtering in the Tracker as it detects translation or rotation rates in the Tracker's sensor. Reducing the amount of filtering results in less filter lag. Unfortunately, electrical noise in the environment—when measured by the sensor—also makes it look like the sensor is undergoing a translation and rotation. As the sensor moves farther and farther away from the transmitter, the amount of noise measured by the sensor appears to increase because the measured transmitted signal level is decreasing and the sensor amplifier gain is increasing. In order to decide if the amount of filtering should be reduced, the Tracker has to know if the measured rate is a real sensor rate due to movement or a false rate due to noise. The Tracker gets this knowledge by the user specifying what the expected noise levels are in the operating environment as a function of distance from the transmitter. These noise levels are the 7 words that form the Vm table. The Vm values can range from 1 for almost no noise to 32767 for a lot of noise.
FILTER_LARGE_CHANGE	When the LARGE_CHANGE filter is selected, the position and orientation outputs are not allowed to change if the system detects a sudden large change in the outputs. Large undesirable changes may occur at large separation distances between the transmitter and sensor when the sensor undergoes a fast rotation or translation. If the LARGE_CHANGE value is TRUE the outputs will not be updated if a large change is detected. If value is FALSE, the outputs will change.
QUALITY QUALITY –cont	This data structure is used to adjust the output characteristics of the Quality number. Also referred to as the METAL error or Distortion number, this value is returned with certain data formats and gives the user an indication of the degree to which the position and angle measurements are in error. This error may be due to 'bad' metals located near the transmitter and sensor, or due to TRACKER 'system' errors. 'Bad' metals are metals with high electrical conductivity such as aluminum, or high magnetic permeability such as steel. 'Good' metals have low conductivity and low permeability such as 300 series stainless steel, or titanium. The QUALITYerror number also reflects TRACKER 'system' errors resulting from accuracy degradations in the transmitter, sensor, or other electronic components. It will represent a level of accuracy degradation resulting from either movement of the sensor or environmental noise. A very small QUALITYerror number indicates no or minimal position and angle errors depending on how sensitive you have set the error indicator. A large QUALITYerror number indicates maximum error for the sensitivity level selected. Users of the QUALITYerror number will find that as a metal detector, it is sensitive to the introduction of metals in an environment where no metals were initially present. This metal detector can fool you, however, if there are some metals initially present and you introduce new metals. It is possible for the new metal to cause a distortion in the magnetic field that reduces the existing distortion at the sensor. When this occurs you'll see the Error value initially decrease, indicating less error, and then finally start increasing again as the new metal causes more distortion. **User beware. You need to evaluate your application for suitability of this metal detector.** Because the TRACKER is used in many different applications and environments, the QUALITYerror indicator needs to be sensitive to this broad range of environments. Some users may want the error indicator to be sensitive to very small amounts of metal in the environment while other applications may only want the error indicator sensitive to large amounts of metal. To accommodate this range of detection sensitivity, the SetSensorParameter() allows the user to set a QUALITY Sensitivity setting that is appropriate to their application. The QUALITYerror number will always show there is some error in the system

QUALITY –cont	even when there are no metals present. This error indication usually increases as the distance between the transmitter and sensor increases, and is due to the fact that TRACKER components cannot be made or calibrated perfectly. To minimize the amount of this inherent error in the Error value, a linear curve fit, defined by a **slope** and **offset**, is made to this inherent error and stored in each individual sensor's memory since the error depends primarily on the size of the sensor being used (25mm, 8mm, or 5 mm). The Quality Parameters Structure (manipulated through the SetSensorParameter() command) allows the user to eliminate or change these values. For example, maybe the user's standard environment has large errors and he or she wants to look at variations from this standard environment. To do this he or she would adjust the Slope and Offset settings to minimize the QUALITYerror values. The QUALITYerror number that is output is computed from the following equation: QUALITYerror = Sensitivity x (ErrorSYSTEM – (Offset + Slope x Range)) Where Range is the distance between the transmitter and sensor. **Sensitivity** The user supplies a Sensitivity value based on how little or how much he or she wants the QUALITYerror value to reflect errors. **Offset** If you are trying to minimize the base errors in the system by adjusting the Offset you could set the Sensitivity =1, and the Slope=0 and read the Offset directly as the QUALITYerror number. **Slope** You can determine the slope by setting the Sensitivity=1 and looking at the change in QUALITYerror as you translate the sensor from range=0 to range max for the system (ie 36"). Since its difficult to go from range =0 to max., you might just translate over say half the distance and double the error value change you measure. **Alpha** The QUALITYerror value is filtered before output to the user to minimize noise jitter. The Alpha value determines how much filtering is applied to the error value. Range is FFFF -> 0 Users should think of it as a signed fractional value with a range of 0.9999 -> 0 (negative numbers not allowed). A zero value is an infinite amount of filtering, whereas a 0.9999 value is no filtering. As Alpha gets smaller the time lag between the insertion of metal in the environment and it being reported in the QUALITYerror value increases.

SERIAL_NUMBER_RX	Returns the serial number of the attached sensor.
SENSOR_OFFSETS	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. 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. 

VITAL_PRODUCT_DATA_RX	Used to read or write to individual bytes in the Vital Product Data (VPD) storage area on the sensor. The VPD section comprises 128 bytes of user modifiable data storage. It is the user's responsibility to define the contents and structure and to maintain that structure. The parameter passed with these commands is a structure VPD_COMMAND_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. Note: Reading or writing VPD is not allowed when the transmitter is running.
VITAL_PRODUCT_DATA_PREAMP	Used to read or write to individual bytes in the Vital Product Data (VPD) storage area on the preamp. The VPD section comprises 128 bytes of user modifiable data storage. It is the user's responsibility to define the contents and structure and to maintain that structure. The parameter passed with these commands is a structure VPD_COMMAND_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. Note: Reading or writing VPD is not allowed when the transmitter is running.
MODEL_STRING_RX	Returns the model string of the sensor in an 11 byte NULL terminated character string.
PART_NUMBER_RX	Returns the part number of the sensor in an 16 byte NULL terminated character string.
MODEL_STRING_PREAMP	Returns the model string of the preamp in an 11 byte NULL terminated character string. (driveBAY and trakSTAR units do not have preamps, therefore this is an invalid parameter for those systems)
PART_NUMBER_PREAMP	Returns the part number of the preamp in an 16 byte NULL terminated character string. (driveBAY and trakSTAR units do not have preamps, therefore this is an invalid parameter for those systems)
PORT_CONFIGURATION	Returns a value of PORT_CONFIGURATION_TYPE containing the current configuration of this sensor port. DOF_6_XYZ_AER and DOF_5_XYZ_AE indicate 6DOF and 5DOF ports, respectively. DOF_NOT_ACTIVE indicates that the port is not logically active because the physical port has been configured for 6DOF operation. For example, if Sensor Port 1 on the unit is configured for '6DOF', and the PORT_CONFIGURATION of Sensor 5 (ID=4) or 9 (ID=8) is requested, the return value will be 'DOF_NOT_ACTIVE'. PORT_CONFIGURATION is currently determined at startup only. Calling SetSensorParameter with the PORT_CONFIGURATION parameter will return BIRD_ERROR_ILLEGAL_COMMAND_PARAMETER.

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 GetErrorText will return a short terse message string describing the meaning of the error code passed.
VERBOSE_MESSAGE	The call GetErrorText will return a message string containing a full and comprehensive description of the problem and possible resolutions where appropriate.

Requirements

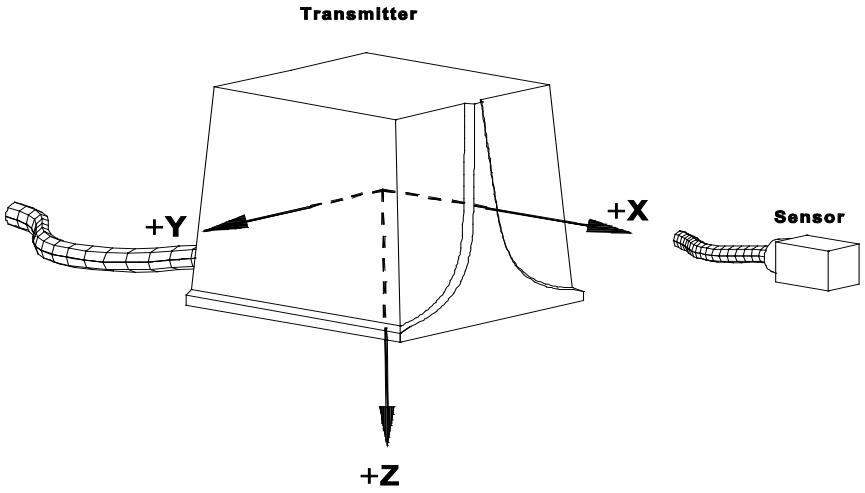
Windows NT/2000: Requires Windows 2000 or later.
Header: Declared in ATC3DG.h
Library: Use ATC3DG.lib

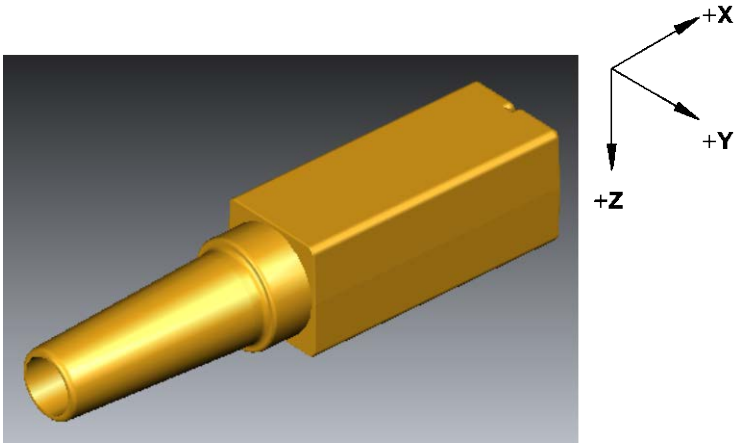
See Also

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	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. 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.  **Figure 61 Measurement Reference Frame (Standard Transmitter)**

	 **Figure 62 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,
   AUXILIARY_PORT,
   COMMUTATION_MODE,
   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 AGC_MODE_TYPE.
MEASUREMENT_RATE	Set the measurement rate. The parameter passed is a double value and represents the measurement rate in Hz. The valid range of values is 20.0<rate<255.0
MAXIMUM_RANGE	Sets the system maximum range. The parameter passed is a double value representing the maximum range in any of the 3 axes in inches. There are three valid ranges, 36.0 inches, 72.0 inches and 144.0 inches.
METRIC	Enables/disables metric position reporting. The parameter passed is a bool. If the value is true then metric reporting is selected otherwise if the value is false then metric reporting is turned off. Metric data is reported in millimeters. Non-metric data is reported in inches.
VITAL_PRODUCT_DATA*	Used to read or write to individual bytes in the Vital Product Data (VPD) storage area on the main board. The VPD contains 512 bytes of user modifiable data storage. It is the user's responsibility to define the contents and structure and to maintain that structure. The parameter passed with these commands is a structure 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 needed with 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. See the AUTOCONFIG section in Chapter 3 for additional details.
AUXILIARY_PORT	Used to select between front panel and auxiliary ports on equipped trackers. The parameter is a pointer to the first element of a 4-element array of type AUXILIARY_PORT_TYPE.
COMMUTATION_MODE	Used to enable/disable the alternate data acquisition mode required when running integrated sensor modules (i.e. M1525). NOTE: This parameter should only be enabled when sensors of this type are connectede, as the system's sensitivity to conductive metal will increase slightly when this mode is being used.
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/trakSTAR sensor (bayBIRD)
TYPE_800_BB_STD_TRANSMITTER	DriveBAY/trakSTAR mid-range TX
TYPE_800_BB_SMALL_TRANSMITTER	DriveBAY/trakSTAR short-range TX
TYPE_090_BB_SENSOR	DriveBAY/trakSTAR sensor (bayBIRD)

Requirements

Windows NT/2000: Requires Windows 2000 or later.
Header: Declared in ATC3DG.h
Library: Use ATC3DG.lib

COMMUNICATIONS_MEDIA_TYPES

A value of the **COMMUNCIATIONS_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 GetSynchronous API is not supported with this interface and GetAsynchronous API cannot be used with the ALL_SENSORS 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

PORT_CONFIGURATION_TYPE

The **PORT_CONFIGURATION_TYPE** enumeration type defines values that are used when getting the PORT_CONFIGURATION with a GetSensorParameter call.

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

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

Requirements

Windows NT/2000: Requires Windows 2000 or later.
Header: Declared in ATC3DG.h
Library: Use ATC3DG.lib

See Also

AUXILIARY_PORT_TYPE

The **AUXILIARY_PORT_TYPE** enumeration type defines values that are used when getting or setting the AUXILIARY_PORT with a GetSystemParameter or SetSystemParameter call.

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

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

Requirements

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 3DGuidance tracker

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 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	The system has detected that the connected sensor does not match the port configuration (i.e. 5DOF sensor in 6DOF port).				
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. This set in Non-Dipole configurations when the algorithm is lost.			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 3DGuidance tracker 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
```

[Sensor**x**]

The **Sensor** section is optional.

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

Format-type
This parameter takes the form of the DATA_FORMAT_TYPE enumerated constant listed in the ATC3DG.h file. Use the exact spelling and case as found in the header file.

Hemisphere-type
This parameter takes the form of the HEMISPHERE_TYPE enumerated constant listed in the ATC3DG.h file. Use the exact spelling and case as found in the header file.

Narrow-flag
This parameter is a Boolean and is selected by entering either yes or no.

Wide-flag
This parameter is a Boolean and is selected by entering either yes or no.

dc-flag
This parameter is a Boolean and is selected by entering either yes or no.

Min-params
These parameters are entered as a sequence of 6 comma separated floating point numbers in the range 0 to +1.0. Note: A Min_param cannot exceed its equivalent Max_param in value.

max-params
These parameters are entered in the same format as the Min_params. Note a Max_param may never have a value lower than its equivalent Min_param.

vm-params
These parameters are entered as 6 comma-separated integers. The valid range for the integers is from a minimum of 1 to a maximum of 32767.

Align-angles
These parameters are entered as 3 comma-separated floating point values. The parameters represent azimuth, elevation and roll. The azimuth and roll values must lie with the range –180 to +180 degrees and the elevation value must lie within the range –90 to +90 degrees.

Change-flag
This parameter is a Boolean and is selected by entering either yes or no.

Distortion-params
These parameters are entered as 4 comma-separated integers. The 4 values are defined as follows: error-slope, error-offset, error-sensitivity and filter-alpha. The slope should have a value between –127 and +127. (Default is 0) The offset should have a value between –127 and +127. (The default is 0) The sensitivity should have a value between 0 and +127 (Default is 2) and the alpha should have a value between 0 and 127. (The default is 12)

The following example shows a typical **Sensor** section from a configuration file:

```
[Sensor2]
Format=SHORT_POSITION_ANGLES
Hemisphere=FRONT
AC_Narrow_Filter=no
AC_Wide_Filter=yes
DC_Filter=yes
Alpha_Min=0.02,0.02,0.02,0.02,0.02,0.02
Alpha_Max=0.09,0.09,0.09,0.09,0.09,0.09
Vm=2,4,8,32,64,256,512
Angle_Align=0,0,0
Filter_Large_Change=NO
Distortion=164,0,32,327
```

[Transmitter**x**]

The **Transmitter** section is optional.

```
[Transmitterx]
XYZ_Reference=reference-flag
XYZ_Reference_Angles=reference-angles
```

Reference-flag

This parameter is a Boolean and should be entered as yes or no. If yes is selected a new sensor position will be calculated for the new reference frame defined by the reference frame angles.

Reference-angles

These parameters take the form a sequence of 3 comma-separated floating point values which represent the azimuth, elevation and roll of the new transmitter reference frame. The azimuth and roll must have values in the range –180 to +180 degrees. The elevation value must be in the range –90 to +90 degrees.

The following example shows a typical **Transmitter** section from a configuration file:

```
[Transmitter1]
XYZ_Reference=no
XYZ_Reference_Angles=0,45,0
```

Chapter 6

Troubleshooting, Maintenance, and Repair

Taking care of your driveBAY 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

driveBAY 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 USB and power connections to ensure positive contact.
2. Inspect to ensure driveBAY 2 is tightly seated into the drive bay slot.
3. 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 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 driveBAY 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 I, 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 driveBAY 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 10 Troubleshooting Symptons, Possible Causes, and Solutions

Symptom	Possible Causes	Solution
No front panel LED illumination	No Power	Check AC connections to power supply Reset hardware by cycling AC power
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
Cubes Demo says 'Invalid Device'	Sensor port of the electronics unit not configured to run the attached sensor type.	Run the driveBAY2 Utility and check the Sensor port configuration. Follow the steps outlined in Configuring for 5DOF Oepration.
Cubes Demo says 'Lost'	Position Algorithm for NonDipole transmitter has not converged on a solution Sensor is outside the calibrated volume of the Non-Dipole transmitter	Check the Performance Motion Box for the sensor/transmitter combination you are running, and ensure that the sensor is within this volume. Rotate the sensor and move it toward the transmitter surface. Consider changing to a Manual Starting Pose in power-up default settings. See Appendix III for details.
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 driveBAY 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 driveBAY 2, and re-check the settings

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 driveBAY. -Cycle power on the driveBAY -Check installation/status of driveBAY 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. 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 driveBAY 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.

LED Definitions

Table 11 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 <u>is 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

There are **no user serviceable parts** inside the driveBAY 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.

Accessories List

6DOF Transmitter Options	
600214-O	**3DG Short-Range Transmitter (SRT) for driveBAY 2/trakSTAR 2** • Cable Length: 10.9 ft. (3.3 m)
600222-K	**3DG Mid-Range Transmitter (MRT) for driveBAY 2/trakSTAR 2** • Cable Length: 10.9 ft. (3.3 m)

5DOF/6DOF Transmitter Options	
600805-9C	**3DG Flat Transmitter for driveBAY 2/trakSTAR 2** • Cable Length: 10.9 ft. (3.3 m)
600806	**3DG MAGnet Transmitter for driveBAY 2/trakSTAR 2** • Cable Length: 10.9 ft. (3.3 m)
600808	**3DG miniMAG Transmitter for driveBAY 2/trakSTAR 2** • Cable Length: 10.9 ft. (3.3 m)

6DOF Sensor Options		
600865-GEN	**3DG Model 55 6DOF Sensor for driveBAY/trakSTAR**	
	• Sensor OD = 0.56 mm • Sensor Length = 80-210 mm	• Cable OD = 3.8 mm • Cable Length = <8.2 ft. [2.5 m]
600855	**3DG Model 90 6DOF Sensor for driveBAY/trakSTAR**	
	• Sensor OD = 0.9 mm • Sensor Length = 7.25 mm	• Cable OD = 0.6 mm • Cable Length = 10.9 ft. [3.3 m]
600798-PT-3.3	**3DG Model 130 6DOF Sensor for driveBAY/trakSTAR**	
	• Sensor OD = 1.5 mm • Sensor Length = 7.7 mm	• Cable OD = 1.2 mm • Cable Length = 10.9 ft. [3.3 m]
600799-PT-3.3	**3DG Model 180 6DOF Sensor for driveBAY/trakSTAR**	
	• Sensor OD = 2 mm • Sensor Length = 9.9 mm	• Cable OD = 1.2 mm • Cable Length = 10.9 ft. [3.3 m]
600786	**3DG Model 800 6DOF Sensor for driveBAY/trakSTAR**	
	• Sensor = 8 x 8 x 20 mm	• Cable OD = 3.8 mm • Cable Length = 10.9 ft. [3.3 m]

5 Degrees-of-Freedom (5DOF) Sensor Options		
600920	**3DG Model 55 5DOF Sensor for driveBAY2 /trakSTAR2**	
	• Sensor OD = 0.56 mm • Sensor Length = 100 mm	• Cable OD = 1.2 mm • Cable Length = 6.2 ft. [1.9 m]
600925	**3DG Model 90 5DOF Sensor for driveBAY2 / trakSTAR2**	
	• Sensor OD = 0.9 mm • Sensor Length = 100 mm	• Cable OD = 1.2 mm • Cable Length = 6.2 ft. [1.9 m]
600930	**3DG Model 130 5DOF Sensor for driveBAY2 / trakSTAR2**	
	• Sensor OD = 1.6 mm • Sensor Length = 8.0 mm	• Cable OD = 1.2 mm • Cable Length = 6.6 ft. [2.0 m]

5 Degrees-of-Freedom (5DOF) Junction Box	
601139	**3DG 5DOF Junction Box for driveBAY2 /trakSTAR2** Cable OD = 3.8 mm Cable Length = 8.0 ft (2.4m)

Cables	
103992	Cable, USB-A Male to USB-B Male
600797	Accessory Kit (Cable, Screws, Splitter)

Chapter 7

Regulatory Information and Specifications

Mid-Range, Short Range, Flat, MAGnet, and miniMAG Transmitter Configurations:

CE	In accordance with EN60601-1 (Medical electrical equipment – general requirements for safety), this equipment is classified as follows: Class I 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. The tracker alone is not a stand-alone medical device, and thus must be tested as part of the larger host system.

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 driveBAY 2™ Tracking System
(w/Mid-Range, Short-Range, Flat, MAGnet, and miniMAG Transmitters)
+ 12V: 1.6A nominal, 2.9A max
+ 5V: 1.0A nominal, 1.2A max

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 driveBAY 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.

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
Angular range:	All attitude: ±180 deg azimuth and roll, ±90 deg elevation
Static accuracy:	Mid-Range, Short-Range, Flat, MAGnet, and miniMAG Transmitters: (see note below) 1.4 mm RMS position 0.5 degree RMS orientation
Update rate:	Mid and Short Range transmitters up to 600 updates/second. Flat – 378 updates/sec (System Measurement Rate: 42Hz) MAGnet and miniMAG – 504 updates/sec (System Measurement Rate: 42Hz)
Position:	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.** In order to meet accuracy specifications, sensors must operate in the Perfomance Motion Box defined for the sensor/transmitter combination.
Outputs:	X, Y, Z positional coordinates, orientation angles, orientation matrix and quaternion.
Interface	USB 2.0

Physical

Size:	Mid-Range Transmitter: 3.75" (9.6cm) cube with 10' (3.05m) cable Short-Range Transmitter: 2.09"(5.3cm) x 2.09"(5.3cm) x 2.71"(6.9cm) Wide Range Trasnmitter: 12" (30.5cm) cube with 20' (6.15m) cable Flat Transmitter: D: 22"(56cm) x22"(56cm) x1.1" (3cm) W: 23 lbs. MAGnet Transmitter: D: 8.1"(20.6cm) x8.1"(20.6cm) x 1.8" (4.6cm) W: 7 lbs. miniMAG Transmitter: D: 4.2"(10.7cm) x4.2"(10.7cm) x 2.1" (5.3cm) W: 2.5 lbs. 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 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) 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) Model 90 Sensor: Sensor Housing: Outside Diameter: 0.9 mm (0.035 inch) Length: 8.8 mm (0.35 inch) Sensor Cable: 3.3 m (10.8 ft) Model 55 Sensor: Sensor Housing: Outside Diameter: 0.55 mm –0/+0.01mm (inch) Length: X.X mm (0.XX inch) Sensor Cable: X.X m (XX.X ft) Electronics Unit Dimensions (L x W x H): 17.7 cm x 14.7cm x 4.1 cm Weight: 0.94 Kg
Power:	DriveBAY2 requires: + 12V: 1.6A nominal, 2.9A max + 5V: 1.0A nominal, 1.2A max
Operating temperature:	41°F to 104°F (5°C to 40°C), 90% 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: 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 'driveBAY 2 Utility.exe' file located on the CD-ROM. This opens the interface configuration window of the Utility.

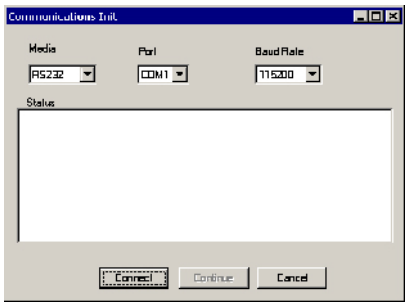


Figure 63 driveBAY 2 Utility Communication Initialize Window

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 **Dipole Settings** tab. Changing settings on this tab will affect power-up behaviors of the tracker when a Dipole transmitter (i.e. MRT,SRT) is attached. See Configurable Power-Up Settings for a description of these settings.

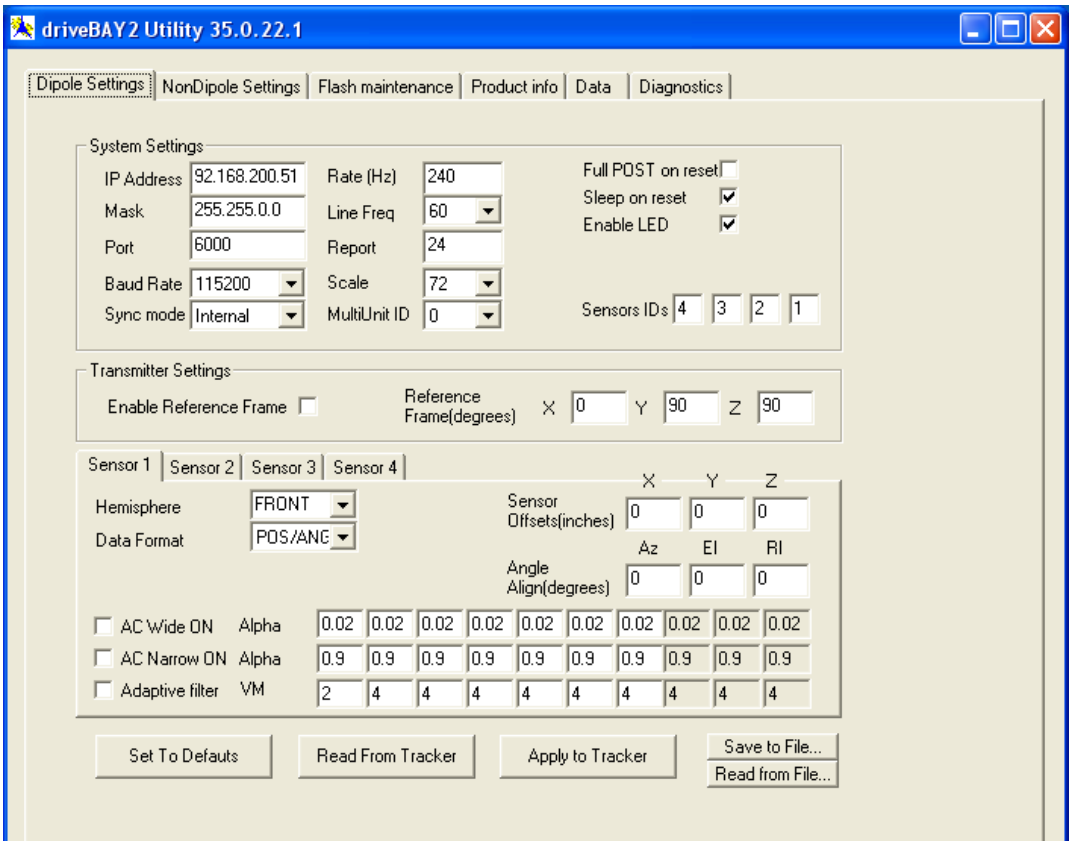


Figure 64 driveBAY 2 Utility Window (Dipole Default Settings Tab Displayed)

5. Select the **NonDipole Settings** tab to alter power-up behaviors of the tracker when a Non Dipole transmitter (i.e. Flat9, MAGnet,miniMAG) is attached. See Configurable Power-Up Settings for a description of these settings.

Note: If you change settings on either tab, you must click 'Apply to Tracker' (on any tab changed) and then re-boot the tracker for the new settings to take affect.

6. To upgrade Flash Sectors, click the 'Flash Maintenance' tab to show the contents of the flash memory device.

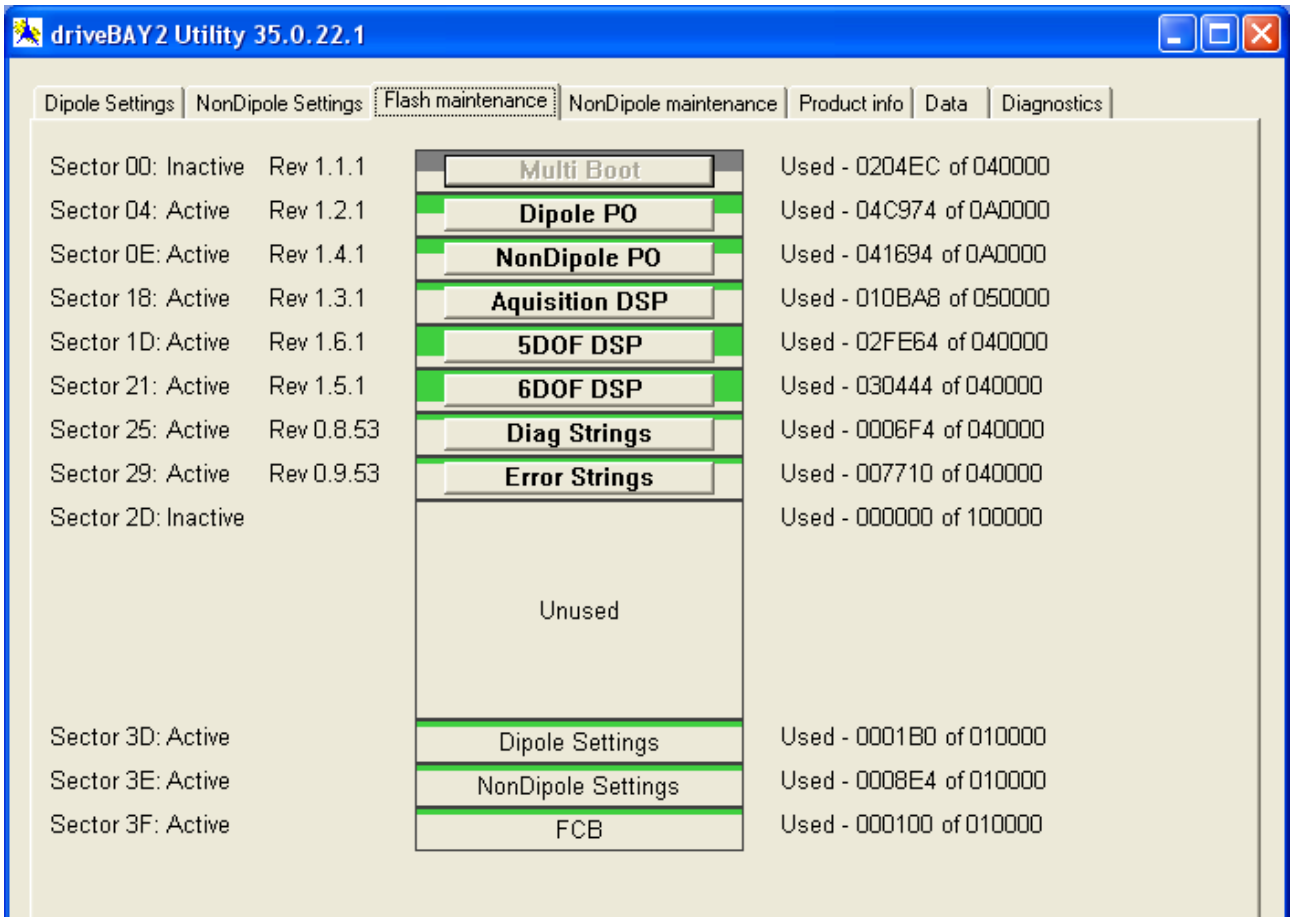


Figure 65 driveBAY 2 Utility Flash Maintenance

7. 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

8. Close the Utility ('X' in title bar) and cycle the power on the tracker.

Appendix II: Application Notes

Computing Stylus Tip Coordinates

In many applications that require a sensor mounted on a tool or pointing device, the position of the tip 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:

$$X_S = X_B + X_0 * M(1,1) + Y_0 * M(2,1) + Z_0 * M(3,1)$$
$$Y_S = Y_B + X_0 * M(1,2) + Y_0 * M(2,2) + Z_0 * M(3,2)$$
$$Z_S = Z_B + X_0 * M(1,3) + Y_0 * M(2,3) + Z_0 * M(3,3)$$

Where:

X_B, Y_B, Z_B are the X, Y, Z position outputs from the 3DGuidance™ sensor with respect to the transmitter's center.

X_O, Y_O, Z_O are the offset distances from the sensor's center to the tip of the stylus.

X_S, Y_S, Z_S are the coordinates of the stylus's tip with respect to the transmitter's center.

M(i, j) are the elements of the rotation matrix.

Often the values of X_O, Y_O, Z_O are not known ahead of time and must be calculated. This may be done by placing the tip of the stylus at a set location, collecting data of the stylus being repositioned with the tip fixed, and solving for X_O, Y_O, Z_O. Since the tip location (X_S, Y_S, Z_S) is fixed, and the 3DGuidance™ sensor position (X_B, Y_B, Z_B) and orientation (M(i, j)) are reported by the system, solving for X_O, Y_O, Z_O may be solved.

Collect many measurement points over a large range of angles and rotations for maximum accuracy.

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:

$$a = Number\ of\ transmitter\ axes$$

$$t_A = Axis\ cycle\ time = \frac{1}{Measurement\ Rate * a} \; (\text{sec})$$

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:

$$UpdateRate = \frac{1}{t_A} = a * Measurement\ Rate$$

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:

$$Update\ Rate = 3 * 80Hz = 240Hz$$

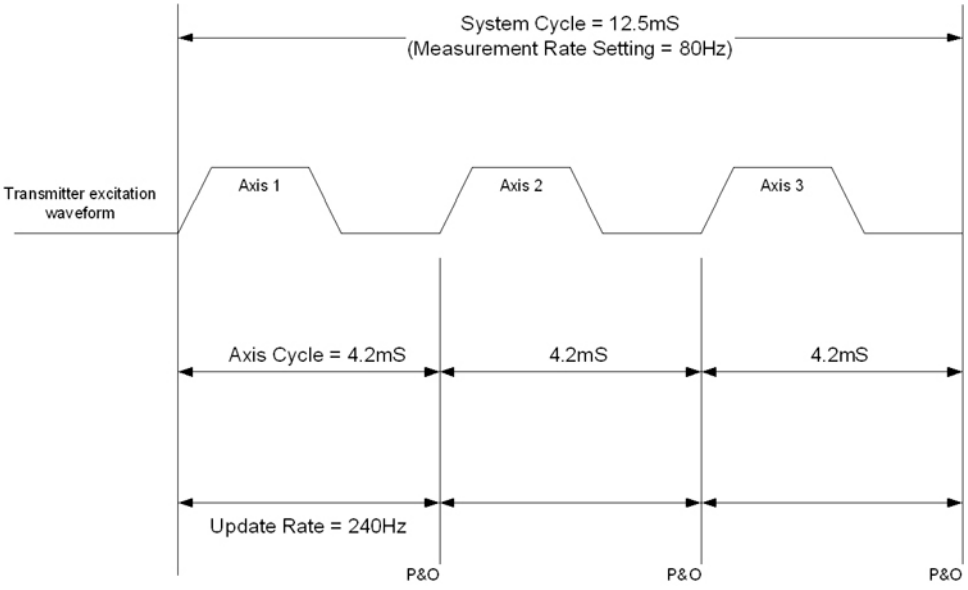


Figure 66 Transmitter Excitation Waveform

Appendix III: 3DGuidance Manual Starting Position for NonDipole Transmitters

Scope

This document covers setting the manual starting position for 3DGuidance driveBAY2/traksTAR2 and medSAFE trackers when running NonDipole transmitters. NonDipole transmitters include the MAGnet transmitter, the miniMAG transmitter, and Flat transmitters. This document does not apply to Short-Range, Mid-Range or Wide-Range Dipole transmitters.

Overview

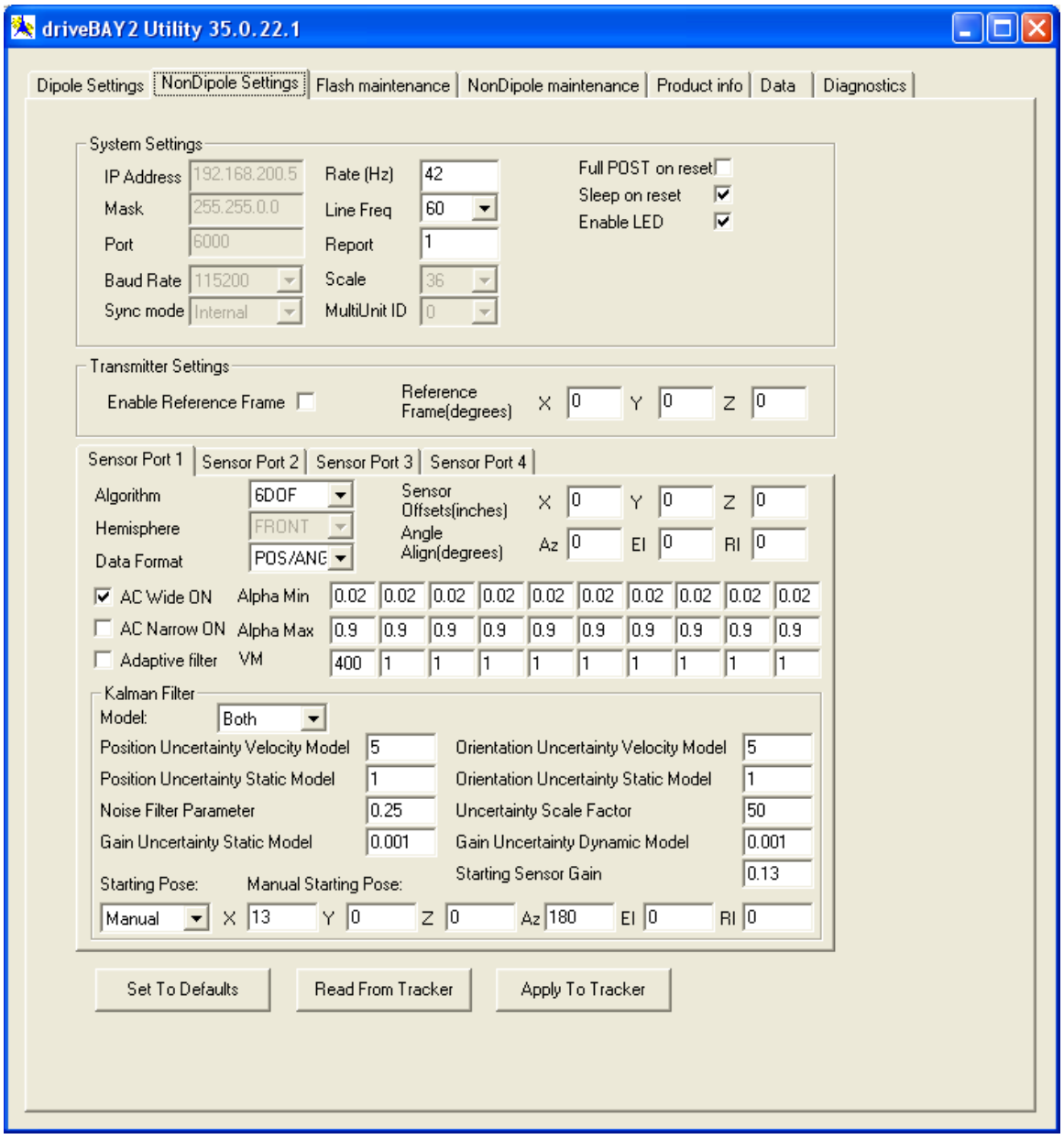
When tracking with NonDipole transmitters, 3DGuidance trackers use an iterative algorithm that is seeded with a starting pose (position and orientation). From that starting point, the algorithm converges on the true tracking solution.

Whenever tracking is interrupted and restarted, the starting pose is used. This includes power-up, sensor hot-swapping, running the transmitter after sleep, and re-entering the tracking volume.

The factory default setting (Auto) uses the center of the tracking volume for the position start and an orientation start of 0 degrees Azimuth, 0 degrees Elevation, 0 degrees Roll. In some cases, however, the algorithm may struggle to converge on the correct tracking solution, as evidenced by extremely noisy data with large quality/distortion values. In these situations, the sensor can be set to use a Manual starting pose selected by the user. By selecting a Manual starting pose close to the true position, algorithm convergence can be greatly improved.

Setting the Manual Starting Pose with the driveBAY2 Utility

Currently the Manual starting pose can be set at power-up using the NonDipole Settings tab in the driveBAY2 Utility (see screenshot below). The **Starting Pose** pulldown menu allows selection of **Auto** or **Manual** modes and the adjacent text boxes specify the **Manual Starting Pose**. After entering the desired settings, click **Apply To Tracker** to save the settings to non-volatile memory in the tracker. These new settings will take effect after powering down and restarting the tracker.



Setting the Manual Starting Pose at Runtime

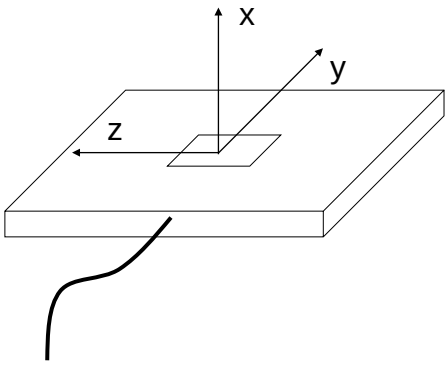
The 3DGuidance API does not currently support this functionality.

Determining the Appropriate Starting Position

The most straightforward way to choose the Manual starting pose is to place a sensor in a typical pose, record the tracking solution (e.g., with Cubes) and enter that position and orientation for the Manual Starting Pose. The following requirements must be observed:

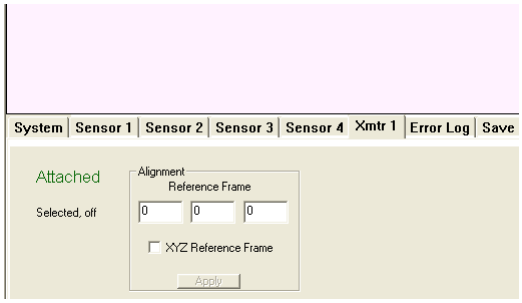
1. The Manual Starting Pose must be expressed in inches and degrees.
2. For MAGnet and miniMAG transmitters, the Manual Starting Pose must be expressed in the factory default transmitter reference frame, i.e., with the Reference Frame set to (0, 0, 0) and the XYZ Reference Frame turned off.
3. For Flat transmitters, the Manual Starting Pose must be expressed in the reference frame shown below, which corresponds to setting the Reference Frame to (0, 90, 90).

Flat Transmitter

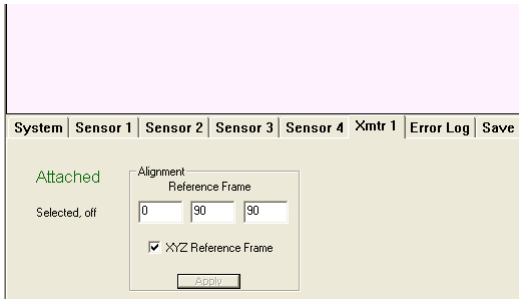


The Cubes screenshots below show the Reference Frame and XYZ Reference Frame settings to apply in the Xmtr tab for cases 2 and 3 above.

MAGnet and miniMAG



Flat Transmitter



Note that these reference frames are only required for determining the correct Manual Starting Pose settings. During tracking, the user is still free to apply any reference frame desired.