ASCENSION TECHNOLOGY CORPORATION

Tracking 3D Worlds

3D Guidance driveBAY™

Installation and Operation Guide

www.ascension-tech.com

3D Guidance driveBAY

Installation and Operation Guide

REVISION NUMBER:
940035 Rev C
06/13/08

TRADEMARKS

driveBAY is a trademark of Ascension Technology Corporation
© 2008 Ascension Technology Corporation. All rights reserved.
P.O. Box 527
Burlington, VT 05402

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

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

Table of Contents

Introduction

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

driveBAY is a high-accuracy electromagnetic tracker designed specifically for use in commercial, industrial and medical applications. Employing Ascension's new and advanced DC tracking technology, driveBAY tracks the position and orientation (six degrees of freedom) of multiple sensors within the operating range of its transmitter. Tracking data is reported serially to your host computer via a USB interface.

This Guide will help you setup, install, and use driveBAY hardware and software. It will also help you configure the device for optimal tracking so be sure to read it before proceeding.

About This Guide

This *Installation and Operation Guide* describes the simple steps you need to understand for operating the tracker and testing performance. It also covers protocols for configuring and communicating with the device.

How this Guide is Organized

This Guide contains seven chapters.

Chapter 1: Preparing for Setup and Safe Performance

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

Chapter 2: Setting Up and Checking the driveBAY

- Describes how to install the driveBAY
- Directs you on connecting 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.
- Discusses basic principles of tracker operation.
- Describes factors that affect tracker performance to include electromagnetic and other interference.

Chapter 4: 3D Guidance API Reference

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

Chapter 5: Troubleshooting

- Lists common setup problems and solutions.

Chapter 6: Maintenance, Repair and Disposal

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

Chapter 7: Regulatory Information, Symbols and Product Specifications

- Lists applicable standards, symbols, specifications, and certifications for this driveBAY.

Guide Conventions

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

Notes: Notes describe important hardware or software features.

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

Names of files, directories and programs: These are italicized (for example, *ATC3DG.lib*)

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

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

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

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

Getting Assistance

If you are experiencing a problem with the installation, setup, or operation of your tracker, we suggest your first consult the troubleshooting table in Chapter 5. It describes potential setup problems and how to resolve them. If you continue to experience problems, contact us as follows:

World Wide Web: http://www.ascension-tech.com/support/

E-mail: support@ascension-tech.com

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

Fax: (802) 893-6659

Chapter

1

Chapter 1: Preparing for Setup and Safe Performance

This chapter describes everything you will need to setup your 3D Guidance driveBAY system.

System Requirements

Intended Use Statement

driveBAY is designed to be integrated with a computer system that requires 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's CD-ROM:

1. *Cubes* - a demonstration utility that communicates using USB and the 3D Guidance API
2. *3D Guidance driveBAY Utility* – a utility for changing default configuration parameters of your system.
3. These utilities communicate with the tracker via the Windows API require Windows XP, Windows XP Embedded or Vista.

Note: A Windows® OS is only required for running the utilities or communicating with the tracker using one of the Windows APIs.

Hardware Requirements

USB port: driveBAY reports data serially to your host computer using a USB cable. It supports both USB 1.1 and 2.0 ports.

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

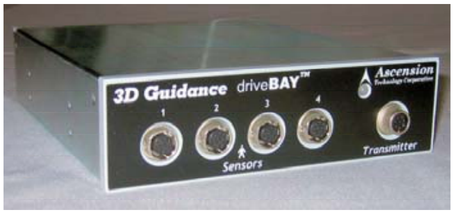
CD-ROM drive: Needed for accessing the utilities and drivers that are shipped on the driveBAY CD-ROM. You may also download these utilities from our FTP site: ftp://ftp.ascension-tech.com/

Unpacking the System

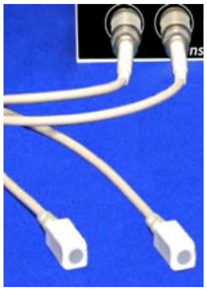
Package Checklist

Your driveBAY tracker is packaged in one shipping box. Make sure that you have the following components before setting up the tracker.

- 3D Guidance driveBAY Electronics Unit:



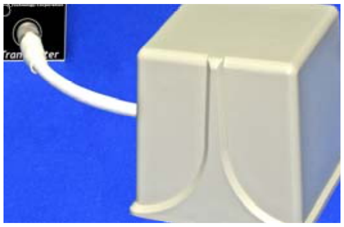
- One to Four Sensor(s):



Note:
The transmitters are specifically calibrated for use with the 3D Guidance driveBAY system.

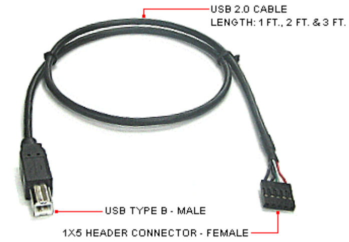
- 3D Guidance driveBAY supports two transmitter options:

Mid-Range Transmitter Option

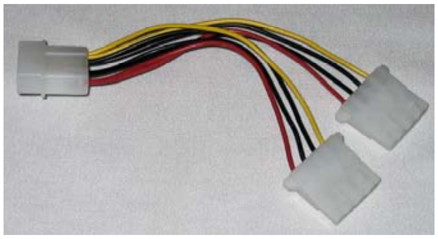


Or a Short-Range Transmitter Option

- Cables:
  - 1 USB Cable (hexagonal connector on one end and header connector on the other)



  - 1 Y Power Cable



- 3D Guidance driveBAY CD-ROM: It contains drivers, demo utilities, sample programs, and this guide.

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

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 & Handling Precautions

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

- Read this Guide.
- Keep your computer turned off while installing the driveBAY.
- Handle the section of cable near the sensor head or transmitter housing with care. Repeated bending of the cable near the sensor head or transmitter housing is the most common cause of tracker failure.

⚠CAUTION!

- When power is applied or the system is running, do not touch exposed electronic components. *Contact with exposed components could cause injury.*

- If you insert the sensor or transmitter in a mounting bracket or holder, be careful when you remove them. Do not yank or pull on the cable.
- Sensors and transmitters can be damaged if you carry, throw, or swing them by their cables or if you let them drop against hard surfaces.
- Sensor and transmitter cables have been precisely bundled, shielded, and calibrated to minimize noise and ensure accurate performance. Do not tamper with them. If you attempt to add your own extension cables or connectors, you may well compromise performance and, of course, void both regulatory approvals and your warranty.
- To clean your equipment, use a cloth to wipe components with a general purpose cleaning solution such as soap and water, isopropyl alcohol, etc. Do not immerse the transmitter, sensors, or cables in any liquids.
- Keep the transmitter, sensors, and cables away from sources of heat.

⊗ CAUTION!

- If mounting the transmitter inside an enclosure, be sure to provide adequate ventilation. Transmitters should not be mounted beneath mattresses, pillows, or any other object that will curtail air circulation in its immediate vicinity.

- Never power up the system or place the transmitter in an explosive atmosphere.
- Tracker components may be subject to interference from or may interfere with other electrical equipment in your environment. Be sure to identify sources of interference in your particular environment before using this tracker. See Electromagnetic and Other Interference in Tracking.

- Do not hang the mid-range transmitter upside down by its rear mounting holes. They are not designed to hold the full weight of the transmitter. Such a set-up could cause damage to the transmitter, nearby equipment, and even human injury.

- Care should be taken to avoid spillage on the electronics unit and components.
- Do not overly flex or twist the sensor cable.

- Do not allow the sensor or any cables to be crushed or subjected to undue strain and stress. The connectors can become warped if stepped on; the internal wires in the sensor cable can break or become weakened if pinched; and the sensor head may be damaged if trapped under heavy weights.

- Do not drop or smack the sensor head against a hard surface. Such impacts can produce internal damage and adversely affect tracking accuracy.

- Be sure to implement a strain relief if you embed a sensor and its cable in an instrument or tool. The point where the sensor cable exits from your tool needs protection. Your sensor will last a long time if you take steps now to distribute forces over an extended region of the cable.
- To extend tracker life, be sure to shut down the transmitter when not in use. You can do this in several ways:

    a. Select "No Transmitter" by setting the System parameter: SELECT_TRANSMITTER value to –1 in the 3D Guidance API.

    b. Recycle the power on the electronic unit (if independent access is available to the unit's power).

    c. Disconnect and reconnect the USB cable.

Environmental Conditions

The driveBAY 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 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

Chapter 2: Setting Up and Checking the driveBAY

This chapter explains how to install the driveBAY so that you can quickly begin tracking.

It consists of five parts:

- ♦ Setting up the driveBAY.
- ♦ Attaching the external cables.
- ♦ Installing the driveBAY drivers.
- ♦ Installing the utilities.
- ♦ Checking the driveBAY by running the demo.

Before You Begin

To install the driveBAY, you must first open your computer, next slide the driveBAY into an open bay, and then connect the driveBAY 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 *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 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 Vista or Windows XP operating system. You cannot install the driveBAY onto a Macintosh computer or one running the Macintosh operating system.
- ❑ The driveBAY'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. (You cannot install the driveBAY on a portable or laptop computer.) You should be able to easily install the driveBAY 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 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 driveBAY

This section explains how to install the driveBAY 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 2-1). Refer to your computer's guide for specific instructions.

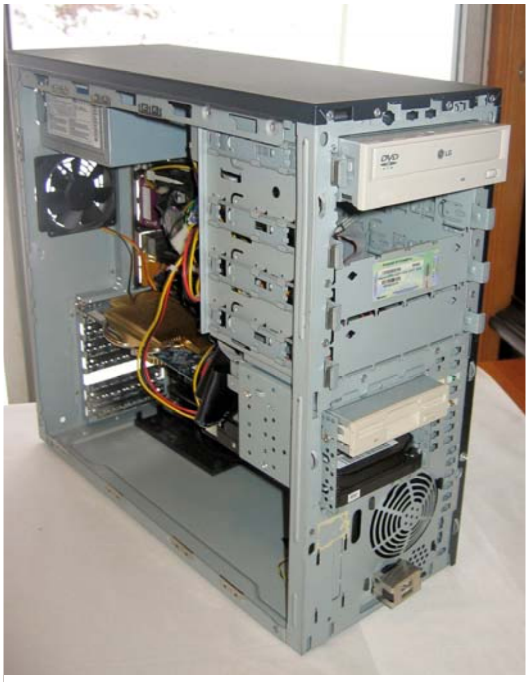


Figure 2-1: An open computer without panels

3. Make sure your computer can accept the driveBAY.

Check the open bay to make sure that it is large enough and deep enough to accept the driveBAY. 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 (Figure 2-2). There might be a free cable that you can plug directly into the driveBAY. If not, you will have to install our HDD power cable Y-adapter.

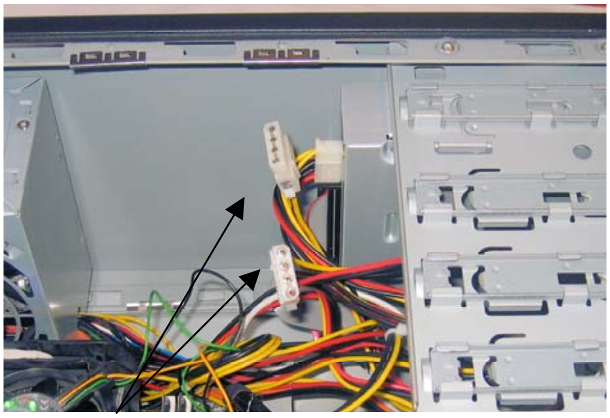


Figure 2-2: HHD power cables

Locate the USB header port on the motherboard. The word 'USB' might be stamped on the motherboard to identify the ports (Figure 2-3). 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. Close up your computer, and call Ascension technical support.

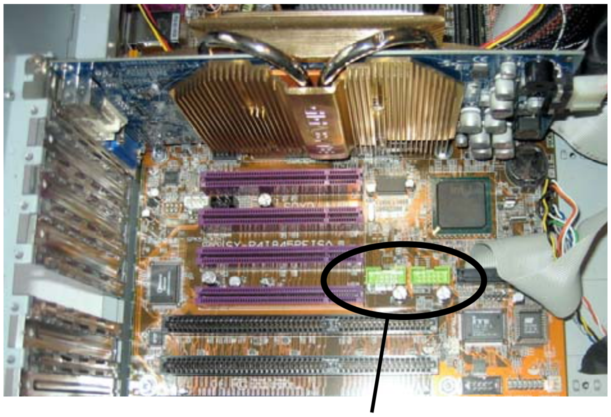


Figure 2-3: Two USB header ports (2 x 5 configurations) on your computer's motherboard

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 (Figure 2-4)

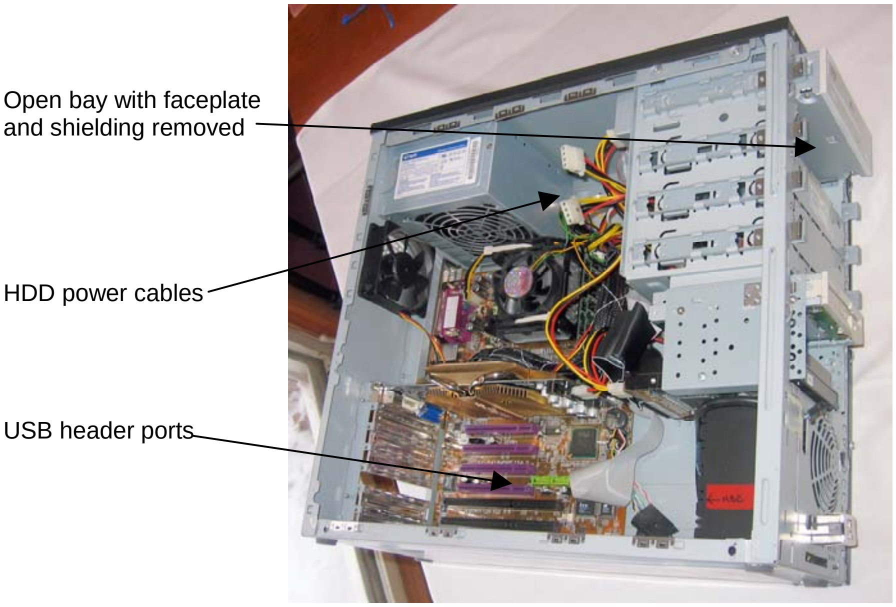


Figure 2-4: An inside look at a computer

5. Insert the driveBAY into the open bay.

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

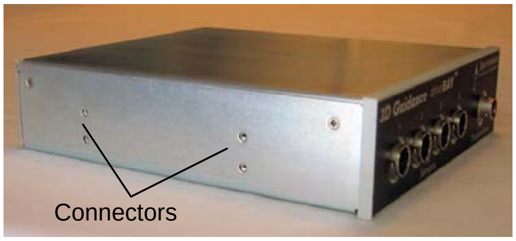


Figure 2-5: For connecting to the computer's chassis

Align the driveBAY and slide it into the open bay until the driveBAY is fully inserted into your computer (Figure 2-6).

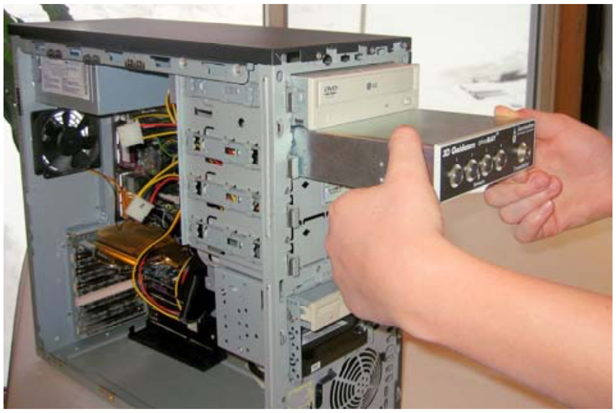


Figure 2-6: Align and slide in driveBAY

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. You might have to adjust the angle of the driveBAY slightly to gain a perfect alignment. Using the Phillips screwdriver, insert the mounting screws through the holes or slots and into the driveBAY; tighten the screws until they are just snug. Avoid over-tightening the screws (Figure 2-7).

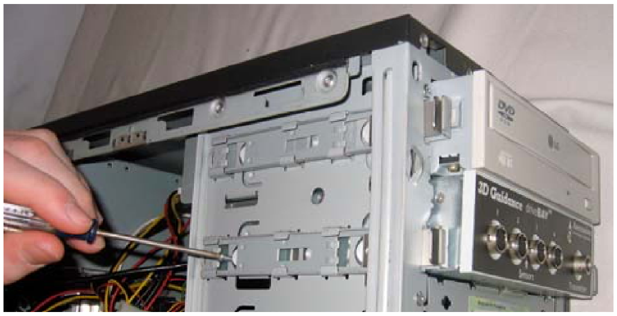


Figure 2-7: Screw in the driveBAY

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 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 into your computer chassis without first attaching mounting slides to the sides of the driveBAY. 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, screw them into place, then slide the driveBAY 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, and any other peripheral connected to the motherboard. Make sure you carefully follow these instructions to avoid any problems.

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

+5V	■	■	+5V
D–	■	■	D–
D+	■	■	D+
Grd	■	■	Grd

Figure 2-8: A 2x4 layout with pins in the same direction

+5V	■	■	Grd
D–	■	■	D+
D+	■	■	D–
Grd	■	■	+5V

Figure 2-9: A 2x4 layout with pins in the reverse direction

+5V	■	■	+5V
D–	■	■	D–
D+	■	■	D+
Grd	■	■	Grd
S-Grd	■		

Figure 2-10: A 2x4 layout with pins in the same direction with an extra ground pin

+5V	■	■	Grd
D–	■	■	Grd
D+	■	■	D+
Grd	■	■	D–
Grd	■	■	+5V

Figure 2-11: A 2x4 layout with pins in the reverse direction with two extra ground pins

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 2-12 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.

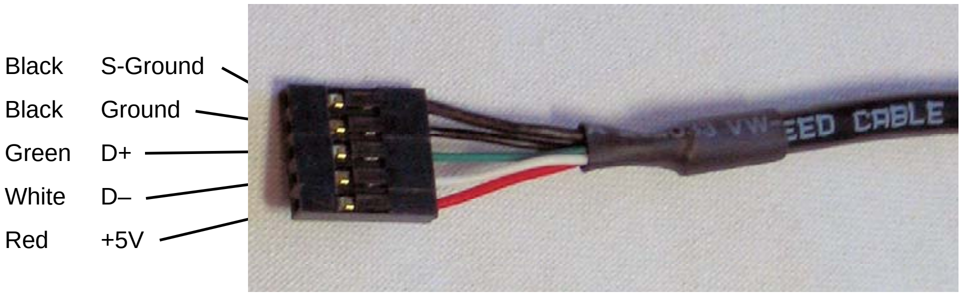


Figure 2-12: Color coded wire end of USB Cable

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 2-13).

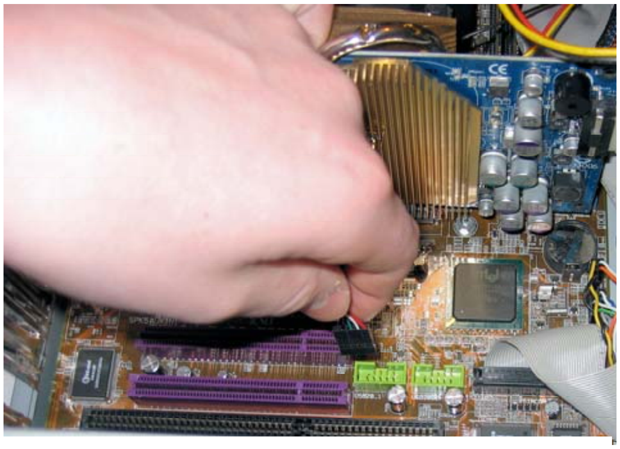


Figure 2-13: Plugging the wired end of the cable into the USB header port

7. Attach the power cable to the driveBAY.

Attach an unused HDD power cable into the back of the driveBAY (Figure 2-14).

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.

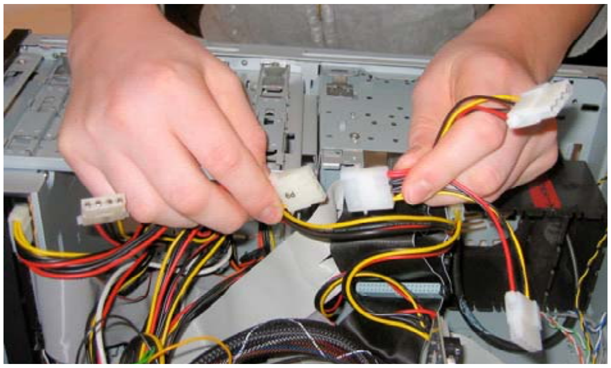


Figure 2-14: Plug the Y adapter into the HDD power cable

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 (Figure 2-15).

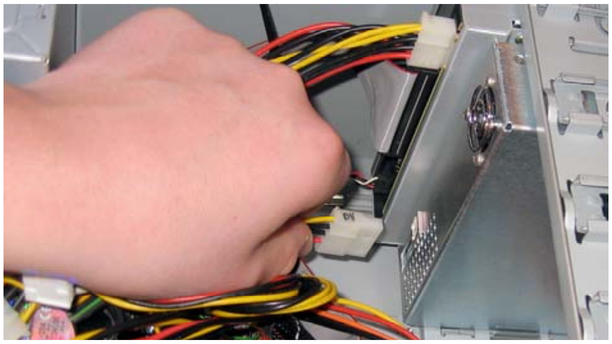


Figure 2-15: Plug the HDD power cable or the Y adapter Into the back of the driveBAY

8. Connect the USB cable to the driveBAY.

Connect the Type-B end of the USB cable to the back of the driveBAY (Figure 2-16).

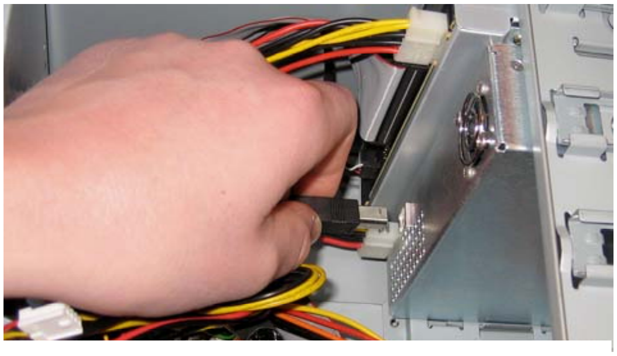


Figure 2-16: Connect the USB cable’s Type B end to the Back of the driveBAY

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 (Figure 2-17).

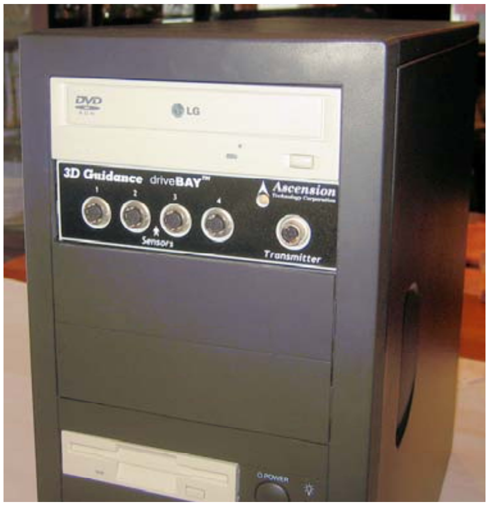


Figure 2-17: The driveBAY installed in your computer

Attaching the External Cables

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

1. **Connect a transmitter to the driveBAY.**

   Connect the mid-range or the short-range transmitter cable to the connector marked 'Transmitter' on the front of the driveBAY. Align the slot on the top of the transmitter cable's plug with the connector, and then push the plug into the connector until it clicks into place (Figure 2-18).

   Caution! The transmitter is heavier than it looks. Grasp it firmly before moving it.

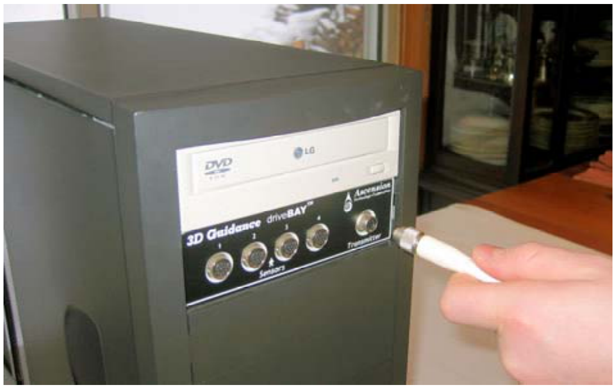


Figure 2-18: Connect the transmitter cable to the driveBAY

   Setup the transmitter in a non-magnetic location (such as wood or plastic, but never on the floor). Try to keep at least a 61-centimeter (24-inch) radius around the transmitter free from any object (which means keeping the transmitter at least that distance away from your computer).

2. **Connect the sensors to the driveBAY.**

Connect a sensor to the 'Sensors' connector marked with a '1'. Rotate the sensor plug until it aligns with the holes on the 'Sensor' connector, and then push until the plug clicks into place (Figure 2-19).

You can connect up to three more sensors to the driveBAY, into the Sensor connectors marked '2', '3', and '4'.

3. **Turn on your computer.**

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

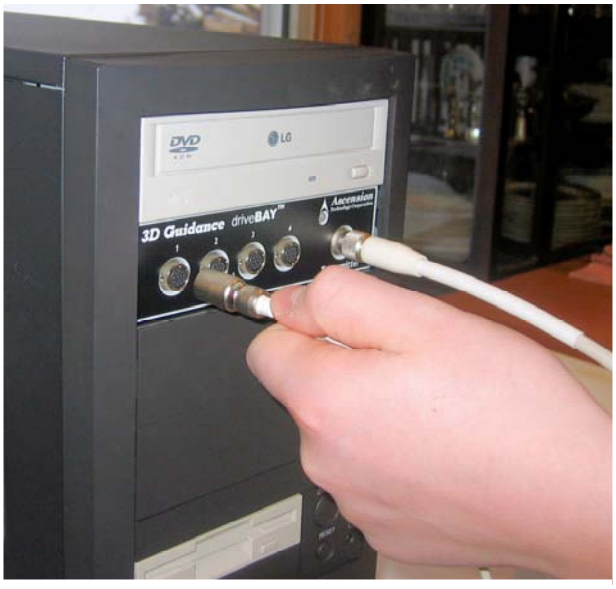


Figure 2-19: Connect the sensor to the driveBAY

The indicator light (on the front of the driveBAY) slowly blinks with a yellowish light while the driveBAY initializes. When complete, the indicator light turns to a slow blinking green. This takes about 10 seconds.

Note: If a slow blinking green light does not appear, then the driveBAY has not properly initialized. Shut down your computer, then open it up and double check all the cable connections. Make sure all the cables are fully seated in their plugs. Close your computer back up, and turn it back on. If you are still having trouble, please refer to the troubleshooting section of this *3D Guidance driveBAY Installation and Operation Guide* for steps to troubleshoot this problem. You can also call Ascension technical support for help.

Installing the driveBAY Drivers

After turning your computer on, the Windows operating system detects the new hardware – the driveBAY – and starts the *New Hardware Wizard*. Follow the prompts on the *New Hardware Wizard*.

1. Allow the *New Hardware Wizard* to search for suitable drivers.
2. When prompted, insert the *3D Guidance driveBAY Installation and Technical Reference CD* into your computer's CD-ROM drive and choose to install the driver's software automatically.
3. Allow the *New Hardware Wizard* to search for the best USB and driveBAY drivers on the driveBAY's CD-ROM.
4. If you receive a warning about the drivers not having passed Windows testing, you can safely ignore it. Simply continue with the installation.
5. After the drivers have been installed, finish by closing the *New Hardware Wizard*.

Tip: You can download the latest drivers and DLLs from the Ascension web site.

Installing the Utilities

The driveBAY CD contains several utilities that you can install onto your computer.

1. Insert the driveBAY CD-ROM into your computer's CD-ROM drive. The window for the install menu automatically opens. (If the CD-ROM is already in the drive, select index.html in the directory to get started.)
2. Click on the *Install 3D Guidance driveBAY Utility* and follow the prompts in the *Setup Wizard*.
3. For the USB demonstration software, click on the *Install CUBES* option and follow the prompts in the *Setup Wizard*.

Note: A DOS-based demo utility, *USBTest.exe*, is also available in the *Direct* sub-directory of the CD-ROM if you would prefer to run that.

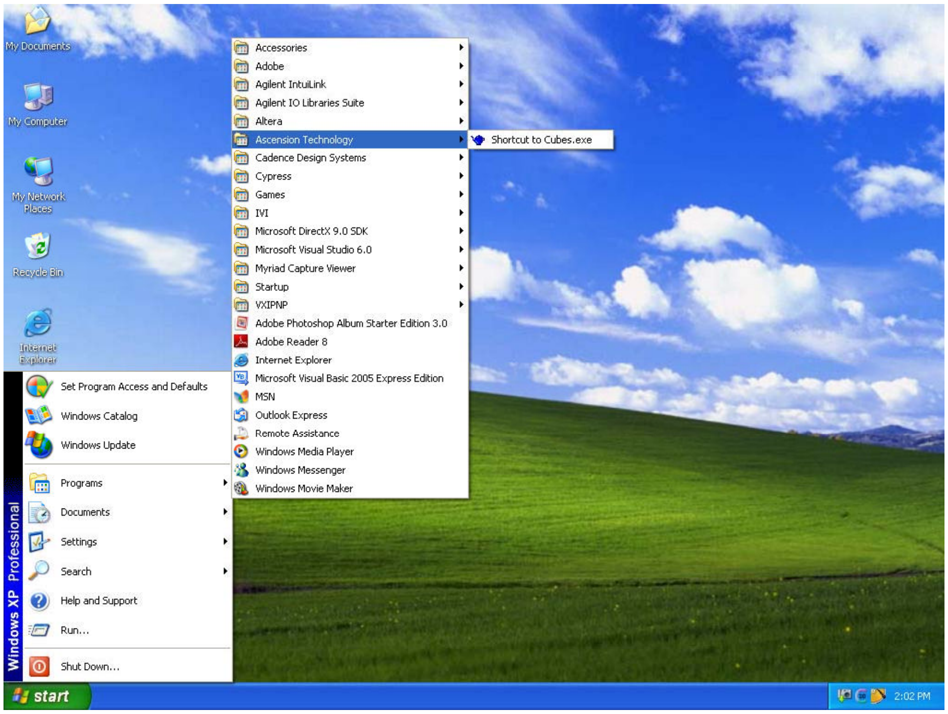
Tip: You can also download these utilities from the Ascension web site.

Checking the driveBAY by Running the Demo

Now that the driveBAY is running and the drivers and utilities installed, you are ready to run the demo software and checkout your system. Proceed as follows:

1. Start the demo utility by selecting *Cubes* from the Ascension Technology program group in the Windows Start menu. (Figure 2-20)

i

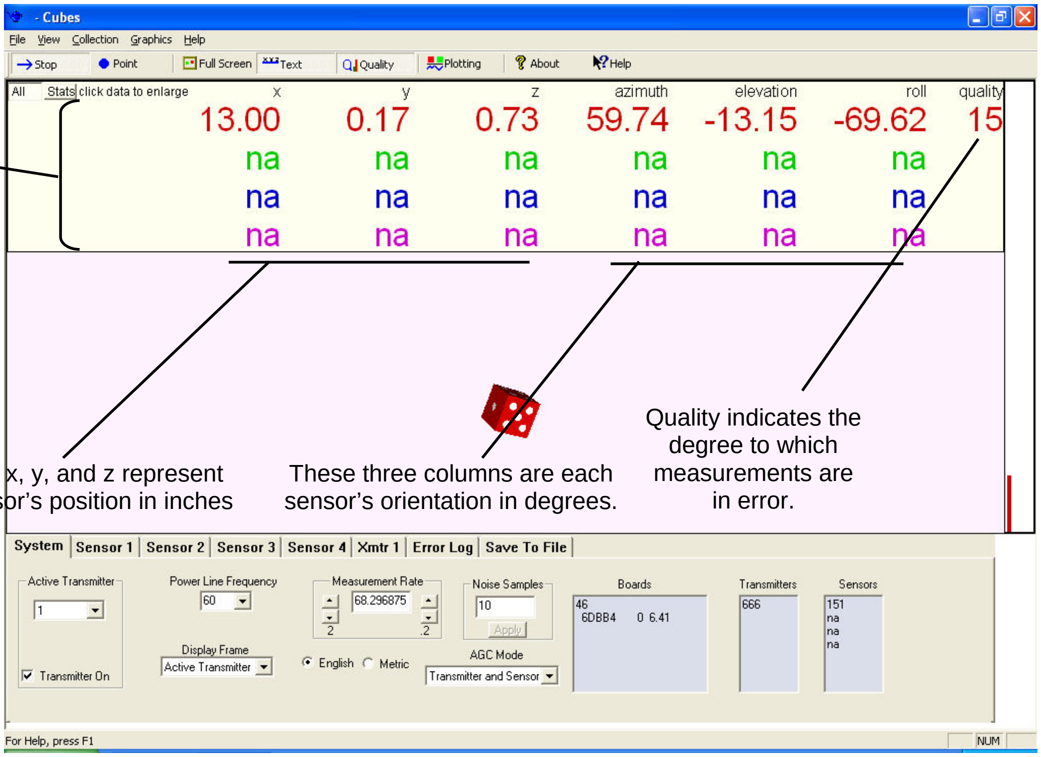


a

Figure 2-20: Selecting the Cubes program from the Windows start

There is a brief three-second pause while the *Cubes* program establishes communication with the driveBAY.

Cubes then displays the main window of the utility (Figure 2-21).



All	Stats	click data to enlarge	x	y	z	azimuth	elevation	roll	quality
			13.00	0.17	0.73	59.74	-13.15	-69.62	15
			na	na	na	na	na	na	
			na	na	na	na	na	na	
			na	na	na	na	na	na	

Active Transmitter: 1

☑ Transmitter On

Power Line Frequency: 60

Display Frame: Active Transmitter

Measurement Rate: 68.296875, 2, 2

Noise Samples: 10 Apply

● English ○ Metric

AGC Mode: Transmitter and Sensor

Boards: 46 6DBB4 0 6.41

Transmitters: 666

Sensors: 151 na na na

For Help, press F1 NUM

Each row represents a sensor, colored coded with the cubes in the middle of the window. In this example, only one sensor is attached.

Columns x, y, and z represent each sensor's position in inches

These three columns are each sensor's orientation in degrees.

Quality indicates the degree to which measurements are in error.

Figure 2-21: Cubes main window

2. To start collecting data from the driveBAY, press the 'Continuous Run' icon on the toolbar (Figure 2-22).

Press 'Continuous Run' to start

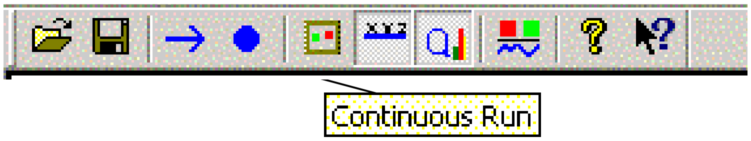


Figure 2-22: Continuous Run icon

If the *Cubes* utility does not run, please consult the troubleshooting table in this guide for assistance.

Interpreting the Information on the Main Window

The top of the main window displays the reported position and orientation for each sensor attached to the driveBAY. Each color-coded row contains the information for a single sensor. The first three values in a row represent sensor position, *in inches*, relative to the transmitter. The next three values in the row represent the sensor's orientation *in degrees*.

Tip: For additional information and assistance on using the QUALITY number, see the *APIReference* found in Chapter 4.

The last column in each row lists the reported quality number. This value indicates the degree to which the position and angle measurements are in error. Error is often attributed to metal in the environment. See the "Performance Factors" section of Chapter 3 for details.

The center of the window displays a colored cube for each sensor attached to the driveBAY. The colors of the cubes correspond to the rows at the top of the window.

The bottom of the window allows you to configure and adjust all the driveBAY's system parameters. Simply click the tabs, then click to select or enter the values you want.

Use the *Cubes* demo utility to become familiar with the sensor's motion region and the tracker's capabilities. If the utility does not run or the driveBAY does not operate as described, please consult the troubleshooting table in of this guide for assistance.

Proceed to Chapter 3 for details about the driveBAY's default configuration and its basic operation.

Chapter

3

Chapter 3: Configuration and Basic Operation

This chapter tells you how to configure the driveBAY and basic operating parameters and factors that influence performance.

DriveBAY is configured at Ascension to optimize tracking for most applications. You can also customize the power-up behavior of the driveBAY to better meet your specific application requirements.

Follow these steps to customize your driveBAY:

1. Review the list of configurable default settings.
2. Determine the settings (if any) you would like to change.
3. Follow the steps in the 'Changing Your Settings' section to run the *driveBAY Utility*.

Default Configuration

The settings in the following table are installed as power-up defaults. The *driveBAY Utility* may be used to alter this default power-up configuration. Those settings not accessible with the Utility may be changed during normal operation through the appropriate software command.

Description: (detailed description following this table)	**Default Setting**	**Utility**	**Configurable via SW call**
Measurement Rate:	Mid and short range transmitters: 80Hz *Note 1: measurement rate cannot be set below 20 Hz.* *Note 2: the tracker update rate is always 3 times measurement rate.*	✓	✓
Scale:	36 inches	✓	✓
Sensor Offsets:	x = 0, y = 0, z = 0	✓	✓
Angle Align:	az = 0, el = 0, rl = 0	✓	✓
Transmitter Reference Frame:	az = 0, el = 0, rl = 0	✓	✓
Hemisphere:	Front	✓	✓
Filter On/Off Status:			
AC Wide Notch	ON	✓	✓
AC Narrow Notch	OFF	✓	✓
ADAPTIVE Filter (DC)	ON	✓	✓
ALPHA_MIN	0.02	✓	✓
ALPHA_MAX	0.90	✓	✓
VM Table	2, 4, 4, 4, 4, 4, 4, 4, 4	✓	✓
Data Format:	POSITION/ANGLE	✓	✓
Report Rate:	1	✓	

Table 1

Configurable Power-up Settings:

You can reconfigure the following parameters using the d*riveBAY Configuration Utility*.

Measurement Rate Sets the acquisition rate for the tracker. You can alter it to optimize susceptibility to distortion from certain metals. See Environment section below.

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

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

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

Reference Frame Defines the reference frame centered at the transmitter's X, Y, and Z-axes. (See Figure 3.1) Reference Frame settings allow you to enter the angles required to mathematically align the axes of the transmitter with those of a new reference frame.

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

AC Wide Notch An eight tap finite impulse response (FIR) notch filter that is applied to the sensor data to eliminate sinusoidal signals with a frequency between 30 and 72 HZ.

AC Narrow Notch A two-tap FIR notch filter that is applied to signals measured by the tracker's sensor. You can use this filter in place of the AC WIDE notch filter when you want to minimize the delay between the measurement and outputs of the sensor data. The transport delay of the AC NARROW notch filter is approximately one third the delay of the AC WIDE notch filter.

Adaptive Filter When ON, it is a low pass filter applied to sensor data that eliminates high frequency noise. Generally, this filter is always ON, unless your application permits noisy outputs. When the filter is ON, you can modify its noise/lag characteristics by changing ALPHA_MIN and Vm.

Data Format Sets the default data format for returned data records.

Report Rate Outputs rate divisor setting. It reduces the number of records output, during STREAM mode (also set by the GetSynchronousRecord call), to the rate determined by the setting. Default setting of 1 makes all outputs available.

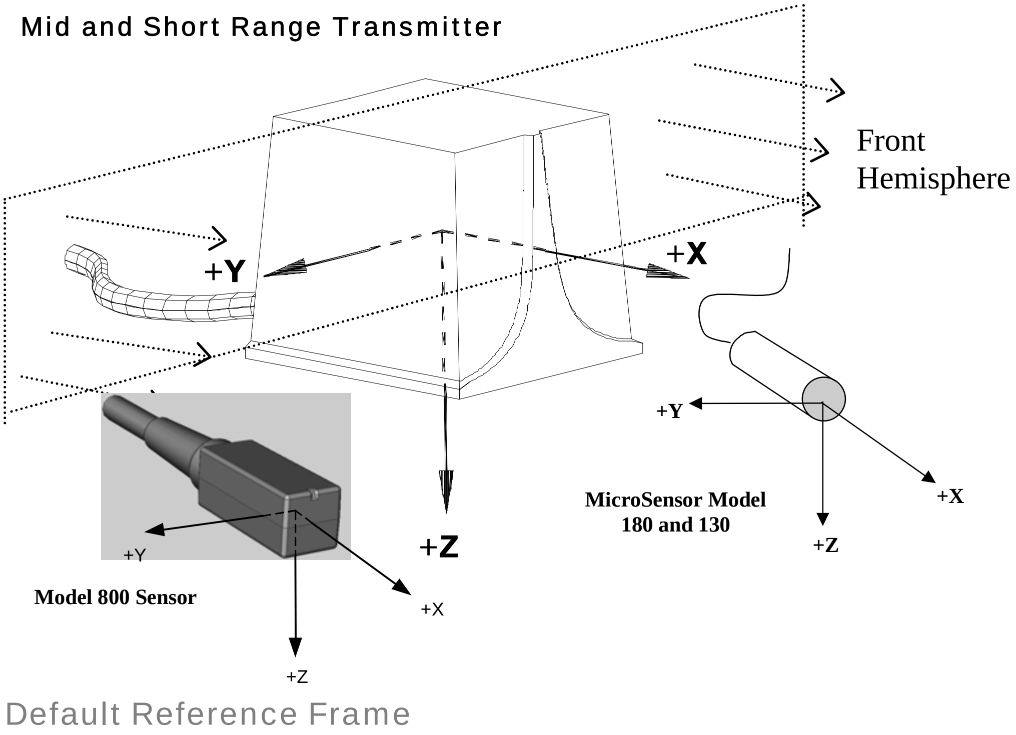


Figure 3.1: Default Transmitter/Sensor Coordinate Frames

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

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

Mid-Range: (cable exits lower rear face)

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

Short Range: (cable exits lower rear face)

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

Changing Your Settings

The *driveBAY Utility* may be used to alter power-up defaults for the system. See Appendix I, *driveBAY Utility*

Mounting the Hardware

Mid-Range Transmitter Mounting and Location

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

⊗ **CAUTION!**
Mounting holes not strong enough to hold the transmitter upside down

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

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

Your transmitter should be located at least 24 inches from the electronics unit. It should not be placed within 10 feet of another operational transmitter since the two may interfere with one another.

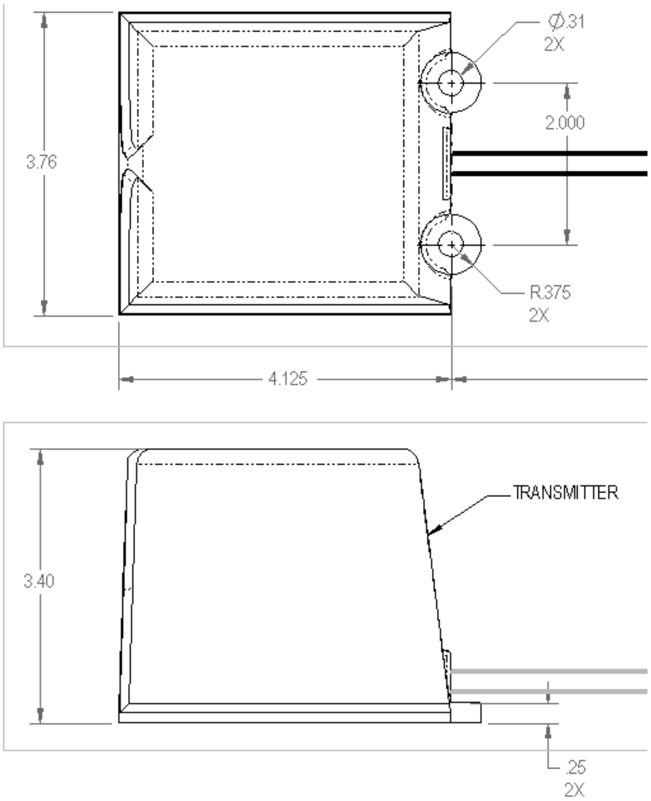


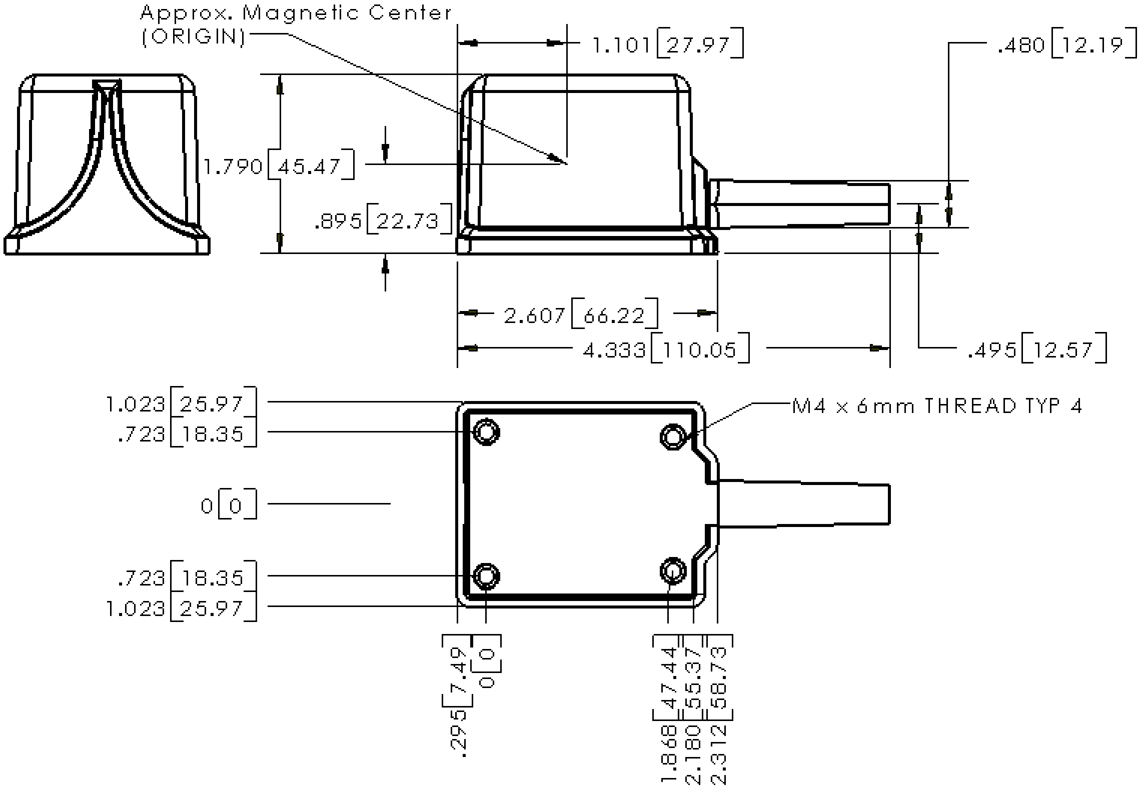
Figure 3.2

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

Short-Range Transmitter Mounting and Location

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

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



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

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

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

Sensor Mounting

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

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

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

Locating Your PC Containing the driveBAY Electronics Unit

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

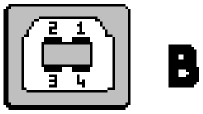
Rear Panel Connectors

There are two connectors on the back panel of the driveBAY unit.

USB

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

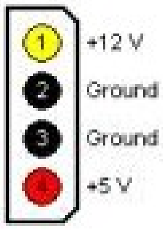
PIN 1: VCC +5VDC
PIN 2: DATA-
PIN 3: DATA+
PIN 4: GROUND



Power

The power connector is a standard disk drive power connector.

Pin 1: +12V: 1.6A nominal with 4 sensors loaded, 2.9A max @ 10.8V
Pin 2: Ground
Pin 3: Ground
Pin 4: +5V: 600 mA nominal



Basic Operation

driveBAY determines the 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. 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

Dipole Transmitter

Two transmitter options are available with driveBAY.

Both the mid and short-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 inverse 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 is sequentially energized in this manner during each measurement cycle.

6DOF Sensor

A 6DOF sensor outputs its position in three dimensions and the orientation of its axes relative to the tracker reference frame. Sensors are factory-calibrated and the calibration data is stored on a memory chip in the sensor's connector housing.

Electronics

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

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

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

The electronics unit contains two onboard processors:

- POServer: It handles all communications to and from the host PC as well as computes the tracking solutions.

- Acquisition/MDSP: It performs all acquisition and digital signal processing of the sensor data.

Measurement Cycle

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

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

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

Calibration

driveBAY components are manufactured within tight tolerances, but still exhibit slight variations. These differences are measured, recorded, and adjusted for by the tracking algorithm to consistently generate accurate measurements. Calibration values represent the measured difference between a particular component and its ideal state.

Calibration allows components to be interchanged with minimal effect on tracker outputs. 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. 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

Excessive Electrical Noise

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

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

CAUSES OF NOISE

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

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

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 such things as 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.

REDUCING NOISE

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

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

Magnetic Distortion

Distortion is a constant deviation from the correct value. 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 tracking volume or motion box. The ferrous magnetic property of the metal, the electrical conductivity of the metal, the physical orientation, and other physical features will all alter the level of tracking distortion.

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

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

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

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

REDUCING DISTORTION

As noted, metal is the primary cause of distortion. Removal and reduction of the amount of metal in and around the tracker is most effective means of controlling distortion in measurements. Sources of metal distortion cannot be shielded, as is often the case with noise sources.

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

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

METAL DETECTION

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

QUALITY/ METAL NUMBER

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

Tracker as the Cause of Interference

Just as some driveBAY 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 driveBAY. If you rely on life-sustaining equipment, such as a pacemaker or defibrillator, be sure to consult with your physician prior to powering up this tracking device.

Factors in Tracker Accuracy

Warm-up

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

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

Default Measurement Rate

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

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

- The application requires a specific measurement rate (e.g. it must be in synchronization with video update rate or another measurement tool).

- The environment is electrically unstable at the default measurement rate or significantly more stable at another measurement rate.

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

Equipment Alteration

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

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

Power Grid Magnetic Interference

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

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

Performance Motion Box

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

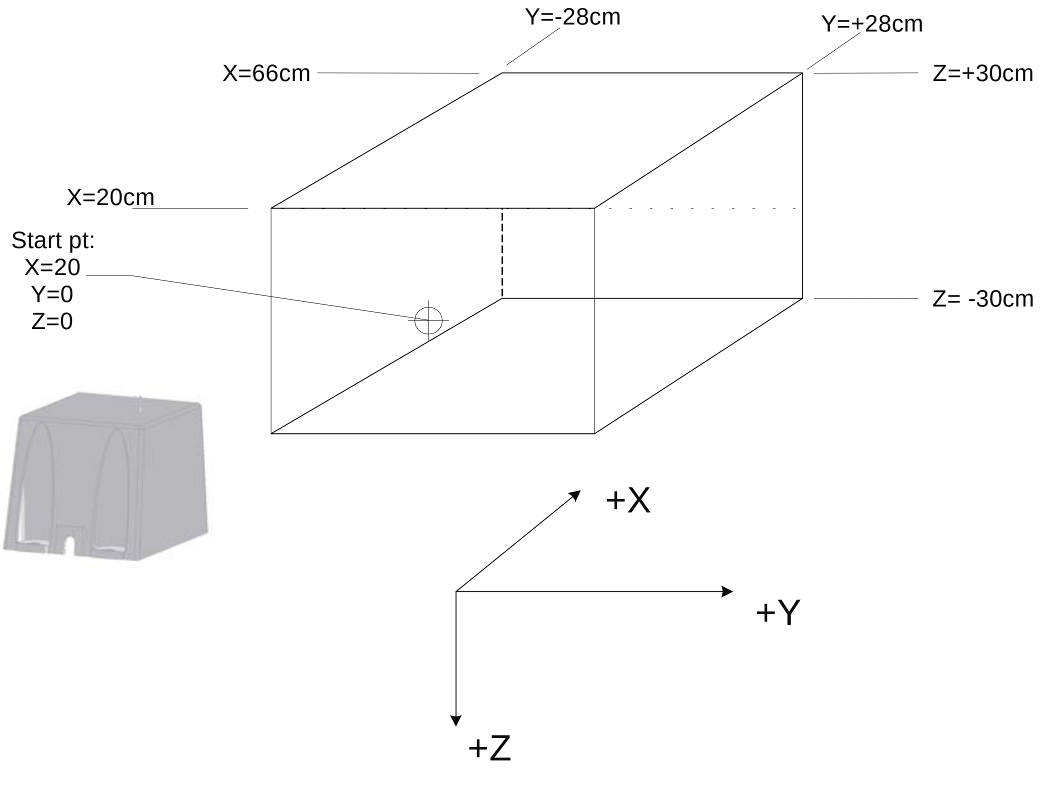


Figure 3.3: Performance Motion Box Mid-Range Transmitter

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

For **Model 800** sensors:

X = 20 to 66cm from the transmitter center

Y = ±28 cm from the transmitter center

Z = ±30 cm from the transmitter center

For the Short Range Transmitter:

For **Model 800** sensors:

X = 15 to 41cm from the transmitter center
Y = ±12 cm from the transmitter center
Z = ±12 cm from the transmitter center

Chapter

4

Chapter 4: 3D Guidance API Reference

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

3D Guidance API Overview

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

If you are new to the 3D Guidance family of trackers and the 3D Guidance API, you will find several tools on the CD-ROM for experimenting with the tracker's capabilities and quickly creating custom code. These tools include two demo applications, CUBES and pciBIRDTalk, and two sample programs containing C++ project files. The tables below describe available tools and their source locations.

3D Guidance API files

API Components	***Description***	***Location***
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	CD-ROM and Website
ATC3DG.lib	Library file required during compiling of any code that makes calls to 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.	
Cubes	A Demo Windows application that displays tracking data (both test and graphical representation) for up to 10 sensors. Supports the following functions: 1. TX On/off -turn the transmitter on or off 2. System RESET - reset/re-initialize the tracker 3. System settings - change measurement rate, line frequency, AGC mode, English vs. metric units 4. Sensor settings- change quality parameters, angle align, and filter settings 5. Transmitter - change reference frame 6. Graph Mode - Plot up to 3 parameters 7. Noise Statistics - Collects X samples and shows AVG, peak-to-peak and RMS deviation NOTE: Cubes is a demo program and should NOT be used when fast data collection is required.	CD-ROM
PCIBirdTalk	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: 1. Check response to particular command without implementing in your own code 2. Check response using a particular non-default setting (filter, etc) without implementing in their code 3. Confirm/Debug hardware and their code by reproducing chosen settings and viewing response 4. Save system settings that have been tied to an .ini file (using SaveSystemConfig call) NOTE: It's important to know how commands are defined and sequence of commands to send for this tool to be used effectively	

Sample 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 3D Guidance API. Specifically, it: 1. Initializes the Tracker 2. Reads in the system configuration (status of board, sensors, transmitter, etc) 3. Turns on the transmitter 4. If all above is valid, collects 100 records (POS/ANG format) from each of the sensors and streams them to the screen. 5. Error Handler - A simple error handler is included. It just takes any error codes returned from the DLL/Tracker and sends them to the screen 6. Transmitter Off - Before closing the application, shows how to turn off the transmitter All necessary source files to re-build and run the application are included, and each of the above steps are accompanied by detailed comment descriptions directly in the code.	CD-ROM
Sample2	This C++ project also contains sample code for a very simple console application using the 3D Guidance 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 the 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 are accompanied by detailed comment descriptions directly in the code.	CD-ROM

Using 3D Guidance API

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

Quick Reference

System Initialization

System Setup

Transmitter Setup

Sensor Setup

Acquiring Position and Orientation Data

Error Handling

Quick Reference

SYSTEM

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

➢ 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)

- **SELECT_TRANSMITTER**

  This command allows us to turn on next transmitter or turn off the current transmitter in the driveBAY.

- **POWER_LINE_FREQUENCY**

  This parameter represents the frequency of the AC power source used by the driveBAY. You need to set it for proper operation of the AC_Wide_Notch_Filter.

- **AGC_MODE**

  The driveBAY has two basic modes of operation.

1) TRANSMITTER_AND_SENSOR_AGC (Not yet implemented)

   In this mode, an automatic gain control (AGC) system implemented in the firmware will dynamically adjust the gain of the input variable gain amplifier (VGA) AND the power level of the transmitter in order to keep the sensor input signal within the dynamic range of the system.

2) SENSOR_AGC_ONLY

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

- **MEASUREMENT_RATE**

The measurement rate is the sample rate of the system. The tracker’s measurement rate is nominally set at 80.0 measurements/second. You can increase the tracker’s measurement rate to a maximum of 115 measurements/second. The downside of going to rates faster than 103 measurements/second is that the noise on your outputs may increase and any errors introduced by nearby metals will increase. You can decrease the tracker's measurement rate to no less than 20 measurements/second. Decreasing the measurement rate is useful if you need to reduce errors produced by 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 reduce the distortions. For low-conductive, low permeability metals such as 300-series stainless steel or nickel, speed changes will have minimal effect. These metals do not introduce errors into the tracker's measurements.

- **Position Scaling**

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

- **METRIC Position Representation**

There is a system option that allows the data formatted in double precision floating point format to be output pre-scaled to either inches (the default) or millimeters. Setting the METRIC flag true will cause output to be in millimeters.

SENSOR

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

➢ Data Format:	Double precision floating point Position/Angles
➢ Angle Align:	0, 0, 0
➢ Filters:	All values in **Alpha max** table = **0.9000**; all values in **alpha min** table = **0.0200**; **Vm table** values = **2, 4, 4, 4, 4, 4, 4**
➢ Hemisphere:	Front hemisphere (in front of the ATC logo on the transmitter)

- Metal Distortion: Filter alpha = 12, Slope = 0, Offset = 0 and Sensitivity = 2.

DATA FORMAT

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

- ANGLES: Data record contains 3 rotation angles. See SHORT_ANGLES_RECORD, DOUBLE_ANGLES_RECORD
- POSITION: Data record contains X, Y, Z position of sensor. See SHORT_POSITION_RECORD, DOUBLE_POSITION_RECORD
- MATRIX: Data record contains 9-element rotation matrix. See SHORT_MATRIX_RECORD, DOUBLE_MATRIX_RECORD
- QUATERNION: Data record contains quaternion. See SHORT_QUATERNIONS_RECORD, DOUBLE_QUATERNIONS_RECORD
- TIME_STAMP and METAL DISTORTION status: Some data formats include a TIME_STAMP and/or a METAL_DISTORTION status field. See DOUBLE_POSITION_TIME_STAMP_RECORD, DOUBLE_POSITION_TIME_Q_RECORD

These data formats for each sensor can be set up using the SetSensorParameter command. See DATA_FORMAT_TYPE for all available data format combinations.

ANGLE ALIGN

It aligns sensor to reference direction. These parameters can be set up for each sensor using the SetSensorParameter command. The current setting of ANGLE ALIGN can be accessed using the GetSensorParameter command. See ANGLE_ALIGN for a full description of its meaning and usage.

FILTERS

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

AC Narrow Notch Filter: A 2 tap finite impulse response (FIR) notch filter applied to signals measured by the tracker's sensor to eliminate a narrow band of noise with sinusoidal characteristics. This filter can be selected/deselected and interrogated through the SetSensorParameter and GetSensorParameter function calls. See FILTER_AC_NARROW_NOTCH for a full description.

AC Wide Notch Filter: A 8 tap finite impulse response (FIR) filter applied to 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 you select this filter, it will lock the output data to the current position and orientation if a sudden large change in position or orientation is detected. See FILTER_LARGE_CHANGE for a full description of its meaning and use.

HEMISPHERE

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

METAL DISTORTION

This command outputs an accuracy degradation indicator. It is also known as the "quality" number. The metal distortion value is output in certain of the data record formats. See DOUBLE_POSITION_TIME_Q_RECORD, DOUBLE_ANGLES_TIME_Q_RECORD, DOUBLE_POSITION_ANGLES_TIME_Q_RECORD for examples. See also DATA_FORMAT_TYPE for a list of all data formats. Those format types containing "_Q_" indicate the presence of the "quality" value.

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

POINT

One data record is output from the selected driveBAY 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.

The rate at which records are transmitted when using the GetSynchronousRecord can be changed through use of the **Report Rate** Divisor. See the Configurable Settings section in Chapter 3 for details.

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.

SERIAL NUMBER

The sensor's serial number can be obtained by calling GetSensorParameter.

BOARD

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

SERIAL NUMBER

The board's serial number can be obtained by calling GetBoardParameter.

SOFTWARE REVISION NUMBER

The tracker's firmware version number is stored as a two-digit revision number. Use the GetBoardConfiguration command to access if for a specified board.

TRANSMITTER

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

REFERENCE_FRAME

Defines a new measurement reference frame. The new reference frame is provided as three angles describing the azimuth, elevation and roll angles. There is no offset component, and the reference frame remains centered on the transmitter. This parameter is changed or examined using the SetTransmitterParameter and GetTransmitterParameter function calls. See REFERENCE_FRAME.

XYZ_REFERENCE_FRAME

When the transmitter REFERENCE_FRAME is changed it will cause the azimuth, elevation and roll angles of all the sensors to change to a new reference frame. Note that it will not cause the x, y and z position coordinates to change unless the XYZ Reference Frame flag is set. This flag is changed and examined with the SetTransmitterParameter and the GetTransmitterParameter function calls. See XYZ_REFERENCE_FRAME.

SERIAL_NUMBER

The transmitter's serial number can be found by using GetTransmitterParameter.

SELECT_TRANSMITTER

This command allows us to turn on next transmitter or turn off the current transmitter. A full description of the operation is found at SELECT_TRANSMITTER.

System Initialization

The first operation that must be performed before the driveBAY can be used is initialization. This is performed calling the function InitializeBIRDSystem(). The call takes no parameters and returns no information except for a completion code. The only acceptable code is BIRD_ERROR_SUCCESS. All other codes are fatal errors that indicate either a condition that has prevented the system from initializing or a prevailing condition that disallows the system from completing the initialization.

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

Note: Initialization is an all-inclusive operation. Internally, the first task it performs is to enumerate the driveBAY 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: In order to use any 3D Guidance API calls, it is necessary to include the header file *ATC3DG.h*. The returned value errorCode must be declared as a variable of type *int*.

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

System Setup

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

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

```
#include “ATC3DG.h”               // needed for enumerated types and calls

int errorCode;

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


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

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

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

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

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

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

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

Another way to initialize the system is to use the RestoreSystemConfiguration() call. This together with the SaveSystemConfiguration() call provide a convenient way for the user to save the current state of the total system to an information file (.inf) and then use that file at a later time to re-initialize the system to that exact state. These calls allow the user to save or restore all settable parameters used by the system, sensors and transmitter. The following code fragment illustrates the usage of the RestoreSystemConfiguration() call.

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

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

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

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

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

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

Sensor Setup

The sensor setup involves selecting a data format, setting the filter and quality parameters, determining the sensor angle alignment and the hemisphere of operation. All of these parameters have an associated default value. The parameter only needs to be changed if the default is inappropriate. In most cases, the default filter and quality parameters will be found to provide adequate performance for most applications. Unless the sensor is going to be attached to something that would cause it to be tilted while in its reference position then the angle align parameters will not need to be changed. The hemisphere will need to be changed if the sensor is going to operate anywhere other than the forward hemisphere, which is the default. Typically the user will only have to set up the data format if something other than position/angles in double floating point is required. At a minimum, nothing needs be changed and the system will still operate successfully.

Note: 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;
typedefbool                              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 Position and Orientation Data

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

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

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

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

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

Error Handling

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

Usually an error code is fatal. In other words, the only acceptable response to a command is BIRD_ERROR_SUCCESS. If any other response is received then the command failed to complete, and the error code will inform the user of the reason why it failed. The function GetErrorText() can be used to generate a message string for output to a file or screen display describing in English the nature of the error code passed to this command. GetErrorText() is the only command that does not require the driveBAY to be initialized before it can be used.

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

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

3D Guidance API

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

3D Guidance API Functions

3D Guidance API Structures

3D Guidance API Enumeration Types

3D Guidance API Status/Error Bit Definitions

3D Guidance Initialization Files

3D Guidance API Functions

The following functions are used with driveBAY:

InitializeBIRDSystem

GetBIRDSystemConfiguration

GetTransmitterConfiguration

GetSensorConfiguration

GetBoardConfiguration

GetSystemParameter

GetSensorParameter

GetTransmitterParameter

GetBoardParameter

SetSystemParameter

SetSensorParameter

SetTransmitterParameter

SetBoardParameter

GetAsynchronousRecord

GetSynchronousRecord

GetBIRDError

GetErrorText

GetSensorStatus

GetTransmitterStatus

GetBoardStatus

GetSystemStatus

SaveSystemConfiguration

RestoreSystemConfiguration

CloseBIRDSystem

InitializeBIRDSystem

The **InitializeBIRDSystem** function resets and initializes the driveBAY 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 driveBAY tracker hardware was found by the host system. Verify that hardware is installed and is of the correct type.
BIRD_ERROR_ACCESSING_PCI_CONFIG	NOTE: This error is for a PCIBIRD system only. The error occurred in the PciBIRD PCI interface. There is a problem with the PCI configuration registers. If error is repeatable there is an unrecoverable hardware failure.
BIRD_ERROR_INVALID_DEVICE_ID	A device has been found that is not supported by this API dll. Verify 3D Guidance model installed.
BIRD_ERROR_FAILED_LOCKING_DEVICE	NOTE: This error is for a PCIBIRD system only. Driver could not lock PCIBIRD resources. Check that there is not another application using the hardware.
BIRD_ERROR_BOARD_MISSING_ITEMS	NOTE: This error is for a PCIBIRD system only. The required resources were not found defined in the configuration registers. Possible corrupt configuration. If error is repeatable there is an unrecoverable hardware failure.
BIRD_ERROR_INCORRECT_PLD	The PLD version on the driveBAY hardware is incompatible with this version of the API dll. Verify model installed.
BIRD_ERROR_COMMAND_TIME_OUT	DriveBAY 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	DriveBAY 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 driveBAY 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 driveBAY firmware initialization did not complete within 10 seconds. It is assumed the board is faulty or the firmware has hung up somewhere. If the error is repeatable there is an unrecoverable hardware failure.
BIRD_ERROR_UNRECOGNIZED_MODEL_STRING	The firmware is reporting a model string that is unrecognized by the API dll. This could be due to a hardware failure causing the model string data to be corrupted. A corrupted board EEProm may cause it or the board installed is of a type not recognized by the API dll. If the error is repeatable return to vendor.

Remarks

When this function is called, it will first reset the driveBAY. The function will then interrogate the board and determine its 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 board to reset and initialize internally. This function must be called first, before any other command can be sent

Requirements

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

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

GetTransmitterConfiguration

The **GetTransmitterConfiguration** will return a structure containing a TRANSMITTER_CONFIGURATION.

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

Parameters

transmitterID
[in] A valid 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 driveBAY 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.

Requirements

Windows: Requires Windows 2000 or later.
Header: Declared in ATC3DG.h

Library: Use ATC3DG.lib

GetSensorConfiguration

The **GetSensorConfiguration** will return a structure containing a SENSOR_CONFIGURATION.

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

Parameters

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

pSensorConfiguration
[out] Pointer to a SENSOR_CONFIGURATION structure that receives the information about the sensor.

Return Values

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

Value	Meaning
BIRD_ERROR_SUCCESS	No errors occurred. Call completed successfully
BIRD_ERROR_SYSTEM_UNINITIALIZED	The driveBAY 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.

Requirements

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

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 driveBAY 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: Requires Windows 2000 or later.
Header: Declared in ATC3DG.h

Library: Use ATC3DG.lib

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

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

GetTransmitterParameter

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

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

Parameters

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

parameterType
[in] Contains the parameter type to be returned in the buffer. Must be a valid enumerated constant of the type TRANSMITTER_PARAMETER_TYPE.

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

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

Return Values

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

Value	Meaning
BIRD_ERROR_SUCCESS	No errors occurred. Call completed successfully
BIRD_ERROR_SYSTEM_UNINITIALIZED	The driveBAY 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: Requires Windows 2000 or later.
Header: Declared in ATC3DG.h
Library: Use ATC3DG.lib

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

SetSystemParameter

The **SetSystemParameter** function allows an application to change basic driveBAY 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 driveBAY 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	driveBAY 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	driveBAY internal watchdog timer has elapsed. If this error is repeatable there is an unrecoverable hardware failure.

Remarks

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

Requirements

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

SetSensorParameter

The **SetSensorParameter** function allows an application to select and change specific characteristics of individual sensors in a driveBAY 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 driveBAY 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	driveBAY 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	driveBAY 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: Requires Windows 2000 or later.
Header: Declared in ATC3DG.h
Library: Use ATC3DG.lib

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

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

GetAsynchronousRecord

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

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

Parameters

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

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.

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 driveBAY 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	driveBAY on-board controller had insufficient time to execute the position and orientation algorithm. This frequently occurs because the driveBAY controller is being overwhelmed with user interface commands. Reduce the rate at which GetAsynchronousRecord is being called.
BIRD_ERROR_SENSOR_BAD	The attached sensor is not saturated but is exhibiting another unspecified problem that prevents it from operating normally. Use the GetSensorStatus function to determine the precise problem.
BIRD_ERROR_INVALID_DEVICE_ID	The transmitterID passed was out of range for the system.
BIRD_ERROR_COMMAND_TIME_OUT	driveBAY 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	driveBAY internal watchdog timer has elapsed. It is necessary to re-initialize the system to recover from this error. If this error is repeatable there is an unrecoverable hardware failure.

Remarks

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

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

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

Requirements

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

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 that is otherwise OK has gone into saturation. This may occur if the sensor is too close to the transmitter or if the sensor is too close to metal or an external magnetic field.
BIRD_ERROR_CPU_TIMEOUT	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 trakSTAR will transmit 80 *3 =240 Updates/sec (1 record every 4mS) for each sensor. Changing the Report Rate setting to 4 will reduce the number of records to 240/4 = 60 Updates/sec (1 record every 17mS) for each sensor. The default Report Rate setting of 1 makes all outputs computed by the tracker available. See the Configurable Settings section in Chapter 3 for details on changing the default setting.

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

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

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

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

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

Requirements

Windows: 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 driveBAY hardware and system has not been initialized yet. The InitializeBIRDSystem function must be called first.
BIRD_ERROR_COMMAND_TIME_OUT	driveBAY 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	driveBAY internal watchdog timer has elapsed. It is necessary to re-initialize the system to recover from this error. If this error is repeatable there is an unrecoverable hardware failure.

Remarks

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

Requirements

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

GetErrorText

The **GetErorText** 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 driveBAY 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: Requires Windows 2000 or later.
Header: Declared in ATC3DG.h
Library: Use ATC3DG.lib

GetSensorStatus

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

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

Parameters

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

Return Values

The function returns a value of type *DEVICE_STATUS*. The returned value contains the status of the selected sensor channel. The bits in the status word have the following meanings:

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

Remarks

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

No error codes are returned so it is not possible to determine if the call was successful through the standard process of inspecting the returned error code. But there are 2 possible runtime error conditions:
1) Calling the function before InitializeBIRDSystem has been called
2) Calling the function with an invalid sensor ID.

Both these conditions have been taken care of by the addition of bits 5 and 6 in the status word.
1) If the function InitializeBIRDSystem has not been called then the "UnInitialized" bit will be set.
2) If this function call was made with an invalid (out of range) sensor ID then the "Non-Existent" bit will be set.

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

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

Requirements

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

GetTransmitterStatus

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

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

Parameters

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

Return Values

The function returns a value of type *DEVICE_STATUS*. The returned value contains the status of the selected transmitter channel. The bits in the status word have the following meanings:

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

Remarks

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

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

1) Calling the function before 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.

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

Requirements

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

GetBoardStatus

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

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

Parameters

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

Return Values

The function returns a value of type *DEVICE_STATUS*. The returned value contains the status of the selected unit. The bits in the status word have the following meanings:

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

Remarks

This function takes as its only parameter an index to a selected 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.

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

Requirements

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

GetSystemStatus

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

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

Remarks

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

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

Requirements

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

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 driveBAY hardware and system has not been initialized yet. The InitializeBIRDSystem function must be called first.
BIRD_ERROR_UNABLE_TO_CREATE_FILE	The call was unable to complete for some unspecified reason. Check the format of the file name string.
BIRD_ERROR_CONFIG_INTERNAL	Internal error in configuration file handler. Report to vendor.

Remarks

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

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

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

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

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

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

The file format is described in 3D Guidance Initialization Files section.

Requirements

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

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 driveBAY 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 %windir%\inf directory. If the file is not found there then a BIRD_ERROR_UNABLE_TO_OPEN_FILE error is generated.

The configuration that is restored contains all of the parameters that can be alternatively initialized using the SetSystemParameter, SetSensorParameter and SetTransmitterParameter function calls. The parameters that can be initialized with each of these calls are listed in the enumerated types SYSTEM_PARAMETER_TYPE, SENSOR_PARAMETER_TYPE and TRANSMITTER_PARAMETER_TYPE.

The file format is described in 3D Guidance Initialization Files

Requirements

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

CloseBIRDSystem

The **CloseBIRDSystem** function closes the driveBAY 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 driveBAY 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 driveBAY system in order to prevent memory and/or resource leaks.

Requirements

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

3D Guidance API Structures

The following structures are used with the driveBAY.

SYSTEM_CONFIGURATION
SENSOR_CONFIGURATION
TRANSMITTER_CONFIGURATION
BOARD_CONFIGURATION
ADAPTIVE_PARAMETERS
QUALITY_PARAMETERS
VPD_COMMAND_PARAMETER
POST_ERROR_PARAMETER
DIAGNOSTIC_TEST_PARAMETER
BOARD_REVISIONS

Data record structures:
SHORT_POSITION_RECORD
SHORT_ANGLES_RECORD
SHORT_MATRIX_RECORD
SHORT_QUATERNIONS_RECORD
SHORT_POSITION_ANGLES_RECORD
SHORT_POSITION_MATRIX_RECORD
SHORT_POSITION_QUATERNION_RECORD
DOUBLE_POSITION_RECORD
DOUBLE_ANGLES_RECORD
DOUBLE_MATRIX_RECORD
DOUBLE_QUATERNIONS_RECORD
DOUBLE_POSITION_ANGLES_RECORD
DOUBLE_POSITION_MATRIX_RECORD
DOUBLE_POSITION_QUATERNION_RECORD
DOUBLE_POSITION_TIME_STAMP_RECORD
DOUBLE_ANGLES_TIME_STAMP_RECORD
DOUBLE_MATRIX_TIME_STAMP_RECORD
DOUBLE_QUATERNIONS_TIME_STAMP_RECORD
DOUBLE_POSITION_ANGLES_TIME_STAMP_RECORD
DOUBLE_POSITION_MATRIX_TIME_STAMP_RECORD
DOUBLE_POSITION_QUATERNION_TIME_STAMP_RECORD
DOUBLE_POSITION_TIME_Q_RECORD
DOUBLE_ANGLES_TIME_Q_RECORD
DOUBLE_MATRIX_TIME_Q_RECORD
DOUBLE_QUATERNIONS_TIME_Q_RECORD
DOUBLE_POSITION_ANGLES_TIME_Q_RECORD
DOUBLE_POSITION_MATRIX_TIME_Q_RECORD
DOUBLE_POSITION_QUATERNION_TIME_Q_RECORD
SHORT_ALL_RECORD

DOUBLE_ALL_RECORD
DOUBLE_ALL_TIME_STAMP_RECORD
DOUBLE_ALL_TIME_STAMP_Q_RECORD
DOUBLE_ALL_TIME_STAMP_Q_RAW_RECORD

SYSTEM_CONFIGURATION

The **SYSTEM_CONFIGURATION** structure contains the system information.

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

Members

measurementRate
Indicates the current measurement rate of the tracking system. Default is 80.0 Hz.

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

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

agcMode
Enumerated constant of the type: AGC_MODE_TYPE. Setting the mode to SENSOR_AGC_ONLY disables the normal transmitter power level switching.

numberBoards
Indicates the number of units 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: 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 unit for this transmitter channel.

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

type
Contains a value of enumerated type DEVICE_TYPES.

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

Requirements

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

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.

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

BOARD_CONFIGURATION

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

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

Members

serialNumber
The serial number of the board.

type
The board type. The type is of the enumeration type BOARD_TYPES.

revision
The board ECO revision number.

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

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

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

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

modelString[10]
Each board has a configuration EEProm. Contained in the EEProm are the calibration values belonging to the board. Also contained in the EEProm is a "model string" which is used to identify the board type. The modelString is a 10-character array, which contains the "model string". The string is not null-terminated. For example, the 8mm sensor driveBAY electronics unit will have the string "6DBB4   ". The string is padded with space characters to the end of the buffer.

Requirements

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

QUALITY_PARAMETERS

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

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

Members

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

error_offset
This value is the offset of the inherent system error.

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

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

Requirements

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

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()**).

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.

```
    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.

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 2.

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 2.

pTestName

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

testNameLen

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

diagError

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

GetSystemParameter() and SetSystemParameter() Diagnostic Test Usage

Call	Action	Input Param.		Output Parameter			
		Suite	Test	Suites	Tests	TestName	DiagError
GetSystemParameter()	Get number of test suites available.	0	0 (Don't care)	Total number of suites available. If no diagnostics are supported this value will be zero.	Don't care	Don't Care	Don't Care.
GetSystemParameter()	Get number of tests available in selected suite.	N	0	Suite number N is returned	Total number of tests available in the suite	32 character zero terminated string containing the **Suite Name**	Error* if N > Total number of available suites.
GetSystemParameter(	Get string name of individual	N	M	Suite number N is returned	Test number M is returned.	64 character zero terminated string containing the	Error* if N > Total number of available

)	test.					**Suite/Test Name**.	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 2

* 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

BOARD_REVISIONS

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

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

Members

boot_loader_sw_number
Major revision number for the boot loader.

boot_loader_sw_revision
Minor revision number for the boot loader.

mdsp_sw_number
Major revision number for the acquisition DSP.

mdsp_sw_revision
Minor revision number for the acquisition DSP.

nondipole_sw_number
Major revision number for the nondipole DSP.

nondipole_sw_revision
Minor revision number for the nondipole DSP.

fivedof_sw_number
Major revision number for the 5DOF DSP.

fivedof_sw_revision
Minor revision number for the 5DOF DSP.

sixdof_sw_number
Major revision number for the 6DOF DSP.

sixdof_sw_revision
Minor revision number for the 6DOF DSP.

dipole_sw_number
Major revision number for the dipole DSP.

dipole_sw_revision
Minor revision number for the dipole DSP.

Requirements

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

See Also

SHORT_POSITION_RECORD

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

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

Members

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

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

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

Remarks

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

Requirements

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

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

SHORT_MATRIX_RECORD

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

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

Members

s[3][3]

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

Remarks

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

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

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

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

COS(Y)*COS(Z)	**COS(Y)*SIN(Z)**	**-SIN(Y)**
-(COS(X)*SIN(Z)) **+(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: Requires Windows 2000 or later.
Header: Declared in ATC3DG.h
Library: Use ATC3DG.lib

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

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: 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: 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: 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. See the default reference frame for the positive X, Y and Z directions.

Requirements

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

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

DOUBLE_MATRIX_RECORD

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

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

Members

s[3][3]
This is a 3x3 array of values. Each value is in the range +0.99997 to –1.00000

Remarks

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

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

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

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

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

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

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

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

Matrix information is 0 when sensor saturation occurs.

Requirements

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

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

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: 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: 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: 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: 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: 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: 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: 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: 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: 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: 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: 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: 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: 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: 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: 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: 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: 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: 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: 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: 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: 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: 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

3D Guidance API Enumeration Types

The following enumeration types are used with the driveBAY.

BIRD_ERROR_CODES

MESSAGE_TYPE

TRANSMITTER_PARAMETER_TYPE

SENSOR_PARAMETER_TYPE

BOARD_PARAMETER_TYPE

SYSTEM_PARAMETER_TYPE

HEMISPHERE_TYPE

AGC_MODE_TYPE

DATA_FORMAT_TYPE

BOARD_TYPES

DEVICE_TYPES

BIRD_ERROR_CODES

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

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

Enumerator Value	Meaning
BIRD_ERROR_SUCCESS=0	No errors occurred. Call completed successfully
BIRD_ERROR_PCB_HARDWARE_FAILURE	The driveBAY 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	driveBAY 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 driveBAY hardware was found by the host system. Verify that hardware is installed and is of the correct type.
BIRD_ERROR_ACCESSING_PCI_CONFIG	NOTE: This error is for a PCIBIRD system only. The error occurred in the pciBird PCI interface. There is a

	problem with the PCI configuration registers. If error is repeatable there is an unrecoverable hardware failure.
BIRD_ERROR_INVALID_DEVICE_ID	The device ID passed was out of range for the system.
BIRD_ERROR_FAILED_LOCKING_DEVICE	NOTE: This error is for a PCIBIRD system only. Driver could not lock PCI/miroBIRD resources. Check that there is not another application using the hardware.
BIRD_ERROR_BOARD_MISSING_ITEMS	NOTE: This error is for a PCIBIRD system only. The required resources were not found defined in the PCI configuration registers. Possible corrupt configuration. If error is repeatable there is an unrecoverable hardware failure.
BIRD_ERROR_NOTHING_ATTACHED	<obsolete>
BIRD_ERROR_SYSTEM_PROBLEM	<obsolete>
BIRD_ERROR_INVALID_SERIAL_NUMBER	<obsolete>
BIRD_ERROR_DUPLICATE_SERIAL_NUMBER	<obsolete>
BIRD_ERROR_FORMAT_NOT_SELECTED	<obsolete>
BIRD_ERROR_COMMAND_NOT_IMPLEMENTED	This function has not been implemented yet.
BIRD_ERROR_INCORRECT_BOARD_DEFAULT	An unexpected response was received from the controller on the driveBAY 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	driveBAY 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 driveBAY 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 driveBAY hardware is incompatible with this version of the API dll. Verify driveBAY 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 that 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	driveBAY on-board controller had insufficient time to execute the position and orientation algorithm. This frequently occurs because driveBAY 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 that is unrecognized by the API dll. This could be due to a hardware failure causing the model string data to be corrupted. 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_MAXIMUM_VALUE	Final error code place holder

Requirements

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

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
};
```

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 1), and are measured with respect to rotations about the physical X, Y and Z axes of the sensor (Figure 2). The ANGLE ALIGN parameter allows you to mathematically change the sensor's X, Y and Z-axes to an orientation, which differs from that of the actual sensor. For example: Suppose that during installation you find it necessary, due to physical requirements, to rotate the sensor, resulting in its angle outputs reading Azim = 5 deg, Elev = 10 and Roll = 15 when it is in its normal "resting" position. To compensate, use the ANGLE_ALIGN parameter, passing as values 5, 10 and 15 degrees. After this sequence is sent, the sensor outputs will be zero, and orientations will be computed as if the sensor were not misaligned. See default reference frame for transmitter and sensor default reference frames.

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 sensor. Reducing the amount of filtering results in less filter lag. Unfortunately, electrical noise in the environment—when measured by the sensor—also makes it look like the sensor is undergoing a translation and rotation. As the sensor moves farther and farther away from the transmitter, the amount of noise measured by the sensor appears to increase because the measured transmitted signal level is decreasing and the sensor amplifier gain is increasing. In order to decide if the amount of filtering should be reduced, the tracker has to know if the measured rate is a real sensor rate due to movement or a false rate due to noise. The tracker gets this knowledge by the user specifying what the expected noise levels are in the operating environment as a function of distance from the transmitter. These noise levels are the 7 words that form the Vm table. The Vm values can range from 1 for almost no noise to 32767 for a lot of noise.
FILTER_LARGE_CHANGE	When the LARGE_CHANGE filter is selected, the position and orientation outputs are not allowed to change if the system detects a sudden large change in the outputs. Large undesirable changes may occur at large

	separation distances between the transmitter and sensor when the sensor undergoes a fast rotation or translation. If the LARGE_CHANGE value is TRUE the outputs will not be updated if a large change is detected. If value is FALSE, the outputs will change.
QUALITY	This data structure is used to adjust the output characteristics of the Quality number. Also referred to as the METAL error or Distortion number, this value is returned with certain data formats and gives the user an indication of the degree to which the position and angle measurements are in error. This error may be due to 'bad' metals located near the transmitter and sensor, or due to TRACKER 'system' errors. ‘Bad’ metals are metals with high electrical conductivity such as aluminum, or high magnetic permeability such as steel. ‘Good’ metals have low conductivity and low permeability such as 300 series stainless steel, or titanium. The QUALITYerror number also reflects tracker ‘system’ errors resulting from accuracy degradations in the transmitter, sensor, or other electronic components. It will represent a level of accuracy degradation resulting from either movement of the sensor or environmental noise. A very small QUALITYerror number indicates no or minimal position and angle errors depending on how sensitive you have set the error indicator. A large QUALITYerror number indicates maximum error for the sensitivity level selected. Users of the QUALITYerror number will find that as a metal detector, it is sensitive to the introduction of metals in an environment where no metals were initially present. This metal detector can fool you, however, if there are some metals initially present and you introduce new metals. It is possible for the new metal to cause a distortion in the magnetic field that reduces the existing distortion at the sensor. When this occurs you’ll see the Error value initially decrease, indicating less error, and then finally start increasing again as the new metal causes more distortion. **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 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 (i.e. 36"). Since its difficult to go from range =0 to max., you might just translate over say half the distance and double the error value change you measure. **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 driveBAY 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.

Requirements

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

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

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,
};
```

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 (Figure 1). 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. See default reference frame for transmitter and sensor default reference frames.
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.

Requirements

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

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
};
```

Enumerator Value	Meaning
SERIAL_NUMBER_PCB	Returns the serial number of the unit's board.

Requirements

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

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,
    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<110.0
MAXIMUM_RANGE	Sets the system maximum range. The parameter passed is a double value representing the maximum range in any of the 3 axes in inches. There is only one valid range and that is 36.0 inches.
METRIC	Enables/disables metric position reporting. The parameter passed is a bool. If the value is true then metric reporting is selected otherwise if the value is false then metric reporting is turned off. Metric data is reported in millimeters. Non-metric data is reported in inches.
END_OF_LIST	Final system parameter type place holder

Requirements

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

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

AGC_MODE_TYPE

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

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

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

Requirements

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

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,
    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
MAXIMUM_FORMAT_CODE	End of table place holder

Requirements

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

BOARD_TYPES

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

```
enum BOARD_TYPES{
   PCIBIRDII
};
```

Enumerator Value	Meaning
PCIBIRDII	<obsolete>
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

Requirements

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

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 that are returned from a call to either GetSensorConfiguration or GetTransmitterConfiguration.

```
enum DEVICE_TYPES{
    STANDARD_SENSOR,
    TYPE_800_SENSOR,
    STANDARD_TRANSMITTER,
    MINIBIRD_TRANSMITTER,
    SMALL_TRANSMITTER,
    TYPE_500_SENSOR
};
```

Enumerator Value	Meaning
STANDARD_SENSOR	Standard Flock sensor with miniDIN connector
TYPE_800_SENSOR	8mm sensor with miniDIN connector (miniBIRD II 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 micro sensors
UNKNOWN_SENSOR	default
UNKNOWN_TRANSMITTER	default
TYPE_800_BB_SENSOR	driveBAY/trakSTAR sensor
TYPE_800_BB_STD_TRANSMITTER	driveBAY/trakSTAR mid-range TX
TYPE_800_BB_SMALL_TRANSMITTER	driveBAY/trakSTAR short-range TX

Requirements

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

3D Guidance API Status/Error Bit Definitions

The following bit definitions are used with the driveBAY.

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 = 3D Guidance board error 010 = Sensor error 100 = Transmitter error Note: All other source codes are invalid
30 - 31	Bits 30 and 31 provide the following advisory error level code Note: It is recommended that all error messages be resolved before proceeding. 00 = Warning 01 = Warning 10 = Fatal Error

DEVICE_STATUS

The **DEVICE_STATUS** is a *typedef* for an *unsigned long* (32 bits) and has the following error bit definitions:

Bit	Name	Meaning	S	B	R	T
0	GLOBAL_ERROR	Global error bit. If any other the other error status bits are set then this bit will be set.	x	x	x	x
1	NOT_ATTACHED	No physical device attached to this device channel.			x	x
2	SATURATED	Sensor currently saturated.			x	
3	BAD_EEPROM	PCB or attached device has a corrupt or unresponsive EEprom		x	x	x
4	HARDWARE	Unspecified hardware fault condition is preventing normal operation of this device channel, board or the system.	x	x	x	x
5	NON_EXISTENT	The device ID used to obtain this status word is invalid. This device channel or board does not exist in the system.		x	x	x
6	UNINITIALIZED	The system has not been initialized yet. The system must be initialized at least once before any other commands can be issued. The system is initialized by calling InitializeBIRDSystem	x	x	x	x
7	NO_TRANSMITTER	Either an invalid system – transmitter command was issued or an attempt was made to call GetAsynchronousRecord when no transmitter was running.	x	x	x	
8	BAD_12V	The +12V power supply has not been connected to this transmitter channel or to this board. In the case of the system error it indicates there is no +12V anywhere in the system and the system cannot run.	x	x		x
9	CPU_TIMEOUT	CPU ran out of time while executing the position and orientation algorithm.		x	x	
10	INVALID_DEVICE	Invalid sensor or transmitter has been attached to this sensor/transmitter channel.			x	x
11 - 31	<reserved>	Always returns zero.	x	x	x	x

The 4 columns with the headings S, B, R and T indicate whether or not the bits are applicable depending on which device status is being acquired. S = system, B = board, R = sensor and T = transmitter.

3D Guidance Initialization Files

The Initialization File is used to set a driveBAY 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 driveBAY 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 driveBAY 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 driveBAY 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 3D Guidance driveBAY boards installed in the PC. This number must match the current number of boards 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 that may have either one of two values. The valid settings are YES or NO. When the value YES is selected the position data will be output with millimeter dimensions. If the value is set to NO the output will be in inches.

power
This parameter is a floating point value representing the AC power line frequency in Hz. Currently only two values are valid. These are 50 and 60 Hz.

mode
This parameter is a string describing the AGC mode to be used for the system.

range
This parameter is a floating point value representing the maximum range that the system will report in inches. Currently the only valid value is 36 (inches).

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

```
[System]
```

```
NumberOfBoards=1
TransmitterIDRunning=0
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 –256 and +256. (Default is 164) The offset should have a value between –127 and +127. (The default is 0) The sensitivity should have a value between 0 and +127 (Default is 32) and the alpha should have a value between 0 and 512. (The default is 327)

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

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

[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

5

Chapter 5: Troubleshooting

Troubleshooting

Most installation and tracking problems are easy to fix. Consult our troubleshooting table for common problems and their solutions. If you continue to experience problems, contact us for technical support.

Symptom	Possible Causes	Solution
No front panel LED illumination	No power	-Check AC connections to power supply -Reset hardware by cycling AC power
Demo Utility doesn't run	No serial communication Software installation unsuccessful	-Check the suggestions outlined in 'Not able to communicate' below -Re-initialize the HOST PC and run the installation again
Power-up defaults did not change after configuring with the utility	Configuration download interrupted driveBAY did not reset(restart) after burning the Flash settings	-Re-start the driveBAY 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, and re-check the settings

Symptom	Possible Causes	Solution
Not able to communicate with the system using USB	Re-initialize USB	-Unplug and replug USB cable on driveBAY. -Cycle power on the driveBAY
	Driver not installed	-Check installation/status of driveBAY USB driver in Windows Device Manager. Re-install if necessary
Can communicate with the system, but data not changing	Sensor is saturated	-Check the error codes for a sensor a saturation condition. Move the sensor farther away from the Transmitter.
	Transmitter is OFF or disconnected	Check status of the Transmitter. Turn ON using SELECT_TRANSMITTER parameter (3D Guidance API) or RUN command (Flock protocol).
	Component connections faulty	-Check that Sensor/Transmitter connectors are correctly installed. Inspect pins for wear or damage. -Contact Ascension for assistance
Data is too noisy	Filters OFF	-Check FILTER STATUS for present state of filter configuration.
	Low Signal	-Decrease distance from sensor to Transmitter.
	External noise in environment	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
	Changing Hemisphere	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.
	Line frequency value set incorrectly.	Determine correct frequency (50 Hz in Europe, 60 Hz in North America) and set to correct value.

Symptom	Possible Causes	Solution
Poor accuracy	Metal in tracking environment.	Check all around the transmitter to the furthest distance from the center of the transmitter to the maximum distance the sensor is used. Move or replace metal, or reposition the driveBAY system.
	Damaged equipment.	Check for damage.
	Sensor or transmitter connector not properly inserted.	Correct by unplugging and plugging the components back into the board.
	A software application error.	Verify formulas and scale factors.

Error Codes

Chapter 4: 3D Guidance API Reference provides a complete listing of all 3D Guidance API error codes via the USB interface.

LED Definitions

LED State	Description of Board Status	Possible Error Conditions resulting in LED state:
OFF	Power applied to board.	a) PLD un-programmed b) PLD error c) LED faulty d) Board error e) Board un-powered.
SOLID RED	PLD loaded and configured.	a) Flash memory un-programmed b) POSERVER DSP error c) Board error
SOLID ORANGE	POSERVER DSP - boot loader successfully loaded and running. All preliminary tests completed.	a) Multi-Boot Loader test failed b) POSERVER main application not found c) Code corrupt? d) DSP error e) Board error
SLOW BLINKING RED (< 1Hz)	POSERVER DSP - application loaded, running and Command Handler executing	a) MDSP application not found in Flash b) Communications failure c) Failed starting MDSP
FAST BLINKING RED (> 2Hz)	MDSP DSP - code downloaded by POSERVER	a) MDSP failed to respond to messages b) MDSP code corrupt c) MDSP error d) Board error
SLOW BLINKING ORANGE (< 1Hz)	POSERVER successfully communicated with MDSP	a) System EEProm error b) PLD version error c) MDSP fails to respond after tests d) MDSP test failed

FAST BLINKING ORANGE (> 2Hz)	All of POSERVER initialization and power on test is complete	a) Transmitter not present b) Transmitter EEProm error c) Transmitter hardware failure
SLOW BLINKING GREEN (< 1Hz)	System is fully functional. The transmitter **is not** being energized during acquisition. (ASLEEP)	a) System ASLEEP – normal b) If auto-config issued then abnormal
SOLID GREEN	System is fully functional. The transmitter **is** being energized during the acquisition. (AWAKE)	a) System AWAKE – normal b) If SLEEP command issued then abnormal.
FAST BLINKING RED/ORANGE (> 2Hz)	Run-time Diagnostic Error	a) One of the run-time diagnostics has failed. b) Primary suspect is a removed transmitter. (This shows up as a TX Temp Sense error.) c) Secondary suspects might be TX Temp. Sense or EU Temp Sense (over-heating problem)

Chapter

6

Chapter 6: Maintenance, Repair and Disposal

Taking care of your driveBAY tracker is simple and straightforward. For years of accurate operation, be sure to treat the components as delicate electronic components.

The parts of the tracker that are physically handled are subject to the most wear. With proper handling and care the electronics unit, sensor and transmitter should indefinitely -- well beyond our warranty period.

User Maintenance

driveBAY requires minimal maintenance. You should do the following to maintain good performance:

Maintenance Prior to Each Use

1. Check the transmitter and sensor cables for nicks and cuts in the insulation. If nicks or cuts are found, the component should be replaced after proper disposal.

2. Inspect component connectors and receptacles for bent or damaged pins or other obstructions.

3. Inspect the transmitter for cracks or exterior damage. If transmitter is cracked or interior of the transmitter is exposed in some way, the component should be replaced after proper disposal.

Periodic Maintenance (As needed)

1. Inspect USB and power connections to ensure positive contact.

2. Inspect to ensure driveBAY is tightly seated into the drive bay slot.

3. Transmitter and sensors are properly mounted as recommended in Mounting Hardware.

Cleaning and Disinfecting

Periodically, clean the equipment (electronics unit, transmitter, sensor, and cables) by wiping down with a cloth dampened in a cleaning solution such as mild soap and water, isopropyl alcohol or a similar acceptable cleaning solution. If the tracker's components come in contact with biological fluid or tissue, be sure to follow your organization's procedures for proper cleaning and disinfection. The electronics unit, transmitters and sensors are not designed to withstand autoclaving or gamma radiation. Sensors are ETO compatible. Do not immerse the electronics unit, transmitter, sensor, or cables in liquids. Components are not waterproof.

Sensor Sterilization

3D Guidance driveBAY sensor materials are tolerant of both cold sterilant (Cidex or equivalent) and Ethylene Oxide (EtO) gas sterilization processes.

However, even if embedded in a medical instrument such as an endoscope or other non-disposable tool, the electronics portion of the sensors should never be subject to autoclaving or gamma radiation. The electronics portion of the sensors resides within the connector.

BROAD GUIDELINES WHEN CONSIDERING THE CIDEX (GLUTARALDEHYDE) PROCESS

Warning: Never use this sterilization process without first consulting the manufacturer's instructions for proper and safe use. Ascension cannot determine appropriate minimum dosages since driveBAY components are always part of a larger medical device. Degree of sterilization is a function of the type of procedure undertaken and the device manufacturers' specifications. In all cases, institutional protocols should be strictly followed.

When considering the use of Cidex, you should:

- Use Cidex classified as "sterilant." A Cidex products classified as "disinfectants" are not adequate.

- Take into account the physical properties of the medical instrument being sterilized: It must be clean, relatively smooth, impervious to moisture, and be of a shape that permits all surfaces to be exposed to the sterilant.

- Ensure the medical instrument receives adequate exposure. All surfaces, both interior and exterior, should be exposed to the sterilant. Tubing must be completely filled and the materials to be sterilized must be clean and arranged in the sterilant to assure total immersion.

- Use fresh solutions. The sterilant solution should be clean and fresh. Most sterilants come in solutions consisting of two parts to form an "activated" solution. The shelf life of activated solutions is indicated in the instructions for commercial products. Generally, this is from one to four weeks.

- Rinse chemically sterilized items. Instruments, implants, and tubing (both inside and out) must be rinsed with sterile saline or sterile water prior to use to avoid tissue damage.

BROAD GUIDELINES WHEN CONSIDERING THE ETO STERILIZATION PROCESS

Warning: Never use this sterilization process without first consulting the manufacturer's instructions for proper and safe use. Ascension cannot determine appropriate minimum dosages since driveBAY components are always part of a larger medical device that is sterilized. Degree of sterilization is a function of the type of procedures undertaken and the device manufacturers' specifications. In all cases, institutional protocols should be strictly followed.

EtO is a long-established and widely used hospital method of sterilization. Its low - temperature environment is compatible with electronic devices, such as the driveBAY sensor assembly. Gas sterilization with ethylene oxide requires the use of an approved gas sterilizer and appropriate monitoring systems to assure sterility and personnel safety. Ethylene gas is irritating to tissue; all materials require appropriate airing time.

Cycle protocols should be implemented in accordance with EN 550 /ISO 11135, "Sterilization of Healthcare Products." This document describes Ethylene oxide and requirements for development, validation, and routine control of a sterilization process for medical devices.

Software Updates

As new features or updates become available for the driveBAY, you may find it necessary to update the firmware stored in the electronic unit's memory. The *driveBAY Utility* included on your CD-ROM allows you to do this without opening the electronics or returning it to Ascension. Instructions for this procedure are included with updates.

Repair

There are no user level repairs that can be made on the electronics unit, transmitters, or sensors. If you have a problem with any part of your tracker, please contact Ascension Technical Support.

Technical Support contact points are as follows:

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

Warranty

Ascension warrants that its products are free from defects in material and workmanship for a period of one (1) year from date of delivery, providing they are not subject to misuse, neglect, accident, incorrect installation, or improper care. If any Ascension products fail due to no fault of the buyer, Ascension will (at its option) either repair the defective product and restore it to normal operation without charge for parts and labor or provide a replacement in exchange for the defective product. Repair work shall be warranted for the remainder of the unexpired warranty period or for a period of 60 days, whichever is longer. This warranty is the exclusive warranty given in lieu of any other express or implied warranty. Ascension disclaims any implied warranties of merchantability and fitness to a particular purpose. Warranties are voided if the buyer utilizes a power supply that does not strictly adhere to Ascension's electrical power requirements, changes the configuration of a tracker, (such as, adding extensions to cables or modifying boards), or mishandles sensors or cables. To avoid warranty issues, customers should carefully adhere to all product advisories. Transmitter and sensor cables and connectors are sensitive electronic components and should be treated with care. Do not drop, pull, twist, or mishandle cables.

Disposal

The European Union has issued a directive, known as the WEEE (Waste Electrical and Electronic Equipment) Directive, to protect the quality of our environment by reducing the amount of electrical equipment waste buried in landfills. WEEE focuses on the recycling and reuse of "equipment that depends on an electronic current or an electromagnetic field to operate and as equipment for the generation, transfer and measurement of such currents and fields." Although none of the driveBAY'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.

Chapter

7

Chapter 7: Regulatory Information, Symbols and Product Specifications

In accordance with EN60601-1 (Medical electrical equipment – general requirements for safety), this equipment can be classified as:

Class I (only when installed in a Class I chassis)
Type B Applied Part
Not AP/APG

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.

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.

CE Conforms with all health, safety and environmental protection standards of the European Union

⚠ 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.

<u>Warning: This tracker does not have approval from the FDA for patient contact applications</u>

EC Declaration of Conformity
Issued by

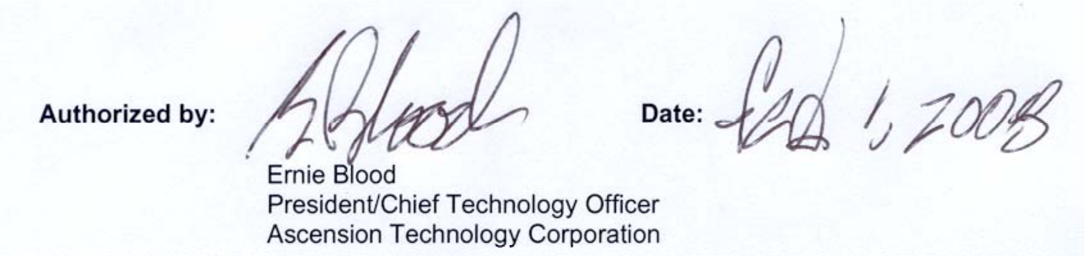
ASCENSION TECHNOLOGY CORPORATION
PO Box 527
Burlington, VT 05402 USA
802-893-6657

Equipment Description: 3D Guidance driveBAY
3D Guidance driveBAY Tracking System

Applicable Directive: 93/42/EEC, Medical Devices

Applicable Standards: EN 60601-1 : 1997
Medical Electrical Equipment: Part 1: General Requirements for Safety

EN 60601-1-2 : 2001
Medical Electrical Equipment: Part 1: General Requirements
for Safety –2. Collateral Standard: Electromagnetic Compatibility- Requirements and Test



Authorized by: EBlood **Date:** Feb 1, 2008

Ernie Blood
President/Chief Technology Officer
Ascension Technology Corporation

FCC Compliance Statement

Radio and television interference

Warning: Changes or modifications to this unit not expressly approved by the party responsible for compliance could void the user's authority to operate the equipment.

Note: This equipment has been tested and found to comply with the limits for a Class B digital device, pursuant to Part 15 of the FCC rules. These limits are designed to provide reasonable protection against harmful interference in a residential installation. This equipment generates, uses, and can radiate radio frequency energy and, if not installed and used in accordance with the instructions, may cause harmful interference to radio communications. However, there is no guarantee that interference will not occur in a particular installation. If this equipment does cause harmful interference to radio or television reception, which can be determined by turning the equipment off and on, the user is encouraged to try and correct the interference by one or more of the following measures:

- Reorient or relocate the receiving antenna.
- Increase the separation between the equipment and receiver.
- Connect the equipment into an outlet on a circuit different from that to which the receiver is connected.
- Consult the dealer or an experienced radio/TV technician for help.

Declaration of Conformity

Model Number:	**3D Guidance driveBAY**
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:
(see note below)
1.4 mm RMS position
0.5 degree RMS orientation

Update rate: Short and mid-range transmitters up to 375 updates/second.

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.

Outputs: X, Y, Z positional coordinates, orientation angles, orientation matrix and quaternion.

Interface: USB

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)

Model 800 Sensor:
Sensor max OD 8.0mm
Sensor max length 20mm
Cable max OD 3.8mm
Cable length: 3 meters

Electronics Unit
Dimensions (L x W x H): 17.7 cm x 14.7 cm x 4.1 cm
Weight: 1.28 Kg

Power:	+12VDC: 2.9A (max.), 1.6A (nom.) +5VDC: 600mA (nom.)
Operating temperature:	5°C to 40°C, 10% - 90% non-condensing humidity
Warm up:	System shall meet accuracy specifications after 5 minutes.

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 Utility

Running the driveBAY Utility

Setup

Before changing settings or beginning an upgrade, setup the tracker with the USB interface.

Note: This Utility can be run with either the driveBAY or trakSTAR systems.

1. Connect the USB cable and the Power cable to the rear panel of the tracker.

2. Connect the other end of the USB Cable to the header on the motherboard or an internal USB port in the host PC.

3. Power on the Tracker. (Done by powering on the PC)

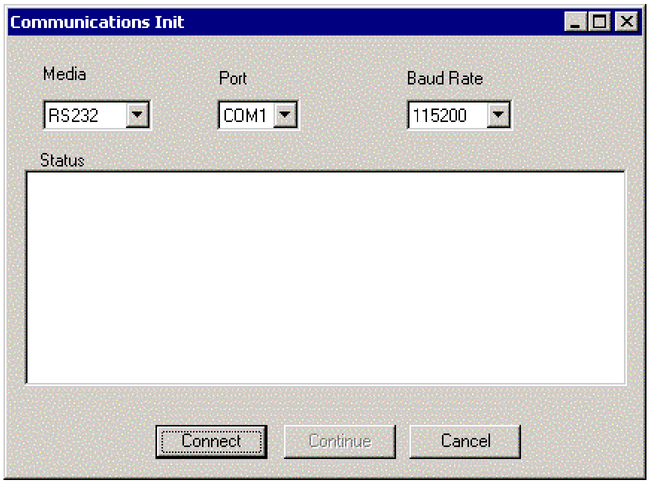
Run the Utility

4. Start the utility by running the 'driveBAY Utility.exe' file located on the CD-ROM.

This will open the interface configuration window of the Utility.

5. Select 'USB' from the 'Media' pull down menu and click Connect.

This will establish communication with the tracker, and initiate a reading of the current configuration. Progression of the reading is shown in the Status window.



Communications Init

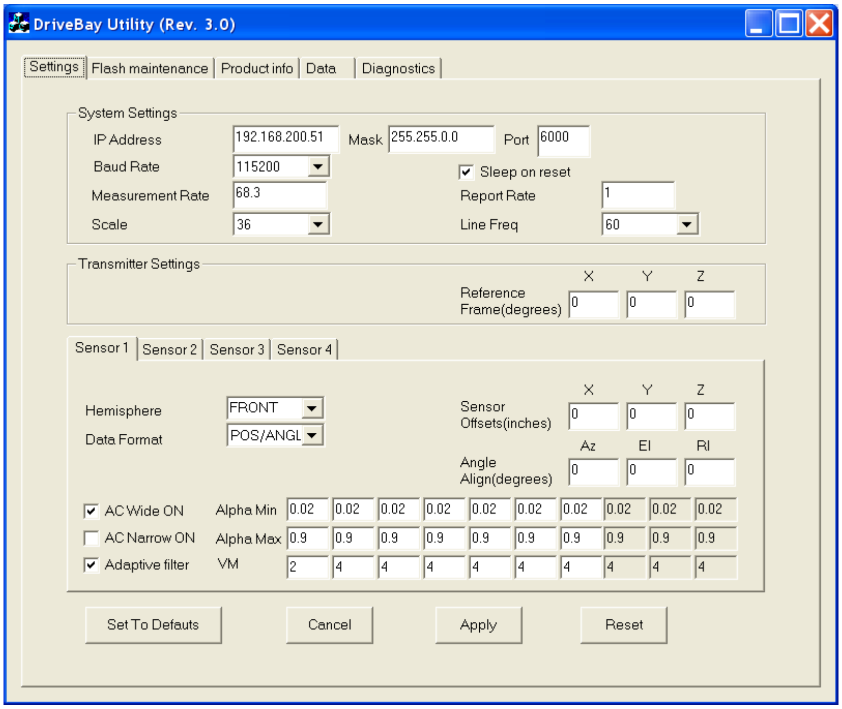
Media: RS232

Port: COM1

Baud Rate: 115200

Status

6. When the reading has completed, select 'Continue'. This will open the Utility to the main display and the 'Settings' tab:



DriveBay Utility (Rev. 3.0)

Settings | Flash maintenance | Product info | Data | Diagnostics

System Settings

IP Address 192.168.200.51 Mask 255.255.0.0 Port 6000

Baud Rate 115200

☑ Sleep on reset

Measurement Rate 68.3

Report Rate 1

Scale 36

Line Freq 60

Transmitter Settings

	X	Y	Z
Reference Frame(degrees)	0	0	0

Sensor 1 | Sensor 2 | Sensor 3 | Sensor 4

Hemisphere FRONT

Data Format POS/ANGL

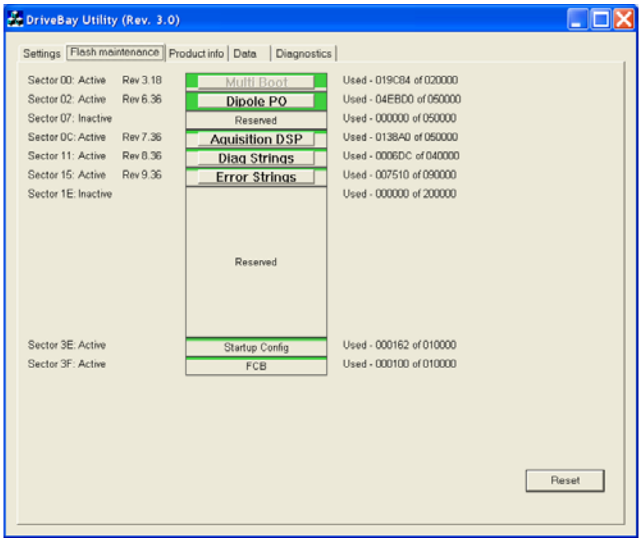
	X	Y	Z
Sensor Offsets(inches)	0	0	0
	Az	El	Rl
Angle Align(degrees)	0	0	0

☑ AC Wide ON	Alpha Min	0.02	0.02	0.02	0.02	0.02	0.02	0.02	0.02	0.02	0.02
☐ AC Narrow ON	Alpha Max	0.9	0.9	0.9	0.9	0.9	0.9	0.9	0.9	0.9	0.9
☑ Adaptive filter	VM	2	4	4	4	4	4	4	4	4	4

Set To Defauts | Cancel | Apply | Reset

7. If you want to change any of the power-up default settings, enter them here, and click 'Apply' to send them to the Flash memory.

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



DriveBay Utility (Rev. 3.0)

Settings | Flash maintenance | Product info | Data | Diagnostics

Sector 00: Active	Rev 3.18	Multi Boot	Used - 019C84 of 020000
Sector 02: Active	Rev 6.36	Dipole PO	Used - 04EBD0 of 050000
Sector 07: Inactive		Reserved	Used - 000000 of 050000
Sector 0C: Active	Rev 7.36	Aquisition DSP	Used - 0138A0 of 050000
Sector 11: Active	Rev 8.36	Diag Strings	Used - 0006DC of 040000
Sector 15: Active	Rev 9.36	Error Strings	Used - 007510 of 090000
Sector 1E: Inactive		Reserved	Used - 000000 of 200000
Sector 3E: Active		Startup Config	Used - 000162 of 010000
Sector 3F: Active		FCB	Used - 000100 of 010000

Reset

9. 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

10. 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 in which a sensor is mounted on a tool or pointing device, the position of its tip must be known. 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:

$$X_S = X_B + X_O * M(1,1) + Y_O * M(2,1) + Z_O * M(3,1)$$

$$Y_S = Y_B + X_O * M(1,2) + Y_O * M(2,2) + Z_O * M(3,2)$$

$$Z_S = Z_B + X_O * M(1,3) + Y_O * M(2,3) + Z_O * M(3,3)$$

Where: X_B, Y_B, Z_B are the X, Y, Z position outputs from the 3DGuidance^{TM} 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^{TM} 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.