InterSense Developer Guide

© 2016 Thales Visionix, Inc.

700 Technology Park Drive, Suite 102

Billerica, MA 01821 USA

Phone +1 781 541 6330 • Fax +1 781 541 6329

www.intersense.com

InterSense Developer Guide

Functions for IS-900 Firmware 4.41 and higher with DLL Version 4.237 and higher.

Functions for IS-1200+ Firmware 1.0 and higher with DLL Version 4.26 and higher.

Functions for IS-1500 with DLL Version 4.2704 and higher.

Functions for InertiaCube supported by all InertiaCubes with DLL Version 4.237 and higher.

Developer Instructions for 4.2401 SDK.

Contacting Thales Visionix

Please contact us if you need assistance.

Thales Visionix, Inc.
700 Technology Park Drive, Suite 102
Billerica, Massachusetts 01821 USA

Telephone: +1 781 541 6330

Fax: +1 781 541 6329

Internet: http://www.intersense.com

Technical Support:

Telephone: +1 781 541 7624
email: techsupport@thalesvisionix.com

Sales:

Telephone: +1 781 541 7650
email: sales@thalesvisionix.com

Patents

The list below identifies the protection granted by the Government of the United States and by other governments to Thales Visionix for InterSense products.

U.S. Patents

5645077, 5807284, 6162191, 6361507, 6786877, 6176837, 6409687, 7395181, 6314055 B1, 6757068 B2, 7301648, 6474159 B1, 6681629 B2, 7000469 B2, 7231063 B2, 6922632 B2, 7725253, 7900524, 8224024, 8473241, 8696458, 8762091, 8972182 and Patents Pending.

Foreign Patents

Europe 1280457 1071369 Israel: 152359 Taiwan: 162248 1071369
China: 99807510.8 Hong Kong: 1039884 Japan: 4690546.

Trademarks

InterSense® is a registered trademark of Thales Visionix, Inc. InertiaCam™, InertiaCube™, IS-900™, VETracker™, and SimTracker™ are trademarks of Thales Visionix, Inc. All other trademarks are the property of their respective owners.

Copyright © 2016 Thales Visionix, Inc.

Table of Contents

1. Introduction

This document describes the API for the InterSense Library (**isense.dll** / **libisense.so** / **libisense.dylib** files included on the product CD). Advantages of the InterSense Library API for customer applications include:

- Provides the simplest programming interface to InterSense processors.
- Unifies the programming interface of the entire InterSense product line.
- Allows you to initialize and retrieve data from up to 32 trackers, each with up to 8 stations.
- Protects your application code from changes in the future.

Details of the communication protocol are also provided as a reference for older applications that may use it, or for use on platforms the library does not support. Serial port and Ethernet (both TCP and UDP) communication is all included in this document.

This API is completely compatible for all InterSense sensors and processors, including InertiaCube, IS-900, IS-1200+ and IS-1500.

2. InterSense Library API

For InterSense SDK Version 4.23 or higher.

This section describes the interface that the application software uses to initialize and retrieve data from InterSense devices using the InterSense library (**isense.dll** / **libisense.so** / **libisense.dylib**). This library and API is provided to simplify communications with all models of InterSense tracking devices. It can detect, configure, and get data from up to 32 trackers, which may have multiple (up to 8) stations in some cases, such as the IS-900 processor. The library maintains compatibility with existing devices, and also makes the applications forward compatible with all future InterSense products. The library is intended to be backwards compatible, so software written for older versions of the DLL should generally run without recompilation using the current version.

NOTE:

The software and accompanying written materials (including instructions for use) is provided "as is" without warranty of any kind; further, licensor does not warrant, or guarantee software, and makes no representations that the use, or the results of use, of the software or written materials are correct, accurate, reliable, current, or otherwise the entire risk as to the results and performance of the software is assumed by licensee.

For complete text of the End User License Agreement, refer to Thales Visionix, Inc. End User License Agreement.

2.1. Sample Program Overview

The library is distributed with sample programs written in C (all platforms) and C# (Windows only) to demonstrate usage. It includes a header file (**isense.h**) with data structure definitions and function prototypes. Most of the API descriptions below are also available in the header file. The header file is heavily commented and contains detailed information about the structures and function calls.

main.c	Main loop of the program. All API calls are made from here.
isense.h	Header file containing function prototypes and definitions, some of which are only applicable to InterSense Professional Series devices and are not used with InterTrax. This file should not be modified.
isense.c	DLL import procedures. This file is included instead of an import library to provide compatibility with all compilers.
isense.dll **libisense.so** **libisense.dylib**	The InterSense DLL and shared libraries. Place these files in the Windows system directory, system library directory, or in the working directory of the application (additional configuration may be required on UNIX platforms).
dlcompat.c **dlcompat.h**	Shared object import procedures for Mac OS X, not used on other operating systems.

2.2. Basics of Using InterSense Library

The API provides an extensive set of functions that can read and set tracker configuration, but in its simplest form can be limited to just three or four function calls, as shown in the simple example below:

```
#include <stdio.h>
#include "isense.h"
#ifdef UNIX
#include <unistd.h>
#endif

void main()
{
    ISD_TRACKER_HANDLE        handle;
    ISD_TRACKER_INFO_TYPE     tracker;
```

```
ISD_TRACKING_DATA_TYPE      data
    int                 i;

    handle = ISD_OpenTracker((Hwnd)NULL, 0, FALSE, FALSE );

    if ( handle > 0 )
        printf( "\n    Az      El      Rl      X       Y       Z \n" );
    else
        printf( "Tracker not found. Press any key to exit" );
```

```
    for (i=0; i < 20; i++) {
        if ( handle > 0 ) {
            ISD_GetTrackingData( handle, &data );

            printf( "%7.2f %7.2f %7.2f %7.3f %7.3f %7.3f  ",
                    data.Station[0].Euler[0],
                    data.Station[0].Euler[1],
                    data.Station[0].Euler[2],
                    data.Station[0].Position[0],
                    data.Station[0].Position[1],
                    data.Station[0].Position[2] );

            ISD_GetCommInfo( handle, &tracker );

            printf( "%5.2f Kb/s %d Rec/s \r",
                    tracker.KBitsPerSec, tracker.RecordsPerSec );
            fflush(0);
        }

#ifdef _WIN32
        Sleep( 1000 );
#elif defined UNIX
        usleep(1e6);
#endif
    }

    ISD_CloseTracker( handle );
}
```

2.2.1. Mapping Ports Using isports.ini

Device names for serial ports vary on different versions of UNIX. The library uses default names for each of the supported operating systems. That, however, is not always sufficient, particularly when you use a USB to Serial converter. To resolve this situation, you can use a configuration file called **isports.ini** to specify port number to device string mapping.

The sample file supplied (in the **Configuration Samples** folder) contains lines for an InterSense USB converter named **/dev/ttyUSB0** (for the Linux port) and a standard serial port (**/dev/ttyS0**, which corresponds to COM1 in Windows). Check your operating system and hardware documentation for specific device names. Enter the information about the port in the **isports.ini** file as follows:

Port*X* = *device***:***baudRate*

Replace the elements in italics with the corresponding information:

X	Logical port number, starting at 1.
device	Path to the device driver, like **/dev/ttyS0**.
baudRate	Optional. All baud rates are tested to detect the tracker. By default, once the tracker is detected, the baud rate is changed to 115200 for best performance. If you are using a very long serial cable, you may want to use this feature to force the lower baud rate. This feature is only available for Windows version in this release.

NOTE: If an **isports.ini** file exists, all required ports must be defined in it; the library searches only those ports.

See also Mapping Ports for IS-1200.

2.2.2. Mapping Bluetooth Ports in isports.ini

To use Bluetooth devices, such as the InertiaCube BT, the library must increase latency/timeout values internally. Since using the timeout values would make detection time unacceptably long, especially on PCs with many COM ports, Bluetooth devices must be detected using a special tag in the **isports.ini** file, **":bluetooth"**.

For example, to connect to an InertiaCube BT detected as COM14:

```
Port1 = COM14:bluetooth
```

Linux Specific Issues

If you are using some recent Linux distributions, you may encounter two potential issues that sometimes prevent proper operation.

> *NOTE: If you encounter any of the problems described in this section, execute a dump from **hcidump** by entering the following Linux command line:*
>
> ```
> hcidump -w bluetooth-debug.pcap
> ```
>
> *Then send the dump file along with the **isense.log** to Technical Support for assistance.*

Linux Issue 1: Modem Manager May Attempt to Send AT Commands to Your RFCOMM Ports

To resolve this issue, you can take two possible actions:

- Remove the package (if no modems are used) with a command like the following on Ubuntu:

  ```
  sudo apt-get remove modemmanager
  ```

  OR

- Create a **udev** rule that sets the vendor/product ID, then sets the **`ID_MM_DEVICE_IGNORE`** environment variable to get the Modem Manager to ignore the device; for example, for the Cirago BTA-6130 class 1 Bluetooth adapter, use this **udev** rule:

```
ATTRS{idVendor}=="0a12", ATTRS{idProduct}=="0001", ENV{ID_MM_DEVICE_IGNORE}="1"
```

  You can obtain the IDs using **lsusb** or similar tools.

Linux Issue 2: D-Bus Use of Bluez (Bluetooth stack on Linux) May Require You to Enter Pairing Code

Since D-bus is now used by Bluez (the standard Bluetooth stack on Linux), you may have to enter a pairing code (1234) that resolves the conflict. You can use any of several GUI utilities to help set the pairing code, including:

- gnome-bluetooth [GNOME]
- bluedevil [KDE]
- blueman [GTK2])

Alternatively, you can use the **bluetooth-agent** command line utility, part of Bluez, if it is installed. First use **rfcomm** to bind the device to a port, then on the KDE desktop use **bluedevil-wizard** to create the pair (but note that you may need to pair twice). First associate the sensor using **rfcomm:**

```
user@host$ hcitool scan
Scanning ...
      00:06:66:4A:5B:3E       BTC-1234567-A
      00:06:66:07:67:F0       BTC-2345678-A
user@host$ sudo rfcomm -i hci0 bind rfcomm0 00:06:66:07:67:F0
user@host$ sudo rfcomm -i hci0 bind rfcomm1 00:06:66:4A:5B:3E
```

Then use the **bluetooth-agent**:

```
user@host$ sudo bluetooth-agent 1234 00:06:66:07:67:F0
Pincode request for device /org/bluez/929/hci0/dev_00_06_66_07_67_F0
Agent has been released
user@host$ sudo bluetooth-agent 1234 00:06:66:4A:5B:3E
Pincode request for device /org/bluez/929/hci0/dev_00_06_66_4A_5B_3E
Agent has been released
```

Finally, add the devices to **isports.ini**:

```
Port1=/dev/rfcomm0:Bluetooth
```

The processor now detects the devices and they operate as expected.

2.2.3. Ethernet Support

Windows and Unix versions support communication to InterSense tracking devices over Ethernet. To use this feature you must define the connection string in the **isports.ini** file in following format:

Port*X* = *IP-address*:*Port*

For example, to configure port 5005 via TCP for 192.168.1.44, enter the line below in the **isports.ini** file:

```
Port1 = 192.168.1.44:5005
```

You can specify UDP ports using only the port number, such as:

```
Port1 = 5001
```

to configure UDP port 5001.

You must also open the port on any client side firewalls.

2.3. Basics of Using InterSense Library with IS-1200+/IS-1500

The IS-1200+/IS-1500 have additional functionality when compared to previous generation trackers. An exposure value can be set by calling ISD_SetExposure().

Other than this function, all functions in this guide are the same for the IS-1200+/IS-1500 as for the IS-900 and InertiaCube.

Three parameters you can set for each particular Optical Inertial Sensor behave differently for the IS-1200+/IS-1500, listed below:

- **Prediction** and **Tip Offset** - these settings may be modified within sfAccess.ini or from within your code.
- **Boresight** – A boresight can be applied either through the DLL API or through sfAccess.ini. Any boresight applied in sfAccess.ini will be separate and in addition to a DLL boresight. Do not use both at the same time.

The InterSense library applies prediction/boresight algorithms after receiving tracking data from the IS-1200+/IS-1500 sensor modules. This differs in behavior from the IS-900 and should be considered when setting up an environment.

Most code written for the IS-900 should not require modifications for the IS-1200+/IS-1500.

2.3.1. Mapping Ports for IS-1200+/IS-1500

Up to 32 different IS-1200+/IS-1500 tracker interfaces can be opened using the InterSense library. For each tracker, an sfAccess.ini file must be created and listed in **isports.ini** in the following format:

Port*X* = **sfAccess:***ini_path*

Replace the elements in italics with the corresponding information:

X	Logical port number, an integer from 1 to 32.
ini_path	Name of the port's **sfAccess.ini** file, such as **sfAccess1.ini**. The **sfAccess.ini** file must reside in the same directory as the **isports.ini** file.

The entries in the **isports.ini** file are NOT case sensitive.

A recommended naming convention modifies the root file name **sfAccess** by adding to it the number of the port that the file belongs to:

```
Port1 = sfAccess:sfAccess1.ini
Port2 = sfAccess:sfAccess2.ini
Port3 = sfAccess:sfAccess3.ini
.
.
.
PortN = sfAccess:sfAccessN.ini
```

In your code, assign one tracker handle to each tracker.

When you subsequently call **`ISD_OpenAllTrackers()`** , **`ISD_OpenTracker()`**, or `ISD_CloseTracker()`, those commands find the tracker and work normally.

2.4. API Functions

Not all functions are applicable to all tracking system models. Calling a function that does not apply to a particular model should have no effect.

ISD_OpenTracker()

```
ISD_TRACKER_HANDLE
ISD_OpenTracker( HWND hParent,
                 DWORD commPort,
                 Bool infoScreen,
                 Bool verbose )
```

This function opens a single tracker. You can call it multiple times to open multiple trackers, though typically Thales recommends you use **`ISD_OpenAllTrackers()`** to open multiple trackers.

`hParent`	Not used. Pass **`NULL`** for this parameter.
`commPort`	If this parameter is a number other than 0, program will try to locate an InterSense tracker on the specified RS232 port. Otherwise it looks for USB device, then for serial port device on all ports at all baud rates. Most applications should pass 0 for maximum flexibility. If you have more than one InterSense device and would like to have a specific tracker connected to a known port, initialized first, then enter the port number instead of 0.
`infoScreen`	Not used. Pass **`FALSE`**.
`verbose`	Pass TRUE if you would like a more detailed report of the DLL activity. Prints messages to Windows console.

ISD_OpenAllTrackers()

```
DWORD
ISD_OpenAllTrackers( Hwnd hParent,
                     ISD_TRACKER_HANDLE *handle,
                     Bool infoScreen,
                     Bool verbose )
```

This function opens multiple trackers. It outputs an array of handles for all detected trackers. Zero is returned on failure.

hParent Not used. Pass NULL for this parameter.

handle An **ISD_TRACKER_HANDLE** array of size **ISD_MAX_TRACKERS**. This is the recommended method for opening multiple trackers. The handle pointer will be populated with handles for all detected trackers when this function returns.

infoScreen Not used. Pass **FALSE**.

verbose Pass **TRUE** if you would like a more detailed report of the DLL activity. Messages are printed to Windows console.

ISD_CloseTracker()

```
Bool
ISD_CloseTracker(ISD_TRACKER_HANDLE handle )
```

This function call de-initializes the tracker, closes the communications port, and frees the resources associated with this tracker. If you pass 0 for the **handle**, all currently open trackers are closed. When the last tracker is closed, the program frees the DLL. Returns FALSE if fails for any reason.

handle Handle to the tracking device. This handle is output by **ISD_OpenTracker()** or **ISD_OpenAllTrackers()**.

ISD_GetTrackerConfig()

```
Bool
ISD_GetTrackerConfig (ISD_TRACKER_HANDLE handle,
                      ISD_TRACKER_INFO_TYPE *Tracker,
                      Bool verbose )
```

Get general tracker information, such as type, model, port, etc. Also retrieves Genlock synchronization configuration, if available. See the **ISD_TRACKER_INFO_TYPE** structure definition for a complete list of items.

handle Handle to the tracking device. This handle is output by **ISD_OpenTracker()** or **ISD_OpenAllTrackers()**.

Tracker Pointer to a structure of type **ISD_TRACKER_INFO_TYPE**.

ISD_SetTrackerConfig()

```
Bool
ISD_SetTrackerConfig( ISD_TRACKER_HANDLE handle,
                      ISD_TRACKER_INFO_TYPE *Tracker,
                      Bool verbose )
```

When used with IS-900 tracking systems this function call sets ultrasonic and synchronization parameters. All other fields in the **ISD_TRACKER_INFO_TYPE** structure are for information purposes only. Has as no effect on the IS-1200+ or IS-1500.

handle — Handle to the tracking device. This handle is output by **ISD_OpenTracker()** or **ISD_OpenAllTrackers()**.

Tracker — Pointer to a structure of type **ISD_TRACKER_INFO_TYPE**.

ISD_GetCommInfo()

```
Bool
ISD_GetCommInfo( ISD_TRACKER_HANDLE handle,
                 ISD_TRACKER_INFO_TYPE *Tracker)
```

Gets **RecordsPerSec** and **KBitsPerSec** without requesting Genlock and other settings from the tracker. Use this function instead of **ISD_SetTrackerConfig()** to prevent your program from stalling while waiting for the tracker response. This call is used to obtain data rate information.

handle — Handle to the tracking device. This handle is output by **ISD_OpenTracker()** or **ISD_OpenAllTrackers()**.

Tracker — Pointer to a structure of type **ISD_TRACKER_INFO_TYPE**.

ISD_SetStationConfig()

```
Bool
ISD_SetStationConfig( ISD_TRACKER_HANDLE handle,
                      ISD_STATION_INFO_TYPE *Station,
                      WORD stationID,
                      Bool verbose )
```

Configures the station as specified in the **ISD_STATION_INFO_TYPE** structure. Before you call this function, you must assign all elements of the structure valid values. General procedure for changing any setting is to first retrieve the current configuration, make the changes, and then apply them. Calling **ISD_GetStationConfig()** is important because you typically only want to change some of the settings, leaving the rest unchanged.

handle — Handle to the tracking device. This handle is output by **ISD_OpenTracker()** or **ISD_OpenAllTrackers()**.

Station — Pointer to a structure of type **ISD_STATION_INFO_TYPE**.

stationID — Number from 1 to **ISD_MAX_STATIONS**.

ISD_GetStationConfig()

```
Bool
ISD_GetStationConfig( ISD_TRACKER_HANDLE handle,
                      ISD_STATION_INFO_TYPE *Station,
                      WORD stationID,
                      Bool verbose )
```

Fills the `ISD_STATION_INFO_TYPE` structure with current settings.

handle Handle to the tracking device. This handle is output by **ISD_OpenTracker()** or **ISD_OpenAllTrackers()**.

Station Pointer to a structure of type **ISD_STATION_INFO_TYPE**.

stationID Number from 1 to **ISD_MAX_STATIONS**.

ISD_ConfigureFromFile()

```
Bool
ISD_ConfigureFromFile(ISD_TRACKER_HANDLE handle,
                      char *path,
                      Bool verbose )
```

When your code first opens a tracker, by default the library automatically looks for a configuration file in current directory of the application. The file name it looks for has the name **isense***X***.ini** where *X* is a number, starting at 1, that identifies the first tracking system in the order that they were initialized. This function provides a way to manually configure the tracker using an arbitrary configuration file instead of the default file.

handle Handle to the tracking device. This handle is output by **ISD_OpenTracker()** or **ISD_OpenAllTrackers()**.

path Pointer to a string representing the complete path to the file to load.

ISD_ConfigSave()

```
Bool
ISD_ConfigSave( ISD_TRACKER_HANDLE handle )
```

Saves tracker configuration. For devices with on-host processing, like InertiaCubes, this function will write to a local file on your PC. The IS-900 saves your configuration in the base unit, and this call will just send a command to commit the changes to permanent storage.

handle Handle to the tracking device. This handle is output by **ISD_OpenTracker()** or **ISD_OpenAllTrackers()**.

ISD_GetTrackingData()

```
Bool
ISD_GetTrackingData( ISD_TRACKER_HANDLE handle,
```

```
ISD_TRACKING_DATA_TYPE *Data )
```

Get data from all configured stations and places the data in the

ISD_TRACKING_DATA_TYPE structure. `TimeStamp` is only available if requested by setting `TimeStamped` field to `TRUE`. Returns `FALSE` if fails for any reason.

handle	Handle to the tracking device. This handle is output by **ISD_OpenTracker()** or **ISD_OpenAllTrackers()**.
Data	Pointer to a structure of type

ISD_TRACKING_DATA_TYPE. Orientation data order is Yaw, Pitch, and Roll for Euler angles and W, X, Y, Z for quaternions.

ISD_GetTrackingDataAtTime()

```
Bool
ISD_GetTrackingDataAtTime (ISD_TRACKER_HANDLE handle,
```

```
ISD_TRACKING_DATA_TYPE *Data,
                            double atTime,
                            double maxSyncWait )
```

Get data from all configured stations and places the data in the

ISD_TRACKING_DATA_TYPE structure. `TimeStamp` is only available if requested by setting `TimeStamped` field to `TRUE`. Returns `FALSE` if fails for any reason.

handle	Handle to the tracking device. This handle is output by **ISD_OpenTracker()** or **ISD_OpenAllTrackers()**.
Data	Pointer to a structure of type

ISD_TRACKING_DATA_TYPE. Orientation data order is Yaw, Pitch, and Roll for Euler angles and W, X, Y, Z for quaternions.

atTime Prediction time to apply (ms)

maxSyncWait Maximum time to wait for the next data record

ISD_GetCameraData()

```
Bool
ISD_GetCameraData( ISD_TRACKER_HANDLE handle,
                   ISD_CAMERA_DATA_TYPE *data )
```

handle Handle to the tracking device. This handle is output by **ISD_OpenTracker()** or **ISD_OpenAllTrackers()**.

Data Camera encoder/other data retrieved from all configured stations

ISD_RingBufferSetup()

```
Bool
ISD_RingBufferSetup( ISD_TRACKER_HANDLE handle,
                     WORD stationID,
                     ISD_STATION_DATA_TYPE *dataBuffer,
                     DWORD samples )
```

By default, **ISD_GetTrackingData()** processes all records available from the tracker and only outputs the latest data. As the result, data samples can be lost if it is not called frequently enough. If all the data samples are required, you can use a ring buffer to store them. **ISD_RingBufferSetup()** accepts a pointer to the ring buffer, and its size. Once activated, all processed data samples are stored in the buffer for use by the application.

ISD_GetTrackingData() can still be used to read the data, but will output the oldest saved data sample, then remove it from the buffer (FIFO). By repeatedly calling **ISD_GetTrackingData()**, all samples are retrieved, the latest coming last. All consecutive calls to **ISD_GetTrackingData()** will output the last sample, but the `NewData` flag will be `FALSE` to indicate that the buffer has been emptied.

handle Handle to the tracking device. This handle is output by **ISD_OpenTracker()** or **ISD_OpenAllTrackers()**.

stationID Number from 1 to **ISD_MAX_STATIONS**.

dataBuffer An array of **ISD_STATION_DATA_TYPE** structures. Pass in `NULL` if you do not need visibility into the complete buffer (typical).

samples The size of the ring buffer. **ISD_GetTrackingData()** should be called frequently enough to avoid buffer overrun.

ISD_RingBufferStart()

```
Bool
ISD_RingBufferStart( ISD_TRACKER_HANDLE handle,
                     WORD stationID )
```

Activates the ring buffer. While the buffer is active, all data samples are stored in the buffer. Because this is a ring buffer, it will store only the number of samples specified in the call to **ISD_RingBufferSetup()**, so the oldest samples can be overwritten.

handle	Handle to the tracking device. This handle is output by **ISD_OpenTracker()** or **ISD_OpenAllTrackers()**.
stationID	Number from 1 to **ISD_MAX_STATIONS**.

ISD_RingBufferStop()

```
Bool
ISD_RingBufferStop( ISD_TRACKER_HANDLE handle,
                    WORD stationID )
```

Stops collection. The library will continue to process data, but the contents of the ring buffer will not be altered.

handle	Handle to the tracking device. This handle is output by **ISD_OpenTracker()** or **ISD_OpenAllTrackers()**.
stationID	Number from 1 to **ISD_MAX_STATIONS**.

ISD_RingBufferQuery()

```
Bool
ISD_RingBufferQuery( ISD_TRACKER_HANDLE handle,
                     WORD stationID,
                     ISD_STATION_DATA_TYPE *currentData,
                     DWORD *head,
                     DWORD *tail )
```

Queries the library for the latest data without removing it from the buffer or affecting the NewData flag. It also outputs the indexes of the newest and the oldest samples in the buffer. These can then be used to parse the buffer.

handle	Handle to the tracking device. This handle is output by **ISD_OpenTracker()** or **ISD_OpenAllTrackers()**.
stationID	Number from 1 to **ISD_MAX_STATIONS**.
currentData	An array of **ISD_STATION_DATA_TYPE** used as the buffer.
head	Pointer to the current head of the ring buffer.
tail	Pointer to the current tail of the ring buffer.

ISD_ResetHeading()

```
Bool
ISD_ResetHeading( ISD_TRACKER_HANDLE handle,
                  WORD stationID )
```

Reset heading (yaw) to zero. Exclusively for the InertiaCube. Has as no effect on other trackers on the IS-900 or IS-1200+ or IS-1500.

handle Handle to the tracking device. This handle is output by **ISD_OpenTracker()** or **ISD_OpenAllTrackers()**.

stationID Number from 1 to **ISD_MAX_STATIONS**.

ISD_BoresightReferenced()

```
Bool
ISD_BoresightReferenced( ISD_TRACKER_HANDLE handle,
                         WORD stationID,
                         float yaw,
                         float pitch,
                         float roll )
```

Boresights (aligns) station using particular reference angles as offsets. This function is useful when you need to apply a particular offset to system output. For example, if a sensor is mounted at 40° relative to the Head Mounted Display (HMD), you can enter 0, 40, 0 to get the system to output (0, 0, 0) for yaw, pitch, and roll, when the HMD is horizontal.

handle Handle to the tracking device. This handle is output by **ISD_OpenTracker()** or **ISD_OpenAllTrackers()**.

stationID Number from 1 to **ISD_MAX_STATIONS**.

Yaw

Pitch

roll Boresight reference angles.

ISD_Boresight()

```
Bool
ISD_Boresight( ISD_TRACKER_HANDLE handle,
               WORD stationID,
               Bool set )
```

If **set** is **TRUE**, boresights (aligns) the station using the current orientation (yaw, pitch, and roll) as reference angles. Or If **set** is **FALSE**, unboresights the stations (removes any previously applied boresight angles), including those **ISD_ResetHeading()** and **ISD_BoresightReferenced()** have set.

The illustration below gives an example of the effect of calling **ISD_Boresight()**. In this example, the sensor moves from position A to B to C. The starting position (A) is at 0° yaw, pitch, and roll. If you turn the station to 90° yaw and 0° pitch/roll (B), then call this function, now the sensor axes are located as shown in C, at 0° yaw, pitch, and roll with roll being rotation around the new x axis and pitch being rotation around the new y axis.

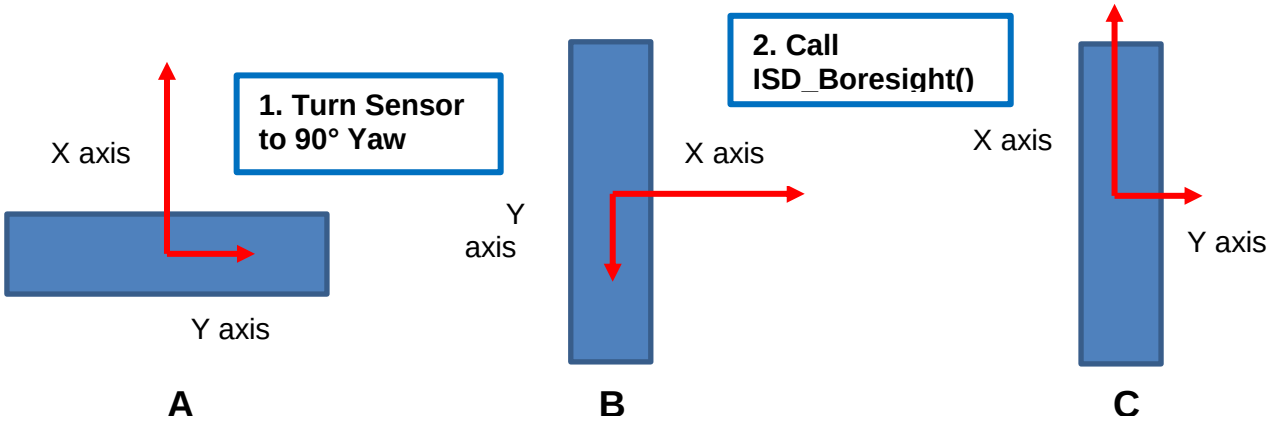


Figure 1 Effect of First Turning Tracker 90°, then Boresighting Station

handle	Handle to the tracking device. This handle is output by **ISD_OpenTracker()** or **ISD_OpenAllTrackers()**.
stationID	Number from 1 to **ISD_MAX_STATIONS**.
set	**TRUE** to set boresight or **FALSE** to clear boresight.

ISD_SendScript()

```
Bool
ISD_SendScript( ISD_TRACKER_HANDLE handle,
                char *script )
```

Send a configuration script to the tracker. Script must consist of valid commands as described in the interface protocol. You should terminate all commands in the script with the newline character **'\n'**. The function adds the linefeed character **'\r'** and is thus not required after each command.

Note that this function may not be supported when using the shared memory interface, such as with sfServer, and is primarily intended for the IS-300/IS-600/IS-900 system. Has no effect on the IS-1200+ or IS-1500.

handle	Handle to the tracking device. This handle is output by **ISD_OpenTracker()** or **ISD_OpenAllTrackers()**.
script	Pointer to a string containing the command script.

ISD_AuxOutput()

```
Bool
ISD_AuxOutput( ISD_TRACKER_HANDLE handle,
               WORD stationID,
               BYTE *AuxOutput,
               WORD length )
```

Sends up to 4 output bytes to the auxiliary interface of the station specified. The number of bytes should match the number the auxiliary outputs that the interface is configured to expect. If too many are specified, extra bytes are ignored. Has no effect on the IS-1200+.

handle	Handle to the tracking device. This handle is output by **ISD_OpenTracker()** or **ISD_OpenAllTrackers()**.
stationID	Number from 1 to **ISD_MAX_STATIONS**.
AuxOutput	An array of **BYTES** to send.
length	Size of **AuxOutput**.

ISD_NumOpenTrackers()

```
Bool
ISD_NumOpenTrackers( WORD *num )
```

Retreives the number of currently opened trackers and stores that number in the parameter passed to it.

***num**	Pointer to the number of open trackers.

ISD_GetTime()

```
float
ISD_GetTime( void )
```

Platform independent time function.

ISD_UdpDataBroadcast()

```
Bool
ISD_UdpDataBroadcast( ISD_TRACKER_HANDLE handle,
                      DWORD port,
```

```
ISD_TRACKING_DATA_TYPE *trackingData,
                       ISD_CAMERA_DATA_TYPE *cameraData )
```

Broadcast tracker data over the network using UDP broadcast.

handle	Handle to the tracking device. This handle is output by **ISD_OpenTracker()** or **ISD_OpenAllTrackers()**.
port	UDP port (0 to 65535) .
trackingData	A

ISD_TRACKING_DATA_TYPE structure containing the data to send, retrieved with **ISD_GetTrackingData()**.

cameraData Pass NULL to this.

ISD_GetSystemHardwareInfo()

```
Bool
ISD_GetSystemHardwareInfo( ISD_TRACKER_HANDLE handle,

An application uses this data structure to output camera data for all
configured stations.  A pointer to a structure of this type is passed to
ISD_GetCameraConfig().
typedef struct
{
   ISD_CAMERA_ENCODER_DATA_TYPE Camera[ISD_MAX_STATIONS];
}
ISD_CAMERA_DATA_TYPE;
```

```
ISD_HARDWARE_INFO_TYPE *hwInfo )
```

Retrieve system hardware information. Note that the system is a single tracker (and will thus have one handle). For details on individual stations (such as the devices on each port of an IS-900), use `ISD_GetStationHardwareInfo()` instead. Has no effect on the IS-1200+ or IS-1500.

handle Handle to the tracking device. This handle is output by **ISD_OpenTracker()** or **ISD_OpenAllTrackers()**.

hwInfo An

An application uses this data structure to output camera data for all configured stations. A pointer to a structure of this type is passed to ISD_GetCameraConfig().

typedef struct

{

ISD_CAMERA_ENCODER_DATA_TYPE Camera[ISD_MAX_STATIONS];

}

ISD_CAMERA_DATA_TYPE;

ISD_HARDWARE_INFO_TYPE structure containing the information. The structure definition is given below.

ISD_GetStationHardwareInfo()

```
Bool
ISD_GetStationHardwareInfo( ISD_TRACKER_HANDLE handle,
                            ISD_STATION_HARDWARE_INFO_TYPE *info,
                            WORD stationID )
```

Retrieve station hardware information. Stations are individual devices (such as a wand or head tracker) connected to a tracker (such as an IS-900). Has no effect on the IS-1200+ or IS-1500.

handle Handle to the tracking device. This handle is output by **ISD_OpenTracker()** or **ISD_OpenAllTrackers()**.

info An **ISD_STATION_HARDWARE_INFO_TYPE** structure containing the information. The structure definition is given below.

stationID Number from 1 to **ISD_MAX_STATIONS**.

ISD_EnterHeading()

```
Bool
ISD_EnterHeading ( ISD_TRACKER_HANDLE handle,
                   ISD_STATION_HARDWARE_INFO_TYPE *info,
                   WORD stationID,
                   float yaw )
```

Provide external yaw data (degrees) to the DLL for use in the Kalman filter. Once provided, this data will be used instead of data from the compass, until the sensor is re-initialized. It should be called at a regular rate to keep providing updated heading information. This function is typically used for special scenarios and is not needed for regular tracking. Has no effect on the IS-1200+ or IS-1500.

handle Handle to the tracking device. This handle is output by **ISD_OpenTracker()** or **ISD_OpenAllTrackers()**.

stationID Number from 1 to **ISD_MAX_STATIONS**.

yaw The current yaw value to use instead of the compass-provided yaw.

ISD_GetPortWirelessInfo()

```
Bool
ISD_GetPortWirelessInfo( ISD_TRACKER_HANDLE handle,
                         WORD port,
                         ISD_PORT_WIRELESS_INFO_TYPE *info )
```

Retrieve wireless configuration information. Has no effect on the IS-1200+.

handle Handle to the tracking device. This handle is output by **ISD_OpenTracker()** or **ISD_OpenAllTrackers()**.

stationID Number from 1 to **ISD_MAX_STATIONS**.

port Station or port to get info from, starting at 0 for the first port.

info An **ISD_PORT_WIRELESS_INFO_TYPE** structure containing the information. The structure definition is given below.

ISD_SetIlluminationMode()

```
Bool
ISD_SetIlluminationMode( ISD_TRACKER_HANDLE handle, Bool night )
```

Set illumination mode for the IR-based IS-1200+.

handle Handle to the tracking device. This handle is output by **ISD_OpenTracker()** or **ISD_OpenAllTrackers()**.

night True for night mode. False for day mode.

ISD_SetExposure()

```
Bool
ISD_SetExposure( ISD_TRACKER_HANDLE handle, int exposure )
```

Set camera exposure value. (IS-1200+/IS-1500 only).

handle Handle to the tracking device. This handle is output by **ISD_OpenTracker()** or **ISD_OpenAllTrackers()**.

exposure 32-bit signed integer representing the exposure value in microseconds. Typical values range from 100 to 10,000.

ISD_StationMagCalStart()

```
Bool ISD_StationMagCalStart(ISD_TRACKER_HANDLE handle, float duration,
   WORD stationID, WORD axis)
```

Begin magnetic calibration

handle Handle to the tracking device. This handle is output by **ISD_OpenTracker()** or **ISD_OpenAllTrackers()**.

duration Time in seconds to calibrate.

stationID Number from 1 to **ISD_MAX_STATIONS**.

axis Number of axes to calibrate, 2 or 3.

ISD_StationMagCalStatus ()

```
Bool
ISD_StationMagCalStatus(ISD_TRACKER_HANDLE handle,
   ISD_STATION_MAGCAL_TYPE *data, WORD stationID)
```

```
Fetch data related to the current magnetic calibration and progress
```

handle Handle to the tracking device. This handle is output by **ISD_OpenTracker()** or **ISD_OpenAllTrackers()**.

Data Current status of magnetic calibration of type ISD_STATION_MAGCAL_TYPE.

stationID Number from 1 to ISD_MAX_STATIONS.

ISD_StationMagCalCancel ()

```
Bool
ISD_StationMagCalCancel(ISD_TRACKER_HANDLE handle, WORD stationID)
```

```
Cancel magnetic calibration, do not apply
```

handle Handle to the tracking device. This handle is output by **ISD_OpenTracker()** or **ISD_OpenAllTrackers()**.

stationID Number from 1 to ISD_MAX_STATIONS.

ISD_StationMagCalApply ()

```
Bool
ISD_StationMagCalApply(ISD_TRACKER_HANDLE handle, WORD stationID)
```

```
Apply new magnetic calibration calibration
```

handle Handle to the tracking device. This handle is output by **ISD_OpenTracker()** or **ISD_OpenAllTrackers()**.

stationID Number from 1 to ISD_MAX_STATIONS.

ISD_StationMagCalClear()

```
Bool
ISD_StationMagCalClear(ISD_TRACKER_HANDLE handle, WORD stationID)
```

```
Restore factory magnetic calibration
```

handle	Handle to the tracking device. This handle is output by **ISD_OpenTracker()** or **ISD_OpenAllTrackers()**.
stationID	Number from 1 to ISD_MAX_STATIONS.

ISD_GetGpsData ()

```
Bool
ISD_GetGpsData(ISD_TRACKER_HANDLE handle, Bool *valid,
ISD_GPS_DATA_TYPE *data)
```

```
Retrieve GPS Data from sfHub.  Only for IS-1500
```

handle	Handle to the tracking device. This handle is output by **ISD_OpenTracker()** or **ISD_OpenAllTrackers()**.
Valid	Valid is true when GetGpsData() is successful.
data	ISD_GPS_DATA_TYPE structure holding output.

ISD_SetGpsData ()

```
Bool
ISD_SetGpsData(ISD_TRACKER_HANDLE handle, ISD_GPS_DATA_TYPE *data)
```

```
Send GPS data to sfHub.  Only for IS-1500
```

handle	Handle to the tracking device. This handle is output by **ISD_OpenTracker()** or **ISD_OpenAllTrackers()**.
data	ISD_GPS_DATA_TYPE structure holding input data.

ISD_GetImageBufSize ()

```
DWORD
ISD_GetImageBufSize(ISD_TRACKER_HANDLE handle)
```

```
Get the camera image buffer size.  Only for IS-1500
```

handle	Handle to the tracking device. This handle is output by **ISD_OpenTracker()** or **ISD_OpenAllTrackers()**.

ISD_GetImage()

```
Bool
ISD_GetImage(ISD_TRACKER_HANDLE handle, int *progress, DWORD *format,
DWORD *sid, BYTE buf[], int bufsize)
```

```
Retrieve the last received image.  Only for IS-1500
```

handle	Handle to the tracking device. This handle is output by **ISD_OpenTracker()** or **ISD_OpenAllTrackers()**.
progress	0-100 percent progress. 0 = waiting, 100 = done. 1-99 = receiving.
format	Image format as ISD_IMAGE_FORMAT_TYPE.
sid	Image sequence ID.
buf	in/out. Image buffer.
bufsize	Size of the image buffer.

ISD_GetImageInfo()

```
Bool
ISD_GetImageInfo(ISD_TRACKER_HANDLE handle, DWORD format, int *size, int
*width, int *Height, int *bpp)
```

```
Reports image info given format (outputs ignored if null).  Only for IS-1500
```

handle	Handle to the tracking device. This handle is output by **ISD_OpenTracker()** or **ISD_OpenAllTrackers()**.
format	Image format as ISD_IMAGE_FORMAT_TYPE.
size	Image size in bytes.
width	Image width in pixels.
height	Image height in pixels.
bpp	Bytes per pixel.

ISD_SetImagesEnabled()

```
Bool
ISD_SetImagesEnabled (ISD_TRACKER_HANDLE handle, Bool enabled)
```

```
Enables/disables image transfer.  Only for IS-1500
```

handle	Handle to the tracking device. This handle is output by **ISD_OpenTracker()** or **ISD_OpenAllTrackers()**.
enabled	True = enabled, false = disabled.

2.5. API Data Structures

ISD_TRACKER_HANDLE

Unique identifier for a tracker returned or output when you successfully call either the **ISD_OpenTracker()** or **ISD_OpenAllTrackers()** function. The handle represents a tracker on either an InertiaCube or an IS-900 processor, and you use this handle to manipulate that tracker in your code.

```
typedef int ISD_TRACKER_HANDLE;
```

ISD_TRACKER_INFO_TYPE

This data structure outputs information about the tracker and provides access to system level settings.

```
typedef struct {
    float  LibVersion;
    DWORD  TrackerType;
    DWORD  TrackerModel;
    DWORD  Port;
    DWORD  RecordsPerSec;
    float  KBitsPerSec;
    DWORD  SyncState;
    float  SyncRate;
    DWORD  SyncPhase;
    DWORD  Interface;
    DWORD  UltTimeout;
    DWORD  UltVolume;
    DWORD  dwReserved4;
    float  FirmwareRev;
    float  fReserved2;
    float  fReserved3;
    float  fReserved4;
    Bool   LedEnable;
    Bool   bReserved2;
    Bool   bReserved3;
    Bool   bReserved4;
}
ISD_TRACKER_INFO_TYPE;
```

LibVersion	InterSense library version (version of DLL or shared library).
TrackerType	One of the values defined in **ISD_SYSTEM_TYPE**.
TrackerModel	One of the values defined in **ISD_SYSTEM_MODEL**.
Port	Number of the hardware ports the tracker is connected to. Starts with **1**.
RecordsPerSec	Communications statistics (number of data records/sec from tracker).
KBitsPerSec	Communications statistics (Kb/sec of data from tracker).

SyncState	Applies to IS-X Series devices only. Can be one of four values: **0** – OFF; system is in free run. **1** – Not used. **2** – ON; user specifies the hardware genlock frequency. **3** – ON; no hardware signal, locked to the user-specified frequency.
SyncRate	Sync frequency – Number of hardware sync signals per second, or, if SyncState is **3** – Data record output frequency.
SyncPhase	The time within the sync period at which a data record is transmitted. The phase point is specified as a percentage of the sync period. 0% (the default) instructs the tracker to output a data record as soon as possible after the sync period begins. 100% delays the output of a record as much as possible before the next sync period begins.
Interface	Hardware interface type, as defined in **ISD_INTERFACE_TYPE**.
UltTimeout	IS-900 only, ultrasonic timeout (sampling rate).
UltVolume	IS-900 only, ultrasonic speaker volume.
FirmwareRev	Firmware revision for tracker.
LedEnable	IS-900 only, blue LED on the SoniDiscs enable flag.

ISD_STATION_INFO_TYPE

An application uses this data structure to get and set station configuration values, using `ISD_GetStationConfig()` and `ISD_SetStationConfig()`.

```
typedef struct {
    DWORD   ID;
    Bool    State;
    Bool    Compass;
    LONG    InertiaCube;
    DWORD   Enhancement;
    DWORD   Sensitivity;
    DWORD   Prediction;
    DWORD   AngleFormat;
    Bool    TimeStamped;
    Bool    GetInputs;
    Bool    GetEncoderData;
    BYTE    CompassCompensation;
    BYTE    ImuShockSuppression;
    BYTE    UrmRejectionFactor;
    BYTE    GetAHRSData;
    DWORD   CoordFrame
    DWORD   AccelSensitivity;
    float   fReserved1;
    float   fReserved2;
    float   TipOffset[3];
    float   fReserved3;
    Bool    GetCameraData;
    Bool    GetAuxInputs;
    Bool    GetCovarianceData;
    Bool    GetExtendedData;
}
ISD_STATION_INFO_TYPE;
```

ID — A unique number identifying a station. It is the same as that passed to the `ISD_SetStationConfig()` and `ISD_GetStationConfig()` functions and can be **1** to **ISD_MAX_STATIONS**.

State — **TRUE** if on, **FALSE** if off. InertiaCubes are considered a tracking system consisting of one station, which cannot be turned off, so this field will always be **TRUE** for InertiaCubes. **The** IS-900 may have up to 7 stations connected.

Compass — Only available for InertiaCube devices. For all others this setting is always **2**. This setting controls the state of the compass component of the InertiaCube. Compass is only used when station is configured for GEOS or Dual modes; in Fusion mode compass readings are not used, regardless of this setting. When station is configured for full compass mode (**2**), the readings the magnetometers inside the InertiaCube produce are used as absolute reference orientation for yaw. Compass can be affected by metallic objects and electronic equipment in close proximity to an InertiaCube. Older versions of tracker firmware supported only **0** and **1**, which stood for ON or OFF. You must use the new notation; as the API correctly interprets only the new settings. Setting 1 is for special use cases and should not be used unless InterSense recommends it.

InertiaCube	InertiaCube associated with this station. If no InertiaCube is assigned, this number is **–1**. Otherwise, it is a positive number **1** to **ISD_MAX_STATIONS**. Only relevant for IS-300 and IS-600 Series devices. For IS-900 systems, it is always the same as the station number, for InterTrax and InertiaCubes it is always 1.
Enhancement	In order to provide the best performance for a large range of various applications, three levels of perceptual enhancement are available. None of the modes introduces any additional latency. The InterTrax is restricted to Mode **2**. Mode 0 provides the best accuracy. The inertial tracker uses gyros to measure angular rotation rates for computing the sensor's orientation. To compensate for the gyroscopic drift, depending on the configuration, the tracker may use accelerometers, magnetometers or SoniDiscs to measure the actual physical orientation of the sensor. That data is then used to compute the necessary correction. In Mode **0** correction adjustments are made immediately, no jitter reduction algorithms are used. The result is in somewhat jumpy output (not recommended for head tracking) but with lower RMS error. Use this mode for accuracy testing or for any application that requires best accuracy. Mode 1 provides accuracy similar to that of Mode **0**, with an addition of a jitter reduction algorithm. This algorithm reduces the accuracy by only a small amount and does not add any latency to the measurements. Mode **1** is recommended for augmented reality applications (i.e., overlaying or mixing both virtual and real objects in a visualization system.) Mode 2 is recommended for use with HMD or other immersive applications. The drift correction adjustments are made smoothly and only while the sensor is moving, so as to be transparent to the user.
Sensitivity	Use this setting only when Perceptual **Enhancement** level is set to **1** or **2**. It controls the minimum angular rotation rate that the InertiaCube picks up. Default is level **3**. Increasing sensitivity does not increase latency during normal movements. It may, however, result in some small residual movements for a couple of seconds after the sensor has stopped. If your application requires sensitivity greater than the maximum this control provides, you must use Perceptual **Enhancement** level **0**. For InterTrax this value is fixed to default and you cannot change it.
Prediction	Inertial sensors can predict motion up to 50 ms into the future, which compensates for graphics rendering delays and further contributes to eliminating simulator lag. IS-300, IS-600, IS-900 and InertiaCubes support this argument. Not available for the InterTrax.
AngleFormat	**ISD_EULER** or **ISD_QUATERNION**. The Euler angles are defined as rotations about Z, then Y, then X in body frame. Angles are output in°. Default is **ISD_EULER**.
TimeStamped	**TRUE** if time stamp is requested. Default is **FALSE**.
GetInput	**TRUE** if button and joystick data is requested. Default is **FALSE**.
GetEncoderData	**TRUE** if raw encoder data is requested. Default is **FALSE**.

CompassCompensation

This setting controls how the system applies Magnetic Environment Calibration. This calibration calculates nominal field strength and dip angle for the environment in which the sensor is used. Based on these values, the system can assign weight to compass measurements, allowing it to reject bad measurements. Values from **0** to **3** are accepted. If you set **CompassCompensation** to **0**, the calibration is ignored and all compass data is used. Higher values result in a tighter rejection threshold, resulting in more measurements being rejected. Using the sensor in an environment with significant magnetic interference can result in drift due to insufficient compensation from the compass data. Default setting is **2**.

Note that you must calibrate the sensor in the ISDemo Compass Calibration Tool for this setting to have any effect.

ImuShockSuppression

Setting controls how the system deals with sharp changes in inertia measurement unit (IMU) data that can be caused by shock or impact. Sensors may experience momentary rotation rates or accelerations that are outside of the specified range, resulting in undesirable behavior. By turning on shock suppression you can have the system filter out corrupted data. Values **0** (OFF) to **2** are accepted, with higher values resulting in greater filtering.

UrmRejectionFactor Setting controls the rejection threshold for ultrasonic measurements. Currently, it is implemented only for the IS-900 PCTracker. Default setting is **4**, which rejects measurements with range errors greater than 4 times the average. Please do not change this setting without first consulting with technical support.

GetAHRSData Obtain AHRS data from sensor instead of calculating it from sensor data. Only the IC4 product supports this data (with FW >= 5). Note that enabling this option increases the sensor's current consumption (and thereby reduces battery runtime on battery powered sensors). Thales Visionix recommends this option only for testing AHRS performance in applications that already use the **isense.dll** (in anticipation of creating an embedded version that talks directly to the IC4). After setting, use **ISD_GetStationConfig()** to verify whether or not the application is using it (as it requires FW >= 5).

CoordFrame Coordinate frame within which the sensor reports position and orientation data. Can be **ISD_DEFAULT_FRAME** or **ISD_VSET_FRAME**. The second option applies to camera tracker only. Default is **ISD_DEFAULT_FRAME**.

AccelSensitivity For 3-DOF tracking with InertiaCube products only. It controls how fast the InertiaCube applies tilt correction, using accelerometers. Valid values are **1** to **4**, with **2** as default.

Level 1 reduces the amount of tilt correction during movement. While it prevents any effect linear accelerations may have on pitch and roll, it also reduces stability and dynamic accuracy. You should use it only in situations when the sensor is not expected to experience a lot of movement.

Level 2 (default) is best for head tracking in static environment, with user seated.

Level 3 allows for more aggressive tilt compensation, appropriate when sensor is moved a lot; for example, when the user is walking for long periods of time.

Level 4 allows for even greater tilt corrections. It reduces orientation accuracy by allowing linear accelerations to effect orientation, but increase stability. This level is appropriate for when the user is running or in other situations where the sensor experiences a great deal of movement.

fReserved1 Reserved for future use.

TipOffset Offset of the reported position from the physical point being tracked. This option is only applicable to system capable of tracking position.

fReserved2 Reserved for future use.

fReserved3 Reserved for future use.

GetCameraData **TRUE** to get computed FOV, aperture, etc. Default is **FALSE**.

GetAuxInputs **TRUE** to get values from auxiliary inputs connected to the I2C port in the MicroTrax device.

GetCovarianceData Do not change this parameter.

GetExtendedData When this flag is set, several items are valid in the **Error! Reference ource not found.** structure:

- **AngularVelBodyFrame**
- **AngularVelNavFrame**
- **AccelBodyFrame**
- **AccelNavFrame**
- **VelocityNavFrame**

This extended data is primarily useful for troubleshooting; since the example application (*ismain*) enables these parameters, you can use the program as a tool for logging data to troubleshoot tracking performance issues. You may want to collect this data for other purposes.

NOTE: Over a serial connection to an IS-900 with several stations, outputting extended data may produce too much data to transmit over the serial port. For this reason, Thales recommends using Ethernet to communicate with for IS-900s when processing extended data.

ISD_STATION_DATA_TYPE

An application uses this data structure to output current data for a station, including position, orientation, time stamp, button, and analog channel state. It is passed to **ISD_GetTrackingData()** as part of **ISD_TRACKING_DATA_TYPE**.

```
typedef struct
{
    BYTE    TrackingStatus;
    BYTE    NewData;
    BYTE    CommIntegrity;
    BYTE    BatteryState;

    float   Euler[3];
    float   Quaternion[4];
    float   Position[3];
    float   TimeStamp;
    float   StillTime;
    float   BatteryLevel;
    float   CompassYaw;

    Bool    ButtonState[ISD_MAX_BUTTONS];
    short   AnalogData[ISD_MAX_CHANNELS];
    BYTE    AuxInputs[ISD_MAX_AUX_INPUTS];

    float   AngularVelBodyFrame[3];
    float   AngularVelNavFrame[3];
    float   AccelBodyFrame[3];
    float   AccelNavFrame[3];
    float   VelocityNavFrame[3];
    float   AngularVelRaw[3];

    BYTE    MeasQuality;
    BYTE    bReserved2;
    BYTE    bReserved3;
    BYTE    bReserved4;

    DWORD   TimeStampSeconds;
    DWORD   TimeStampMicroSec;
    DWORD   OSTimeStampSeconds;
    DWORD   OSTimeStampMicroSec;

    BYTE    CompassQuality;

    INT8    TrackingState;

    BYTE    bReserved3;
    BYTE    bReserved4;

    float   Reserved[36];

    float   Cbn[3][3];

    float   RotSig[3];
    float   PosSig[3];
    float   VelSig[3];

    float   Temperature;

    float   MagBodyFrame[3];
```

```
}
ISD_STATION_DATA_TYPE;
```

TrackingStatus	Status that represents Tracking Quality, ranging from **0** to **255**, where **0** means the tracker is lost.
NewData	**TRUE** if data changed since last call to **ISD_GetTrackingData().**
CommIntegrity	Communication integrity, measured as the percentage of packets received from tracker, from **0** to **100**.
BatteryState	Status of the battery, for wireless devices only: **0**=N/A, **1**=Low, **2**=OK
Euler	Orientation in Euler angle format (Yaw, Pitch, Roll).
Quaternion	Orientation in Quaternion format (W, X, Y, Z).
Position	Station position in meters.
TimeStamp	Time stamp in seconds, reported only if requested.
StillTime	Number of seconds the tracker is motionless. InertiaCube and PC-Tracker products only.
BatteryLevel	Battery voltage in volts, if available.
CompassYaw	Magnetometer heading, computed based on current orientation. Available for InertiaCube products only, such as IC2, IC3 and IC2+
ButtonState	Only if requested. Current hardware is limited to 10 channels, with only two in use. Used by IS-900 wands that have a built-in analog joystick. Channel 1 is x axis rotation, channel 2 is y axis rotation
AnalogData	Only if requested
AuxInputs	Number of auxiliary input channels (OEM products).
AngularVelBodyFrame	rad/sec, in sensor body coordinate frame. Reported as rates about x, y and z axes, corresponding to Roll, Pitch, Yaw order. This parameter is the processed angular rate, with current biases removed— the angular rate used to produce orientation updates.
AngularVelNavFrame	rad/sec, in world coordinate frame, with boresight and other transformations applied. Reported as rates about x, y and z axes, corresponding to Roll, Pitch, Yaw order.
AccelBodyFrame	meter/sec^2, in sensor body coordinate frame. These three values are the accelerometer measurements in the sensor body coordinate frame. Only factory calibration is applied to this data; gravity component is not removed. Reported as accelerations along x, y and z axes.
AccelNavFrame	meters/sec^2, in the navigation (earth) coordinate frame. These three values are the accelerometer measurements with calibration, current sensor orientation applied, and subtracted. These values are the best available estimate of tracker acceleration. Reported as accelerations along x, y and z axes.
VelocityNavFrame	meters/sec, 6-DOF systems only. Reported as velocity along X, Y and Z axes.
AngularVelRaw	Raw gyro output, only factory calibration is applied. Some errors can occur due to temperature-dependent gyro bias drift.

MeasQuality	Ultrasonic Measurement Quality (Applies only to IS-900 processors with firmware versions >= 4.26).
HardIronCal	1 if Hard Iron Compass calibration exists and is being applied, 0 otherwise
SoftIronCal	1 if Soft Iron Compass calibration exists and is being applied, 0 otherwise
EnvironmentCal	1 if Environmental Compass calibration exists. See CompassCompensation field more information.
TimeStampSeconds	Time stamp in whole seconds.
TimeStampMicroSec	Fractional part of the time stamp in micro-seconds.
OSTimeStampSeconds	Reserved for future use.
OSTimeStampMicroSec	Reserved for future use.
CompassQuality	If Environmental Calibration exists this value contains the calculated quality of the compass measurement based on deviation from nominal dip angle and magnitude values. Values are 0 - 100.
TrackingState	Denotes the state the tracker is in. 0: No communication. 1: Tracking with relative position. 2: Tracking with point of reference for position. 3: Not tracking, lost.
bReserved3	Reserved for future use.
bReserved4	Reserved for future use.
Reserved	Reserved for future use.
Cbn	3x3 Rotational Matrix
RotSig	roll/pitch/yaw orientation sigma (rad) (IS-1200+ and IS-1500 only)
PosSig	X/Y/Z position sigma (m) (IS-1200+ and IS-1500 only)
VelSig	X/Y/Z velocity sigma (m/s) (IS-1200+ and IS-1500 only)
MagBodyFrame	3- DOF sensors only. Magnetometer data along the x, y, and z axes. Units are nominally in Gauss, with factory calibration applied. Note, however, that most sensors are not calibrated precisely since the exact field strength is not necessary to for tracking purposes. Relative magnitudes should be accurate, however. Fixed metal compass calibration may rescale the values as well.

ISD_CAMERA_ENCODER_DATA_TYPE

An application uses this data structure to retrieve information about a camera/encoder. This structure is passed as a member of the ISD_CAMERA_DATA_TYPE structure to ISD_GetCameraConfig().

```
typedef struct
{
    BYTE    TrackingStatus;
    BYTE    bReserved1;
    BYTE    bReserved2;
    BYTE    bReserved3;

    DWORD   Timecode;
    LONG    ApertureEncoder;
    LONG    FocusEncoder;
    LONG    ZoomEncoder;
    DWORD   TimecodeUserBits;

    float   Aperture;
    float   Focus;
    float   FOV;
    float   NodalPoint;

    float   CovarianceOrientation[3];
    float   CovariancePosition[3];

    DWORD   dwReserved1;
    DWORD   dwReserved2;

    float   fReserved1;
    float   fReserved2;
    float   fReserved3;
    float   fReserved4;
}
ISD_CAMERA_ENCODER_DATA_TYPE;
```

TrackingStatus	Tracking status for the current station.
bReserved1	Reserved byte.
bReserved2	Reserved byte.
bReserved3	Reserved byte.
TimeCode	Not currently implemented.
ApertureEncoder	Aperture encoder counts, relative to last reset or power up
FocusEncoder	Focus encoder counts
ZoomEncoder	Zoom endcoder counts
TimecodeUserBits	Time code user bits. Not implemented.
Aperture	Computed aperture value
Focus	Computed focus value (mm), not yet implemented
FOV	Computed Field of View (deg)
NodalPoint	Nodal point offset due to zoom and focus (mm)
CovarianceOrientation	Orientation uncertainty. Available only for IS-1200+
CovariancePosition	Position uncertainty. Available only for IS-1200+
dwReserved1	Reserved DWORD
dwReserved2	Reserved DWORD

fReserved1	Reserved float
fReserved2	Reserved float
fReserved3	Reserved float
fReserved4	Reserved float

ISD_TRACKING_DATA_TYPE

An application uses this data structure to output current data for a station, including position, orientation, time stamp, button, and analog channel state. You pass it to **ISD_GetTrackingData()** as part of ISD_TRACKING_DATA_TYPE.

```
typedef struct {
    ISD_STATION_DATA_TYPE Station[ISD_MAX_STATIONS];
}
ISD_TRACKING_DATA_TYPE;
```

ISD_CAMERA_DATA_TYPE

An application uses this data structure to output camera data for all configured stations. A pointer to a structure of this type is passed to ISD_GetCameraConfig().

typedef struct

{

ISD_CAMERA_ENCODER_DATA_TYPE Camera[ISD_MAX_STATIONS];

}

ISD_CAMERA_DATA_TYPE;

ISD_HARDWARE_INFO_TYPE

An application uses this data structure to output system hardware information when using the **ISD_GetSystemHardwareInfo()** API. For more detailed descriptions of elements in the structure, please refer to the comments in the **isense.h** file.

```
typedef struct {
    Bool    Valid;

    DWORD   TrackerType;
    DWORD   TrackerModel;
    DWORD   Port;
    DWORD   Interface;
    Bool    OnHost;
    DWORD   AuxSystem;
    float   FirmwareRev;

    char    ModelName[128];

    struct {
        Bool    Position;
        Bool    Orientation;
        Bool    Encoders;
        Bool    Prediction;
        Bool    Enhancement;
        Bool    Compass;
        Bool    SelfTest;
        Bool    ErrorLog;

        Bool    UltVolume;
        Bool    UltGain;
        Bool    UltTimeout;
        Bool    PhotoDiode;

        DWORD   MaxStations;
        DWORD   MaxImus;
        DWORD   MaxFPses;
        DWORD   MaxChannels;
        DWORD   MaxButtons;

        Bool    MeasData;
        Bool    DiagData;
        Bool    PseConfig;
        Bool    ConfigLock;

        float   UltMaxRange;
        float   fReserved2;
        float   fReserved3;
        float   fReserved4;

        Bool    CompassCal;
        Bool    AHRS;
        Bool    bReserved3;
        Bool    bReserved4;

        DWORD   dwReserved1;
        DWORD   dwReserved2;
        DWORD   dwReserved3;
        DWORD   dwReserved4;
    }
```

```
    Capability;

    Bool    bReserved1;
    Bool    bReserved2;
    Bool    bReserved3;
    Bool    bReserved4;

    DWORD   BaudRate;
    DWORD   NumTestLevels;
    DWORD   dwReserved3;
    DWORD   dwReserved4;

    float   fReserved1;
    float   fReserved2;
    float   fReserved3;
    float   fReserved4;

    char    cReserved1[128];
    char    cReserved2[128];
    char    cReserved3[128];
    char    cReserved4[128];
}
ISD_HARDWARE_INFO_TYPE;
```

Valid — **TRUE** if **ISD_GetSystemHardwareInfo()** succeeded.

TrackerType — The hardware's type of tracker, a **ISD_SYSTEM_TYPE** name:

- ISD_NONE = 0, Not found, or unable to identify
- ISD_PRECISION_SERIES = InertiaCubes, NavChip, IS-300, IS-600, IS-900 and IS-1200+
- ISD_INTERTRAX_SERIES = InterTrax

TrackerModel — The hardware's model, an **ISD_SYSTEM_MODEL** name:

- ISD_UNKNOWN = 0,
- ISD_IS300 = 3-DOF system (unsupported)
- ISD_IS600 = 6-DOF system (unsupported)
- ISD_IS900 = 6-DOF system
- ISD_INTERTRAX = InterTrax (Serial) (unsupported)
- ISD_INTERTRAX_2 = InterTrax (USB) (unsupported)
- ISD_INTERTRAX_LS = InterTraxLS, verification required (unsupported)
- ISD_INTERTRAX_LC = InterTraxLC (unsupported)
- ISD_ICUBE2 = InertiaCube2
- ISD_ICUBE2_PRO = InertiaCube2 Pro
- ISD_IS1200 = 6DOF system
- ISD_ICUBE3 = InertiaCube3
- ISD_NAVCHIP = NavChip
- ISD_INTERTRAX_3 = InterTrax3 (unsupported)
- ISD_IMUK = K-Sensor
- ISD_ICUBE2B_PRO = InertiaCube2B Pro
- ISD_ICUBE2_PLUS = InertiaCube2 Plus
- ISD_ICUBE_BT = InertiaCube BT

Port — Hardware port number (**1** for **COM1/ttyS0**, etc.).

Interface	Hardware interface (**RS232**, **USB**, etc.) .
OnHost	**TRUE** if tracking algorithms are executed in the library.
AuxSystem	Position tracking type of the hardware (**ISD_AUX_SYSTEM_TYPE**): • ISD_AUX_SYSTEM_NONE = 0, • ISD_AUX_SYSTEM_ULTRASONIC • ISD_AUX_SYSTEM_OPTICAL • ISD_AUX_SYSTEM_MAGNETIC • ISD_AUX_SYSTEM_RF • ISD_AUX_SYSTEM_GPS
FirmwareRev	Firmware revision.
ModelName	Tracker system model name (character string).
Capability	Settings of the hardware:
Position	**TRUE** if the hardware can track position.
Orientation	**TRUE** if the hardware can track orientation.
Encoders	**TRUE** if the hardware can support lens encoders.
Prediction	**TRUE** if the hardware has predictive algorithms are available.
Enhancement	**TRUE** if the hardware enhancement level can be changed.
Compass	**TRUE** if the hardware compass setting can be changed.
SelfTest	**TRUE** if the hardware has the self-test capability.
ErrorLog	**TRUE** if the hardware can keep error log.
UltVolume	**TRUE** if the hardware can control ultrasonic volume via software.
UltGain	**TRUE** if the hardware can control microphone sensitivity by software.
UltTimeout	**TRUE** if the hardware can change ultrasonic sampling frequency.
PhotoDiode	**TRUE** if the hardware's SoniDiscs support photodiode.
MaxStations	Number of supported stations.
MaxImus	Number of supported inertia measurement units (IMUs).
MaxFPses	Maximum number of Fixed Position Sensing Elements (constellation/galaxy).
MaxChannels	Maximum number of analog channels supported per station.
MaxButtons	Maximum number of digital button inputs per station.
MeasData	**TRUE** if the hardware can provide measurement data.
DiagData	**TRUE** if the hardware can provide diagnostic data.
PseConfig	**TRUE** if the hardware supports PSE configuration/reporting tools.
ConfigLock	**TRUE** if the hardware supports configuration locking.
UltMaxRange	Maximum ultrasonic range.
fReserved2	Reserved for future use.
fReserved3	Reserved for future use.

fReserved4	Reserved for future use.
CompassCal	**TRUE** if the hardware supports dynamic compass calibration.
AHRS	**TRUE** if hardware supports AHRS data output (onboard processing of yaw/pitch/roll)
bReserved2	Reserved for future use.
bReserved3	Reserved for future use.
bReserved4	Reserved for future use.
dwReserved1	Reserved for future use.
dwReserved2	Reserved for future use.
dwReserved3	Reserved for future use.
dwReserved4	Reserved for future use.
bReserved1	Reserved for future use.
bReserved2	Reserved for future use.
bReserved3	Reserved for future use.
bReserved4	Reserved for future use.
BaudRate	Serial port baud rate.
NumTestLevels	Number of self test levels.
dwReserved3	Reserved for future use.
dwReserved4	Reserved for future use.
fReserved1	Reserved for future use.
fReserved2	Reserved for future use.
fReserved3	Reserved for future use.
fReserved4	Reserved for future use.
cReserved1	Reserved for future use.
cReserved2	Reserved for future use.
cReserved3	Reserved for future use.
cReserved4	Reserved for future use.

ISD_STATION_HARDWARE_INFO_TYPE

An application uses this data structure to output station (individual tracking device) hardware information using **ISD_GetStationHardwareInfo()**. For more detailed descriptions of elements in the structure, please refer to the comments in the **isense.h** file.

```
typedef struct {
    Bool    Valid;
    DWORD   ID;
    char    DescVersion[20];
    float   FirmwareRev;
    DWORD   SerialNum;
    char    CalDate[20];
    DWORD   Port;

    struct {
        Bool    Position;
        Bool    Orientation;
        DWORD   Encoders;
        DWORD   NumChannels;
        DWORD   NumButtons;
        DWORD   AuxInputs;
        DWORD   AuxOutputs;
        Bool    Compass;
        Bool    bReserved1;
        Bool    bReserved2;
        Bool    bReserved3;
        Bool    bReserved4;
        DWORD   dwReserved1;
        DWORD   dwReserved2;
        DWORD   dwReserved3;
        DWORD   dwReserved4;
    }
    Capability;

    Bool    bReserved1;
    Bool    bReserved2;
    Bool    bReserved3;
    Bool    bReserved4;
    DWORD   Type;
    DWORD   DeviceID;
    DWORD   dwReserved3;
    DWORD   dwReserved4;

    float   fReserved1;
    float   fReserved2;
    float   fReserved3;
    float   fReserved4;

    char    FullSerialNum[17];

    char    cReserved1[111];
    char    cReserved2[128];
    char    cReserved3[128];
    char    cReserved4[128];
}
ISD_STATION_HARDWARE_INFO_TYPE;
```

Valid	**TRUE** if **ISD_GetStationHardwareInfo()** succeeded.
ID	Unique number identifying a station. It is the same number passed to the **ISD_SetStationConfig()** and **ISD_GetStationConfig()** functions and can be a value from 1 to **ISD_MAX_STATIONS**.
DescVersion	Station descriptor version. The descriptor is information set at the factory.
FirmwareRev	Station firmware revision.
SerialNum	Station serial number.
Port	Hardware port number.
Capability	
Position	**TRUE** if station can track position.
Orientation	**TRUE** if station can track orientation.
Encoders	Number of lens encoders, if 0 then none are available.
NumChannels	Number of analog channels supported by this station, wand has 2 (joystick axes).
NumButtons	Number of digital button inputs supported by this station.
AuxInputs	Number of auxiliary input channels (OEM products).
AuxOutputs	Number of auxiliary output channels (OEM products).
Compass	**TRUE** if station has a compass.
bReserved1	Reserved for future use.
bReserved2	Reserved for future use.
bReserved3	Reserved for future use.
bReserved4	Reserved for future use.
dwReserved1	Reserved for future use.
dwReserved2	Reserved for future use.
dwReserved3	Reserved for future use.
dwReserved4	Reserved for future use.
bReserved1	Reserved for future use.
bReserved2	Reserved for future use.
bReserved3	Reserved for future use.
bReserved4	Reserved for future use.
Type	Station type.
DeviceID	Link ID for the wireless device.
dwReserved3	Reserved for future use.
dwReserved4	Reserved for future use.
fReserved1	Reserved for future use.
fReserved2	Reserved for future use.

fReserved3	Reserved for future use.
fReserved4	Reserved for future use.
FullSerialNum	Full serial number of device (null-terminated character array)
cReserved1	Reserved for future use.
cReserved2	Reserved for future use.
cReserved3	Reserved for future use.
cReserved4	Reserved for future use.

ISD_PORT_WIRELESS_INFO_TYPE

An application uses this data structure to get information about the wireless hardware on a given port, using **ISD_GetPortWirelessInfo()**. You can use the **radioVersion** field to check the type of radio hardware:

- 2.4 GHz (Chipcon, MicroTrax only): **128**
- 900 MHz (MicroTrax only): **144**

```
typedef struct {
    Bool    valid;

    LONG    status;
    Bool    wireless;
    DWORD   channel;
    DWORD   id[4];
    DWORD   radioVersion;

    DWORD   dReserved1;
    DWORD   dReserved2;
    DWORD   dReserved3;
    DWORD   dReserved4;
}
ISD_PORT_WIRELESS_INFO_TYPE;
```

valid	**TRUE** if **ISD_GetStationHardwareInfo()** succeeded.
status	Current connectivity of wireless device to the port: • **0** = Port unknown. • **1** = Port not available. • **2** = No device connected. • **3** = Device connected.
wireless	**TRUE** if the station is wireless.
channel	Channel number setting for the port.
id	Link ID for the wireless device, up to four per port. **0** if no wireless device is connected.

radioVersion	The wireless radio type: • **128** = 2.4 GHz (MicroTrax only) • **144** = 900 MHz (MicroTrax only)
dReserved1	Reserved for future use.
dReserved2	Reserved for future use.
dReserved3	Reserved for future use.
dReserved4	Reserved for future use.

ISD_STATION_MAGCAL_TYPE

An application uses this data structure to output current data for a station, including position, orientation, time stamp button and analog channel state. You pass it to ISD_StationMagCalStatus().

```
typedef struct
{
    Bool   Valid;
    Bool   Dynamic;

    float  MagBiasFactory[3];
    float  MagBiasCalStored[3];

    float  MagScaleFactory[3];
    float  MagScaleCalStored[3];

    float  MagMisAlignFactory[3][3];
    float  MagMisAlignCalStored[3][3];

    struct
    {
        Bool   InProgress;
        DWORD  NumSamples;
        DWORD  NumSamplesRequired;

        Bool   MagCalComputed;
        float  MagScaleCal[3];
        float  MagMisAlignCal[3][3];
        float  MagBiasCal[3];

        float  MagMin[3];
        float  MagMax[3];
        float  MagTemp;

        double Accuracy;
    }
    Progress;

    Bool    bReserved1;
    Bool    bReserved2;
    Bool    bReserved3;
```

```
    Bool    bReserved4;

    DWORD   dwReserved1;
    DWORD   dwReserved2;
    DWORD   dwReserved3;
    DWORD   dwReserved4;

    float   fReserved1[3];
    float   fReserved2[3];
    float   fReserved3[3];
    float   fReserved4[3];
} ISD_STATION_MAGCAL_TYPE
```

Valid	Valid if ISD_StationMagCalStatus() completed successfully..
Dynamic	True if this is a dynamic calibration.
MagBiasFactory	Factory values for bias calibration.
MagBiasCalStored	Current calibration value for bias.
MagScaleFactory	Factory values for scale calibration.
MagScaleStored	Current calibration value for scale.
MagMisAlignFactory	Factory values for calibration misalignment.
MagMisAlignStored	Current calibration value for misalignment.
Progress	
InProgress	True if a calibration is in progress.
NumSamples	Total number of samples collected.
NumSamplesRequired	Number of samples required to complete calibration.
MagCalComputer	True if the magnetic calibration has been computed.
MagScaleCal	Output values from calibration for scale.
MagMisAlignCal	Output values from calibration for misalignment.
MagBiasCal	Output values from calibration for bias.
MagMin	Output values from calibration for minimum amplitude.
MagMax	Output values from calibration for maximum amplitude.
MagTemp	Temperature in Celsius.
Accuracy	Weighted accuracy of the calibration process.
bReserved1	Reserved for future use.
bReserved2	Reserved for future use.
bReserved3	Reserved for future use.
bReserved4	Reserved for future use.
dReserved1	Reserved for future use.
dReserved2	Reserved for future use.
dReserved3	Reserved for future use.
dReserved4	Reserved for future use.

fReserved1	Reserved for future use.
fReserved2	Reserved for future use.
fReserved3	Reserved for future use.
fReserved4	Reserved for future use.

ISD_GPS_DATA_TYPE

An application uses this data structure to output current data for a station, including position, orientation, time stamp button and analog channel state. You pass it to ISD_StationMagCalStatus().

```
typedef struct
{
    struct
    {
        BYTE format;       // GPS sentence format identifier.
        UINT timestamp;    // GGA timestamp (HH:MM:SS from GPS converted to sec)
        double latitude;      // GGA latitude
        double longitude;     // GGA longitude
        float altitude;       // Meters
        UINT16 nSatellites; // Satellite count
        BYTE quality;      // GPS fix quality. 1 = GPS fix
        float geoIdHeight;    // Height of sea level (meters)
        float hdop;           // Horizontal dilution of precision.
    }
GGA;
    struct
    {
        UINT timestamp;    // Minimum Recommended timestamp (HH:MM:SS from GPS converted
                              to sec)
        double latitude;      // Minimum Recommended Latitude
        double longitude;     // Minimum Recommended Longitude
        float speed;          // m/s
        float bearing;        // degrees
        float magVar;              // Variation from magnetic north (degrees)
        Bool dataActive;    // Is GPS data good?
    }
    RMC;
} ISD_GPS_DATA_TYPE;
```

GGA

format	GPS Sentence format identifier.
timestamp	GGA timestamp (HH:MM:SS from GPS converted to seconds)
latitude	Latitude from GGA message.
longitude	Longitude from GGA message.
altitude	Altitude in meters.
nSatellites	Number of satellites used for tracking.
quality	GPS fix quality, 1 = GPS fix.
geoIdHeight	Height of sea level (meters).
hdop	Horizontal dilution of precision.

RMC

timestamp	RMC timestamp (HH:MM:SS from GPS converted to seconds).
latitude	Latitude from RMC message.
longitude	Longitude from RMC message.
speed	Speed in meters per second.
bearing	Bearing in degrees.
magVar	Variation from magnetic north in degrees.
dataActive	True if current GPS data is valid.

ISD_IMAGE_FORMAT_TYPE

Enum which specifies the image format retrieved from ISD_GetImage() or ISD_GetImageInfo().

```
typedef enum
{
    ISD_VGA_GRAY_8 = 10,
    ISD_VGA_GRAY_16,
    ISD_VGA_YUYV_8,
    ISD_WVGA_GRAY_8 = 20,
    ISD_WVGA_GRAY_16,
    ISD_SXGA_GRAY_8 = 30,
    ISD_SXGA_GRAY_16,
    ISD_UVGA_GRAY_8 = 40,
    ISD_UVGA_GRAY_8_UND = 41,
    ISD_VGA_24 = 50,
    ISD_UVGA_24,
} ISD_IMAGE_FORMAT_TYPE;
```

Terminology

IS-900 models contain an ultrasonic subsystem that includes SoniDiscs (Ultrasonic Emitters or Beacons) and Microphones (Ultrasonic Receiver Modules – URMs). To generalize the interface protocol and configuration tools for these tracker models, these components are referred to as Position Sensing Elements (PSEs).

A PSE may be Mobile or Fixed. Mobile PSEs are assigned to the stations; the system tracks their movements (i.e. MicroTrax Microphones). Fixed PSEs form a constellation used as a reference for tracking (i.e. SoniStrips & SoniDiscs).

2.6. Commands Sent from the Host to the Tracker

<>	Carriage return line feed pair – not needed for single character commands. CR – ASCII value 13, LF – ASCII value 10.
{}	List of parameters required for command.
[]	List of optional parameters for command. Omitting those results in a query.

Since the IS-900 emulates most (but not all) of the commands in the Polhemus Fastrak™ protocol, you can use the IS-900 with most applications without writing new driver code. Several additional commands access some of the advanced features of the IS-900 that do not have any counterpart in the Fastrak™ protocol.

NOTE: Firmware version 3.xx extended Fastrak™ protocol to support up to 32 stations (actual number allowed is determined by your hardware configuration). StationNum in commands, status and data records is now encoded in an extended hexadecimal notation. Numbers 1 to F conform to standard hexadecimal notation, with numbers greater that F represented by additional upper case letters of the alphabet. For example, the number 16 is displayed as G.

2.7. Standard Fastrak™ Interface Commands

Data Record Request

P Request a data record from all active stations.
Only used in polled mode.

Output Mode

C Put in continuous output mode.

c Put in polled output mode.

Default **Polled mode**

Alignment Reference Frame

A{stationNum},[Ox,Oy,Oz,Xx,Xy,Xz,Yx,Yy,Yz]<>

Sets the coordinate frame with respect to the outputs for the station that will be reported. The coordinate frame is defined by a set of three points. Ox,Oy,Oz defines the origin of the new coordinate system, Xx,Xy,Xz defines a point on the positive x axis and Yx,Yy,Yz defines a point on the positive y axis. The points on the positive X and positive Y axis are in the new coordinate frame rather than the original coordinate frame. Values are not cumulative/incremental, so any future changes are relative to the original coordinate system.

For example, in order to configure station 1 to use a new origin of (50, 60, 70) cm with the coordinate frame rotated 90° about the +z axis, the command would be A1,50,60,70,50,61,70,49,60,70 (adding +1cm to the Xy value and -1cm to the Yx value). Any length of vector along +x and +y may be used, as these are normalized internally to a length of 2m (200cm), although for simplicity we recommend adding the same value, in the example, however, the command A1,50,60,70,50,400,70,12,60,70 (adding +340cm to Xy and -38cm to Yx) would result in the same translation/rotation of the coordinate system. If you omit optional parameters in the call, the command outputs current values. Units are centimeters.

Default **(in Fusion Mode) X=North, Y=East, Z=Down**

Position origin is defined by SoniStrip constellation array horizontally leveled. For details on SoniStrip Arrays, see the *IS-900 SimTracker, VETracker, and SimTracker LT User Guide* for details.

Reset Alignment Reference Frame

R{stationNum}<>
Resets reference frame to the default.

Boresight Reference Angles

G{stationNum},[yawref, pitchref, rollref]<>

Sets the boresight reference angles for the specified station. If set, the next Boresight command uses these values instead of current orientation. If you omit optional parameters, the command outputs current reference angles. Units are degrees.

Default **0,0,0**

Boresight Compatibility Mode

MBF<> Switch system to Fastrak™ Compatible mode.

MBI<> Switch system to Firmware Version 2.x Compatible mode.

In firmware versions prior to 3.00 the B{stationNum}<> command was implemented as the Heading Boresight (see below) and full boresight was not available. To maintain compatibility with the user software written at that time, two Boresight Compatibility modes are available. In Fastrak™ Compatible mode *B{stationNum}*<> command executes full 3-DOF boresight and *MB{stationNum}*<> effects heading only. In the Firmware Version 2.x Compatible mode the meanings of these commands are reversed.

Default **Firmware Version 2.x Compatible, *MBI*<>**

Boresight

B{stationNum}<> (Fastrak™ compatibility mode)

MB{stationNum}<> (Firmware Version 2.x compatibility mode)

Boresight a station. If a *G{stationNum}[yawref, pitchref, rollref]*<> command has set boresight reference angles prior to issuing of Boresight command, then that orientation becomes the new reference point. The angles the tracker outputs at that orientation become zero. Otherwise, system uses current station orientation and that becomes the new reference line of sight.

Be sure that the object being tracked (like an HMD) is leveled and is pointing down the x axis when boresighting a station.

Unboresight

b{stationNum} <> (Fastrak™ compatibility mode)

Mb{stationNum}<> (Firmware Version 2.x compatibility mode)

Unboresight a station. Reference angles are cleared for the specified station.

Heading Boresight

B{stationNum}<> (Firmware Version 2.x compatibility mode)

MB{stationNum}<> (Fastrak™ compatibility mode)

This command has no effect with the IS-900.

Heading Unboresight

b{stationNum}<> (Firmware Version 2.x compatibility mode)

Mb{stationNum}<> (Fastrak™ compatibility mode)

This command has no effect with the IS-900.

Set Serial Communication Parameters

o{rate,parity,bits,HHS}<> Change serial communication parameters.

rate Is one of 3,12,24,48,96,192,384,576,1152
(rate is multiplied by 100)

parity N = none
O = odd
E = even

bits 7 or 8

HHS (Hardware handshake)
0 = OFF
1 = ON

Default serial communications settings:

rate	1152
parity	N
bits	8
HHS	OFF

System Record Request

S Request a system status record be sent.

Station Status
l{stationNum},[state]<>

Set the stationNum to on or off.

state 0 = OFF, 1 = ON

Default **All connected stations are on.**

Output Units Control

U Sets output data record position units to inches.

u Setsposition units to centimeters. Setting only matters in 6-DOF mode.

Default **U**

System control

^*K*	Save the current settings to nonvolatile memory.
W	Restore the system settings to the factory defaults.
^Y	Restart the firmware to the power up condition.
^*S*	Suspend data transmission.
^*Q*	Resume data transmission.

Caution: Sending **W** command causes loss of all user configuration data.

Output Record Mode

F Put in ASCII output mode.

f Put in Binary output mode.

Default **F**

Output Record List Settings

O{stationNum},[p1],[p2],[p3],.....,[pn]<>

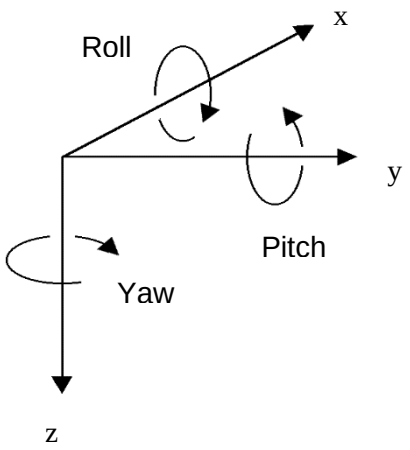
Sets the output data list for *stationNum*. If optional parameters are omitted, a data record containing current output list settings for the station is output.

Default **2,4,1**

Item	Description	Format
0	ASCII space character	1 ASCII byte
1	ASCII CR, LF pair	2 ASCII bytes
2	x, y, z position coordinates	3 floats
4	yaw, pitch, roll Euler angles	3 floats
5	x axis direction cosines	3 floats
6	y axis direction cosines	3 floats
7	z axis direction cosines	3 floats
11	orientation quaternion	4 floats
16	stylus switch status (always 0)	1 byte (ASCII/binary)
18	x, y, z in 16 bit binary format	see below
19	yaw, pitch and roll in 16 bit binary format	see below
20	quaternion in 16 bit binary format	see below
21	time stamp, in selected time units	see section 2.9.1
22	buttons	see below
23	joystick	see below
29	AngularVelNavFrame	3 floats
31	AngularVelBodyFrame	3 floats
32	AccelNavFrame	3 floats
33	AccelBodyFrame	3 floats
34	VelocityNavFrame	3 floats
36	AngularVelRaw	3 floats
40	tracking status	0-255
68 – 71	auxiliary inputs	
75	communication integrity	0-100
76	measurement quality	0-100

Data Item 4 – Euler Angles

The Euler angles are defined as rotations about Z, then Y, then X in body frame. Angles are output in degrees.



Data Items 5, 6, 7 – Direction Cosines

You can use direction cosines to construct a 3x3 rotation matrix:

$$\begin{pmatrix} x1 & x2 & x3 \\ y1 & y2 & y3 \\ z1 & z2 & z3 \end{pmatrix}$$

X direction cosines.

Y direction cosines.

Z direction cosines.

You can also construct this matrix from Euler angles:

$$\begin{pmatrix} cos(P)*cos(Y) & sin(R)*sin(P)*cos(Y) - cos(R)*sin(Y) & cos(R)*sin(P)*cos(Y) + sin(R)*sin(Y) \\ cos(P)*sin(Y) & sin(R)*sin(P)*sin(Y) + cos(R)*cos(Y) & cos(R)*sin(P)*sin(Y) - sin(R)*cos(Y) \\ -sin(P) & cos(P)*sin(R) & cos(P)*cos(R) \end{pmatrix}$$

Data Item 11 – Orientation Quaternion

Quaternion is output as q = [w, x, y, z]. To convert from Quaternion to rotation matrix, apply the following formula:

$$\begin{pmatrix} 1-2y^2-2z^2 & 2xy-2wz & 2xz+2wy \\ 2xy+2wz & 1-2x^2-2z^2 & 2yz-2wx \\ 2xz-2wy & 2yz+2wx & 1-2x^2-2y^2 \end{pmatrix}$$

Data Items 18, 19, 20 – 16- bit binary format.

16-bit binary format can be used in applications requiring fastest possible serial I/O. Each floating point number is stored in 2 bytes with only 14 bits containing actual data. This results in lower accuracy than the standard IEEE floating point format.

Data is 2's-complement. The first byte of the data set has its high-order bit set to 1; all others have them set to zero. You can use this setup for data synchronization. Data is output low-order byte, then

high-order byte. Use following code sample as an example of how to decode this format:

To decode position:

```
lo = (dataRecord[3] & 0x007F);
hi = (dataRecord[4] & 0x007F);
int14bit = (lo << 2) | (hi << 9);
result = (float) int14bit * 3.0 / 32768.0;
```

Result is a number representing position (in meters) and has a full range of ± 3.0 meters (–300.0 to+299.963 centimeters or –118.110 to 118.096 inches).

To decode Euler angles:

```
lo = (dataRecord[3] & 0x007F);
hi = (dataRecord[4] & 0x007F);
int14bit = (lo << 2) | (hi << 9);
result = (float) int14bit * 180.0 / 32768.0;
```

Resulting number represents orientation and has a full range of ± 180.0 (–180.0 to +179.978)°.

To decode Orientation Quaternion:

```
lo = (dataRecord[3] & 0x007F);
hi = (dataRecord[4] & 0x007F);
int14bit = (lo << 2) | (hi << 9);
result = (float) int14bit * 1.0 / 32768.0;
```

Resulting quaternion value has range of ± 1.0.

Data Item 22 – Buttons.

One 3-digit integer in ASCII format or one byte in binary format.

Bits represent the button states of a station's buttons. If a button is pressed, the corresponding bit is 1, otherwise it is 0. The bit assignments for the wand are (where bit 0 is the least significant bit):

Bit	Button	MicroTrax Wand button
0	1	1 on wand image
1	2	2 on wand image
2	3	3 on wand image
3	4	4 on wand image
4	5	center (press joystick)
5	6	trigger

Data Item 23 – Joystick.

Two integers, one for each axis, values ranging from 0 to 255. Two 3-digit integers in ASCII format or two 1-byte unsigned values in binary format. Values at limits and at center are:

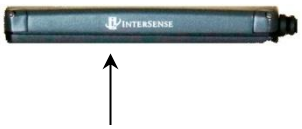
Axis	Position	Value
left/right (1st integer)	left	0
	center	127
	right	255
front/rear (2nd integer)	rear	0
	center	127
	front	255

Define Tip Offsets

N{stationNum},[Ox, Oy, Oz]<>

By default, the point being tracked is for each station is:

Wand Station: Tip.



Tip Offset: intersection of front surface and bottom surface, and at mid length.

Head Tracker Station: Intersection of front surface, bottom surface and at mid length, see image to the left.

Hand Tracker Station: center top, in dent.

Use this command to define a set of position offsets, so a different point can be tracked. Offsets are measured in the body coordinate frame of the MicroTrax station and are entered in centimeters. If optional parameters are omitted, current settings are output.

Default **0,0,0**

Position Operational Envelope

V{stationNum},[Xmax, Ymax, Zmax, Xmin, Ymin, Zmin]<>

This command sets the boundaries of the area where position is to be tracked. Whenever a station leaves the defined range, position tracking stops and only resumes after it is back within the defined boundaries. Enter the parameters in meters. If you omit optional parameters, the command outputs current settings. Units are meters.

Default **200,200,200,-200,-200,-200**

2.8. Fastrak™ Commands Implemented for Compatibility

Hemisphere

H{stationNum},[p1,p2,p3]<>

Sets the tracking hemisphere for a magnetic tracking system. Because InterSense trackers are not magnetic the parameters are ignored. However, you can set them and then query them for compatibility with software such as MultiGen SmartScene or Immersion Corporation haptic Software. If you omit optional parameters, the command outputs a data record containing current Hemisphere settings for the station.

Default **1,0,0**

2.9. InterSense-Specific Commands

All InterSense-specific commands start with the letter M (for *Manufacturer-specific*) and must be completed by a CR,LF pair.

2.9.1. System Configuration Commands

Time Units

Time stamp recorded is the time when the tracker data was collected from the hardware. The time index is set to zero when tracker is first turned on.

MT<> Sets the units for the data record time stamp to milliseconds.

Mt<> Sets the units for the data record time stamp to microseconds.

Default T

Set Current Time to Zero

MZ<> This command sets current time index of the tracker to zero.

Set Ethernet Communication Parameters

*MEthMode{*mode*}*<> Sets Ethernet configuration mode.
2=DHCP, 1=manual, 0=disabled

MEthIp{address}<> Sets system IP address (manual mode).
Use dotted format like 192.168.1.1.

MEthSubnet{subnet}<> Sets subnet mask (manual mode).

MEthUdp{state}<> Sets state of UDP output. 1=enabled, 0=disabled

MEthUdpPort{port}<> Sets UDP output port.

MEthUdpIp{address}<> Sets UDP output IP address. Dotted address=unicast, 0=broadcast

MEthTcpPort{port}<> Sets TCP communication port.

MEthApply<> Applies Ethernet settings.

Changes to IP address, subnet mask, and TCP port do not take effect until they are applied. (Settings must still be saved to store them into non-volatile memory.)

If the parameter to the above commands is omitted, the command outputs the current respective setting in a record starting with a **31** header.

Default Ethernet communication settings:

Mode	**Manual**
IP address	**None**
Subnet	**255.255.255.0**
UDP state	**OFF**
UDP IP address	**None**
UDP port	**5001**
TCP port	**5005**

InterSense System Status Record Request

MS<> Request manufacturer-specific system status record.

This record is information about parameters specific to the InterSense product, additional to the standard system status information obtained using the **S** command.

Tracking Status Record Request

MP<> Requests tracking status information for all 12 stations. See the data structure's **`TrackingStatus`** parameter for the description

Ultrasonic Timeout Interval

MU[interval]<> This command has no effect with the IS-900.

Default **N/A**

Ultrasonic Receiver Sensitivity

Mg[Level]<> This command has no effect with the IS-900.

Default **N/A**

Genlock Synchronization

MG[State, Rate]<>

State 0 – Genlock is off (free run).
1 – Reserved.
2 – External sync, manual (supply strobe rate).
3 – Internal sync, supply output record rate.

Rate Value in Hertz used with *State* = 2 and 3.

Default **0**

Genlock Phase

MGP[Param]<>

Param can be one of these:

The Phase (0 to 100%).

+ to increase to the next phase point.

– to decrease to the next phase point.

Please see for the Appendix on Genlock in the *IS-900 SimTracker, VETracker, and SimTracker LT User Guide* for complete details.

Default **0**

Genlock Sync Source

MGS[source]<>

source 1 – TTL
2 – NTSC

Configuration Lock

Use configuration lock commands to prevent unintentional changes to tracker configuration. This command provides two levels of protection. The first level prevents changes to saved settings. The second level prevents changes to current unsaved (session) settings as well as saved settings.

MConfigLockMode[mode]<>

mode 0 – Lock off
1 – Lock saved settings
2 – Lock saved and session settings

Default 0

In mode 1 (lock saved settings), the Fastrak commands to save current (^*K*) and restore factory (*W*) settings are disabled. In mode 2 (lock saved and session settings), the Fastrak command to restore saved settings (^*Y*) is disabled as well as ^*K* and *W* commands. In mode 0 (lock off), ^*Y*, ^*K* and *W* are all enabled. When mode is changed, it is saved to nonvolatile memory without affecting any other saved settings. If mode is omitted, the current setting is output. The LCD menu options to save and restore settings are not affected by Lock mode.

SoniStrip LED control

ML[state]<>

If *state* is 0, the blue LEDs on the SoniStrips are disabled. It is important to have these LEDs to visually confirm that the ultrasonic system is operating, so don't disable the LEDs unless they interfere with your application.

Default 1

Beacon Scheduler

MSchAlg[n]<>

Selects beacon Scheduling Algorithm. If *n* is 1, uses a distance-based algorithm. This algorithm chooses beacons based on distance only. If *n* is 2, uses a directional algorithm, which chooses beacons based on orientation as well as distance.

Default 2

Error Reporting

Systems store hardware and configuration errors internally and can report them to the application. If the application is not setup to accept error messages, or if you are not sure whether it is, you should disable error reporting. This setting is not saved with tracker configuration, so error reporting is off every time tracker is turned on.

ME<> Outputs all errors, one per error message.

MEC<> Clear all errors from internal list.

ME1<> Enable error reporting.

ME0<> Disable error reporting.

Default Error reporting is OFF

Command Logging

Command logging captures all host commands into a file for debugging purposes. The log file holds a maximum of 500 kB. When the max size is reached, the file is rewound and overwritten with new entries. Long series of P and MP commands are abbreviated to save space. The command logging state is also indicated on the LCD (see *IS-900 SimTracker, VETracker, and SimTracker LT User Guide* for LCD screen displays). Command logging control is available on the LCD menu under **System Config → Command Log: Enable Log, Disable Log** and **Clear Log**. You can access it with the following command set:

MLogOpen<>	Enables logging. If settings are saved, logging will remain on through reset. An existing log file is always appended to.
MLogClose<>	Disables logging.
MLogClear<>	Disables logging and deletes log file. Use *MLogOpen* to resume logging.
MLogState<>	Returns logging state (0=off, 1=on), 31LS{0,1}<>
MLogSend<>	Outputs log file to host one command per line. 31LF<timestamp in ms>:<command><> timestamp is a decimal number and is not zero-padded. log file can alternatively be retrieved using ISDEMO.

(For details on ISDEMO, see the *IS-900 SimTracker, VETracker, and SimTracker LT User Guide*).

2.9.2. InterSense-Specific Station Parameters

InterSense Station Status Record Request

Ms{stationNum}<>

Request an individual sensor status record for *stationNum*. This is information about parameters specific to the InterSense product.

Prediction Interval

Mp{stationNum},[Interval]<>

Sets the time-interval of prediction for *stationNum*. Interval is an integer number of time in milliseconds. Suggested range is 0 to 50 ms. You use this parameter for both position and orientation prediction. If you omit an optional parameter, the command outputs its current prediction value.

Default 0

Perceptual Enhancement Level

MF{stationNum},{Mode}<>

To provide the best performance for a large range of various applications, 3 levels of perceptual enhancement are available. None of the modes introduces any additional latency.

Mode 0 – Provides the best accuracy. Makes drift correction adjustments immediately; uses no jitter reduction algorithms. This results in somewhat jumpy output (not recommended for head-tracking) but with lower RMS error. Use this mode for accuracy testing or for any application that requires best accuracy.

Mode 1 – Provides accuracy similar to that of mode 0, with an addition of a jitter reduction algorithm. This algorithm reduces the accuracy by only a small amount and does not add any latency to the measurements.

Mode 2 – Recommended for use with HMD or other immersive applications. Makes drift correction adjustments smoothly and only while the sensor is moving, so they remain transparent to the user.

Default 2

Compass Heading Correction

MH{stationNum, mode}<>

Turns on the ***stationNum's*** compass heading correction. To operate effectively, the magnetic field in the environment needs to be homogeneous.

NOTE: Only valid when an InertiaCube is being used as a station (i.e. not valid for MicroTrax devices).

For an InertiaCube, the modes are:

Mode 0 – Compass is OFF. Does not apply heading compensation. Not recommended; Modes 1 or 2 are preferred.

Mode 1 – Partial compass mode. Uses magnetometer readings to reduce drift and maintain stability, but not as an absolute measurement system. In this mode, system is much less susceptible to magnetic interference, but heading drift slowly accumulates. This mode is particularly useful when you are using high rotational sensitivity settings.

Mode 2 – FULL compass mode. Uses readings that the magnetometers inside the InertiaCube produce as the absolute reference orientation for yaw.

Default 2

Mh{stationNum}<>

Turns off the *stationNum's* compass heading correction. There can be slow drift in the yaw direction. **Only valid when an InertiaCube is being used as the station; not valid for MicroTrax devices.**

Rotational Sensitivity Level

MQ{stationNum},[Sensitivity Level]<>

Adjusts rotational sensitivity of a station when the Perceptual Enhancement Level is set to level 2. Sensitivity Level is an integer 1 to 4 where 1 is the lowest and 4 is the highest sensitivity. Thales Visionix recommends that you set sensitivity to 4 when enhancement is set to level 1. If you omit an optional parameter, the command outputs its current value.

Default 3

2.9.3. Station and Constellation Configuration Commands

This section describes the commands used to assign sensing devices to the logical components of the tracking system. Such devices with the IS-900 are SoniStrips and the SoniDiscs (beacons) that make up the constellation array. All commands in this section start with MC.

A Configuration Session is the period during which MC commands are received and accepted. It starts when the first MC command arrives and ends when these commands are explicitly activated with the MCe<> (or discarded with the MCx<>) command.

Associate Fixed PSE with a Constellation

MCF{FPSE number},{xp, yp, zp, xn, yn, zn, IDcode}<>

Configures or adds a Fixed PSE. For IS-900 FPSE is a SoniDisc ultrasonic transponder beacon. If only the FPSE number parameter is present, then the command outputs current data for that PSE on the serial port. If that PSE is not configured, the output record contains Hardware ID of −1. If you provide no parameters, then the command outputs data for all configured Fixed PSEs, each PSE in a separate record. This command does not take effect until the *MCe<>* command explicitly activates it.

FPSE number	A Unique number identifying a Fixed PSE (beacon) within a constellation (a complete set of fixed PSEs). Numbering starts at 1.
xp, yp, zp	FPSE position in meters.
xn, yn, zn	Normal vector.
IDcode	Hardware ID of the PSE.
<>	CR LF pair.

Disassociate Fixed PSE from the Constellation

MCf[Fixed PSE number, IDcode]<>

If PSE number and *IDcode* are not associated in the current configuration, this command is ignored. This command does not take effect until explicitly activated by the *MCe<>* command.

Clear All Fixed PSEs (Constellation) Command

MCC<>

Delete all Fixed PSEs from the configuration. This command does not take effect until explicitly activated by the *MCe<>* command.

Apply New Configuration

MCe<>

Reconfigures the system with the new MicroTrax station and SoniDisc information. A Configuration Session is the period during which the processor receives and accepts the station and constellation configuration commands. The configuration session starts when the first *MC* command arrives. *MC* commands are saved but not applied to the system configuration until the *MCe<>* command is received. This *MCe<>* command executes all *MC* commands that have been received since the start of the Configuration Session and computes the new system state.

Cancel Configuration Session

MCx<>

Cancels the configuration session command. Discards all *MC* commands received during the session.

2.10. Records Output from the Tracker to the Host

2.10.1. Format Considerations

Record Headers

The first byte of each record identifies its type.

0 – Data record.
2 – Fastrak™ status record.
3 – InterSense manufacturer-specific status record.

Floating Point Numbers

Floating point numbers can be output as IEEE 32 bit floats or as ASCII numbers in X.xf notation, where:

X Total number of characters used to represent the float.
x Number of digits after the floating point.
f Symbol indicating that number is a float.

For example, number -42.6 in 10.4f format would look as follows: “-42.6000”

2.10.2. Status Record Hexadecimal Character Decoding

System Status, Manufacturer Status, and Manufacturer Station records use Hexadecimal Characters to encode status data. Each character can be 0 to F and can encode 4 bits. Logical AND operator can be used to test specific bits. Please see following code example:

```
unsigned short byte1, byte2, byte3;
char hexChar[2];

hexChar[1] = 0x00;

hexChar[0] = statusRecordBuffer[3];
sscanf(hexChar,"%x", &byte1);
hexChar[0] = statusRecordBuffer[4];
sscanf(hexChar,"%x", &byte2);
hexChar[0] = statusRecordBuffer[5];
sscanf(hexChar,"%x", &byte3);
```

2.10.3. Fastrak™ System and Data Records

Data Record

System sends this record in response to the *P* command in Polled mode or continuously in Continuous mode. Sends a separate data record for each active station. The list of data items in each station record depends on how the list was set up with the **O** command. For most of the commonly used data list items, the format depends on commands **F**, **f**, **U**, **u**, **MT**, **Mt**.

0{stationNum}{status}{dataItem1,dataItem2,....}

stationNum A hexadecimal number up to C (decimal 12)

status An ASCII space character. This status byte is currently unused.

dataItemX

ASCII – In ASCII numbers, set with the F command, any numbers output as floats are output as ASCII numbers. Each number is 7 ASCII characters: a sign, 3 digits, a decimal point and 2 more digits. If programming in C use **scanf("%7.2f", &value)** to read them.

Binary – In Binary mode, set with the **f** command, the floats are sent as 4 byte IEEE 32-bit floats. Note that these floats are little endian.

Time Stamp 32 bit floating point number

ASCII – In ASCII mode, it is allowed 14 characters and displayed as an integer (digits after the floating point are ignored).

Binary – In binary mode, it is output as a 32 bit float. Note that these floats are little endian.

Binary16 mode

Set by selecting 18, 19 and/or 20 only with the **O** command. This is the fastest way to get data but the most difficult to decode and the least accurate.

Example:

For the default data record O1,2,4,1<> in ASCII mode **F**, the output record set for active Stations 1 and 2 would look as follows:

```
'01    1.23   41.83   12.18   13.04   76.11  34.12CRLF'
'02   23.01 -452.94    0.01   -1.01   23.32  12.34CRLF'
```

Station 1: x=1.23, y=41.83, z=12.18, yaw=13.04, pitch=76.11, roll=34.12

Station 2: x=23.01, y=-452.94, z=0.01, yaw=−1.01, pitch=23.32, roll=12.34

System Status Record

System sends this record in response to the **S** command. It contains system wide status information. Some information in the status record requires bit decoding.

21S{statusRecord}<>

Bytes	Explanation
1	Record type, 2.
2	Station Number, always 1.
3	Sub-record type, S.
4	Config Hex Char 0.
5	Config Hex Char 1.
6	Config Hex Char 2.
7 – 9	BIT error. Currently unused.
10 – 15	Blank.
16 – 21	Firmware version ID
22 – 53	System identification.
54 – 55	CR, LF.

To decode each of the Config bytes:

Config Hex Char 0 – Unused
Config Hex Char 1 – Unused
Config Hex Char 2 – See table below for possible bit settings.

Bit	Meaning
0	Output Format (0 = ASCII, 1 = Binary).
1	Output Units (0=Inches, 1=Centimeters).
2	Unused.
3	Transmit Mode (0 = Polled, 1 = Continuous).

Output List Record

System sends this record in response to an *O{stationNum}<>* command. The command outputs the list of currently selected output parameters for that station, 2 bytes per item.

2{stationNum}O{par1par2...parN}<>

Bit	Explanation
1	Record type, 2
2	Station Number. A hexadecimal number up to C.
3	Sub-Record type, O.
4 – 5	*par1.*
6 – 7	*par2.*
…	CR, LF.

For example, the default data list would be output as:

```
'21O 2 4 1<>'
```

Station State Record

System sends this record in response to an *l{stationNum}*<> command.

If you omit the *stationNum* parameter, the record outputs the state of the first four stations. This action maintains compatibility with the Fastrak™ protocol. If you pass *stationNum*, then byte 4 of the record contains the state of the requested station. Bytes 5, 6, and 7 contain the state of stations 2, 3, and 4. If you replace *stationNum* with the * wild-card character, then the command outputs theh status for 32 stations in a single record.

Output of values are interpreted as 1 for ON and 0 for OFF.

2{stationNum}l{stat1stat2stat3stat4}<>

Bytes	Explanation
1	Record type, 2.
2	Station Number, 1, 2, 3, 4
3	Sub-record type, i.
4	State of station *stationNum*, or station 1 if omitted.
5	State of station 2.
6	State of station 3.
7	State of station 4.
8 – 35 (IF '*')	Status of stations 5 to 32.
8, 9 OR 36, 37	CR, LF.

For example, if stations 1 and 3 are on:

l1<> command will return **'2111010<>'**

*l**<> command will return **'21110100000000000000000000000000000<>'**

Alignment Reference Frame Record

System sends this record in response to an *A{stationNum}*<> command. It outputs 9 ASCII floats that were last set with the *A* command. Each float is represented as seven characters with two digits after the floating point. Units are centimeters.

2{stationNum}A{OxOyOzXxXyXzYxYyYz}<>

Bytes	Explanation
1	Record type, 2.
2	Station Number. A hexadecimal number up to C.
3	Sub-record type, A.
4 – 24	*OxOyOz* **–** origin coordinates.
25 – 45	*XxXyXz* **–** vector in the direction of the positive x axis.
46 – 66	*YxYyYz* **–** vector in the direction of the positive y axis.
67 – 68	CR, LF.

Boresight Reference Angles Record

System sends this record in response to a *G{stationNum}* command. It outputs three ASCII floats that were last set with the **G** command. Each float is represented as seven characters with two digits after the floating point. Units are degrees.

2{stationNum}G{yawref pitchref rollref}<>

Bytes	Explanation
1	Record type, 2.
2	Station Number. A hexadecimal number up to C.
3	Sub-record type, G.
4 – 10	*yawref* – azimuth reference angle.
11 – 17	*pitchref* – elevation reference angle.
18 – 24	*rollref* – roll reference angle.
25 – 26	CR, LF.

Hemisphere Record

System sends this record in response to an *H{stationNum}*<> command.

It outputs 3 ASCII floats that were last set with the *H* command. Each float is represented as 7 characters with 2 digits after the floating point.

2{stationNum}H{p1p2p3}<>

Tip Offset Record

System sends this record in response to an *N{stationNum}*<> command.

It outputs 3 ASCII floats that were last set with the *N* command. Each float is represented as 7 characters with 3 digits after the floating point. Units are centimeters.

2{stationNum}N{Ox Oy Oz}<>

Bytes	Explanation
1	Record type, 2.
2	Station Number. A hexadecimal number up to C.
3	Sub-record type, N.
4 – 11	*Ox* – X-direction tip offset.
12 – 19	*Oy* – Y-direction tip offset.
20 – 27	*Oz* – Z-direction tip offset.
28 – 29	CR, LF.

Position Operational Envelope Record

System sends this record in response to a *V{stationNum}*<> command. It outputs six ASCII floats that were last set with the *V* command. Each float is represented as sseven characters with two digits after the floating point. Units are meters.

2{stationNum}V{Xmax,Ymax,Zmax,Xmin,Ymin,Zmin}<>

Bytes	Explanation
1	Record type, 2.
2	Station Number. A hexadecimal number up to C.
3	Sub-record type, V.
4 – 11	*Xmax* – Maximum X-direction value.
12 – 19	*Ymax* – Maximum Y-direction value.
20 – 27	*Zmax* – Maximum Z-direction value.
28 – 35	*Xmin* – Minimum X-direction value.
36 – 43	*Ymin* – Minimum Y-direction value.
44 – 51	*Zmin* – Minimum Z-direction value.
52 – 53	CR, LF

2.10.4. InterSense-Specific Records

Manufacturer System Status Record

System sends this record in response to an *MS*<> command. It outputs a status record specific to the InterSense system.

31S{statusRecord}<>

Bytes	Explanation
1	Record type, '3'.
2	Station Number, Always '1'.
3	Sub-record type, 'S'.
4	Config Hex Char 0.
5	Config Hex Char 1.
6	Config Hex Char 2.
7, 8	CR, LF.

To decode each of the Config Hex Characters

Config Hex Char 0 – Reserved
Config Hex Char 1 – Reserved
Config Hex Char 2 – See table below for possible bit settings.

Bit	Meaning
0	Reserved.
1	Boresight Compatibility Mode (0 = Firmware Version 2.x, 1 = FASTRACK™).
2	Time units (0 = *milli*seconds, 1 = *micro*seconds).
3	ReceiverPod LEDs (0 = OFF, 1 = ON)

Manufacturer Station Record

System sends this record in response to the *Ms{stationNum}*<> command. It outputs a station status record specific to the InterSense system.

3{stationNum}s{statusRecord}<>

Bytes	Explanation
1	Record type, '3'.
2	Station Number, a hexadecimal number up to C.
3	Sub-record type, 'S'.
4	Config Byte 0.
5	Config Byte 1.
6	Config byte 2.
7, 8	CR, LF.

To decode each of the Config bytes

Config Byte 0 – Unused
Config Byte 1 – Corresponds to the Perceptual Enhancement Level.

Can be 0, 1, or 2. The remaining 2 bits reserved for future expansion of this option.

Config Byte 2 – See table below for possible bit settings.

Bit	Meaning
0	Reserved.
1	Heading Compensation Mode bit 1.
2	Heading Compensation Mode bit 2.
3	Reserved.

Heading compensation bits are translated as:

00 – Compass mode 0 (compass is off)
01 – Compass mode 1 (Partial compass mode)
10 – Compass mode 2 (Full compass mode)

Prediction Interval Record

System sends this record in response to *Mp{stationNum}*<> command.

It returns an ASCII integer for the number of milliseconds of prediction.

3{stationNum}p{interval}<>

Bytes	Explanation
1	Reserved.
2	Station Number. A hexadecimal number up to C.
3	Sub-Record type, 'p'
4 – 5	Prediction interval.
6, 7	CR, LF.

Sensitivity Level Record

System sends this record in response to *MQ{station number}*<> command.

It returns the current sensitivity settings of a station. Settings are only relevant when Perceptual Enhancement level is set to 1 or 2.

3*{Station Number}Q{Sensitivity Level}*<>

Bytes	Explanation
1	Record type , '3'.
2	Station Number. A hexadecimal number up to C.
3	Sub-Record type, ‘Q'.
4	Sensitivity level 1 to 5
5, 6	CR, LF.

Genlock Synchronization Record

System sends this record in response to *MG*<> command. It returns the current synchronization settings of the system.

31G{State}, {Rate}, {Number of cycles per signal}<>

Bytes	Explanation
1	Record type , '3'.
2	Always 1.
3	Sub-Record type, ‘G'.
4	State, 0 to 3.
5 – 11	Strobe rate or Output record rate.
12 – 14	Number of computational cycles tracker performs between sync signals or output records. Multiply this parameter by Rate to determine the internal update rate of the tracker.
15, 16	CR, LF.

2.10.5. Records Specific to IS-900 Models

Ultrasonic Timeout Record

System sends this record in response to the *MU*<> command. It returns an ASCII integer for the number of milliseconds of the ultrasonic timeout.

31U{Interval}<>

Bytes	Explanation
1	Record type , '3'.
2	Station Number. Always 1.
3	Sub-Record type, ‘U''.
4, 5	Ultrasonic timeout value.
6, 7	CR, LF.

Ultrasonic Sensitivity Record

System sends this record in response to the *Mg*<> command. It returns an ASCII integer representing current sensitivity setting of the ultrasonic receivers.

31g{Level }<>

Bytes	Explanation
1	Record type , '3'.
2	Station Number. Always 1.
3	Sub-Record type, ‘g''.
4	Sensitivity Level.
5, 6	CR, LF.

Fixed PSE Record

System sends this record in response to *MCF[Fixed PSE number]*<> command. It returns the current settings of a single PSE or entire constellation.

31F{Fixed PSE number}{Fixed PSE record}<>

Bytes	Explanation
1	Record type, 3.
2	Constellation Number. Always 1.
3	Sub-Record type, M.
4 – 10	Fixed PSE number in ASCII decimal format.
11 – 40	x, y, z components of position vector in *10.4f* ASCII format. Values are in meters.
41 – 61	x, y, z components of normal vector in *7.2f* ASCII format.
62 – 68	Hardware ID code in ASCII format.
69, 70	CR, LF.

Tracking Status Record

System sends this record in response to *MP*<> command.

It returns the tracking status information for all 12 stations. Range measurement is defined as an ultrasonic signal received by a single URM. For example, if the system is configured with two URMs and six ultrasonic beacons (SoniDiscs), 12 range measurements per cycle could be received.

31P{Tracking state record}<>

Bytes	Explanation
1	Record type, 3.
2	Station Number. Always 1.
3	Sub-Record type, P.
4	Tracking state identifier for station 1. State can be: **L** for lost, **T** for tracking, or **X** if the station is invalid or the wireless link is down. **0** means that the station is connected or the station state is off.
5	Number of range measurements received this cycle.
6	Number of range measurements rejected.
3+3*N	Tracking state identifier for station N.
40 – 44	Update rate per station.
45	Not used, always blank.
46	Genlock identifier. Can be G for Genlock on and synchronized, X if Genlock on but not synchronized, or blank if Genlock is off.
47, 48	CR, LF.

3. IS-900 UDP Packet Formats

For firmware version 4.20 or higher

This section details the format of UDP packets that can be transmitted over the Ethernet interface of the processor.

3.1. UDP Station Packet

The fields for this type of data are in each single packet received over a UDP connection. Note that **`float`** values (**`Orientation`**, **`Position`**, and **`TimeStamp`**) are little endian. In most situations you should be deploying the DLL to read this information, rather than parsing it directly. The DLL can parse all the UDP packets emitted from the IS-900/IServer. This data structure information is useful only if you are trying to track information from a device not supported by the InterSense DLL.

Byte	Field	Data Type	Description
1	**StartByte**	unsigned char	Always 0xFF.
2	**PacketType**	unsigned char	Types of the packet depend on the output type. Those applicable to IServer/IS-900 devices are: • **`UDP_STATION_DATA`** – Regular data output from the IS-900/IServer. • **`UPD_EXTENDED_STATION_DATA`** – Extended data format from IServer or from IS-900 if one of two Fastrak data items have been configured to appear in the output: • **`31`** – Gyro-angular velocity • **`32`** – Accelerometer data in the navigation frame
3	**PacketSeqNum**	unsigned char	Next sequential packet number available, from 0 to 254. After value 254, the sequencing starts over at 0.
4	**Checksum**	unsigned char	Simple checksum of all data bytes after the first four header bytes. Should result in a modulus 256.
5	**Model**	unsigned char	Tracker model as defined in the **insense.h** file. **`ISD_SYSTEM_MODEL`** (**`3`** for IS-900).
6	**StationNum**	unsigned char	Station number between 1 and 8 inclusive. For IS-900s, corresponds to the physical port each tracker is plugged into. For InertiaCubes using IServer, the value is sequential.
7	**TrackingStatus**	unsigned char	For IS-900, tracking status byte in the value range of **`0`** to **`255`**, where **`0`** is lost and higher values are better. The tracking status depends on number of errors, number of measurements, and other related factors.
8	**ButtonState**	unsigned char	An 8-bit value indicating the state of buttons for trackers that have them. **`0`** is Not Pressed, **`1`** is Pressed.
9–16	**AnalogData[8]**	unsigned char	Array of values that refer to the data source. 0 and 1 are X and Y joystick axes where the device is so equipped, primarily with Wand trackers, 2 to 5 are analog AUX data that are not typically used, and 6 and 7 are always **`0`**.

17–28	**Orientation[3]**	float	Euler angles of yaw, pitch, and roll, in degress.
29–40	**Position[3]**	float	Always in meters, regardless of IS-900 unit settings.
41–44	**TimeStamp**	float	Time stamp of the data in seconds, reported only if requested.

3.2. UDP Station Extended Packet

Extended data packet that is similar to **`udpStationPacket`**. You receive this extended packet instead of the standard packet when you select output record list item 31 or 32. This data is in a single data packet received over a UDP connection and contains extended information that is richer in detail than the information in the **`udpStationPacket`** packet. For both packet types the first 7 bytes are identical, but after that this data packet contains significantly more advanced information. Note that **`float`** values (**`Orientation`**, **`Position`**, and **`TimeStamp`**) are little endian. In most situations you should be deploying the DLL to read this information, rather than parsing it directly. The DLL can parse all the UDP packets emitted from the IS-900/IServer. This data structure information is useful only if you are trying to track information from a device not supported by the InterSense DLL.

Byte	Field	Data Type	Description
1	**StartByte**	char	Always **0xFF**.
2	**PacketType**	char	Types of the packet depend on the output type. Those applicable to IServer/IS-900 devices are: • **`UDP_STATION_DATA`** – Regular data output from the IS-900/IServer. • **`UPD_EXTENDED_STATION_DATA`** – Extended data format from IServer or from IS-900 if one of two Fastrak data items have been configured to appear in the output: • **`31`** – Gyro-angular velocity • **`32`** – Accelerometer data in the navigation frame
3	**PacketSeqNum**	char	Next sequential packet number available, from 0 to 254. After value 254, the sequencing starts over at 0.
4	**Checksum**	char	Simple checksum of all data bytes after the first four header bytes. Should result in a modulus 256.
5	**Model**	char	Tracker model as defined in the **insense.h** file. **`ISD_SYSTEM_MODEL`** (**`3`** for IS-900).
6	**StationNum**	char	Station number between 1 and 8 inclusive. For IS-900s, corresponds to the physical port each tracker is plugged into. For InertiaCubes using IServer, the value is sequential.
7	**TrackingStatus**	char	For IS-900, tracking status byte in the value range of **`0`** to **`255`**, where **`0`** is lost and higher values are better. The tracking status depends on number of errors, number of measurements, and other related factors.
8	**CommIntegrity**	char	Communication integrity of wireless link with a value from **`0`** to **`100`**; should normally be **`100`** or very close for wired devices and in the mid-**`90`**s or better for wireless devices.

9	**DeviceType**	char	Type of device. Set to a value of **DEVICE_STATION_TYPE**.
10	**TrackingQuality**	char	Currently unused.
11	**Flags**	char	The type of data items collected: • bitwise OR of 0x01 = Angular velocity • 0x02 = Nav-frame velocity • 0x04 = Accelerometer data
12	**ButtonState**	char	An 8-bit value indicating the state of buttons for trackers that have them. **0** is Not Pressed, **1** is Pressed.
13–20	**AnalogData[8]**	char	Array of values that refer to the data source. 0 and 1 are x and y joystick axes where the device is so equipped, primarily with Wand trackers. 2 to 5 are analog AUX data that are not typically used. 6 and 7 are always **0**.
21–28	**AuxInputs[8]**	char	Currently unused.
29–40	**Orientation[3]**	float	Euler angles of yaw, pitch, and roll, in degress.
41–52	**Position[3]**	float	Always in meters, regardless of IS-900 unit settings.
53–64	**AngVel[3]**	float	Number of rad/sec, in sensor body coordinate frame.
65–76	**VelNav[3]**	float	Velocity in number of meter/sec, if available.
77–88	**AccelNav[3]**	float	Acceleration in meter^2/sec, in the navigation (earth) coordinate frame.
89–100	**AccelBody[3]**	float	Acceleration in meter^2/sec, in sensor body coordinate frame.
100–103	**TimeStamp**	float	Timestamp of the data in seconds, reported only if requested.
104–107	**StillTime**	float	Number of seconds the tracking device has been motionless, up to 10 sec. Reported for InertiaCube/PCTracker only.
108**–**119	**GyroRate[3]**	float	Raw gyro output with only factory calibration applied.
120**–**131	**AngVelNav[3]**	float	Angular velocity in Nav frame.
132**–**143	**MagRaw[3]**	float	Reported for InertiaCube only; array of raw magnetometer readings (approximately in Gauss unless fixed metal calibration applied; then normalized) that are used to calculate MagYaw.
144–147	**MagYaw**	float	Reported for InertiaCube only; raw Magnetometer heading, computed based on current orientation.

4. Quick References

4.1. IS-900 Interface Protocol Commands

For a more detailed description of Interface Protocol Commands, see Section 0.

NOTE: "<>" represents a newline character, and "^" represents a control character

Command	Syntax	
Data Record Request	*P*	
Output mode	*C*	Set continuous output mode
	c	Set polled output mode
Alignment Reference Frame	*A{stationNum},[Ox,Oy,Oz,Xx,Xy,Xz,Yx,Yy,Yz]<>*	
Reset Alignment Reference Frame	*R{stationNum}<>*	
Boresight Reference Angles	*G{stationNum},[yawref, pitchref, rollref]<>*	
Boresight Compatibility Mode	*MBF<>*	*Switch to Fastrak™ Compatible mode.*
	MBI<>	*Switch to Version 2.x Compatible mode (default).*
Boresight	*B{stationNum}<>*	*(Fastrak™ compatibility mode)*
	MB{stationNum}<>	*(Version 2.x compatibility mode)*
Unboresight	*b{stationNum}<>*	*(Fastrak™ compatibility mode)*
	Mb{stationNum}<>	*(Version 2.x compatibility mode)*
Heading Boresight	*B{stationNum}<>*	*(Version 2.x compatibility mode)*
	MB{stationNum}<>	*(Fastrak™ compatibility mode)*
Heading Unboresight	*b{stationNum}<>*	*(Version 2.x compatibility mode)*
	Mb{stationNum}<>	*(Fastrak™ compatibility mode)*
Set Serial Communication Parameters	*o{rate,parity,bits,HHS}<>*	
System Record Request	*S*	
Station Status	*l{stationNum},[state]<>*	
Output Units Control	U	Set units to inches.
	u	Set units to centimeters.
System control	^K	Save current settings to non-volatile memory.
	W	Restore factory default settings.
	^Y	Restart the firmware to the power up condition.
	^S	Suspend data transmission.
	^Q	Resume data transmission.
Output record mode	*F*	*Put in ASCII output mode.*
	f	*Put in Binary output mode.*
Output record list settings	*O{stationNum},[p1],[p2],[p3],.....,[pn]<>*	
Define Tip Offsets	*N{stationNum},[Ox, Oy, Oz]<>*	
Position Operational Envelope	*V{stationNum},[Xmax,Ymax,Zmax,Xmin,Ymin,Zmin]<>*	
Hemisphere	*H{stationNum},[p1,p2,p3]<>*	
Time Units	*MT<>*	Sets to milliseconds.
	Mt<>	Sets to microseconds.
Set Current Time to Zero	*MZ<>*	

Ethernet Communication Parameters	*MEthIp{address}<>*	*Set IP address*
	MEthTcpPort{port}<>	Set TCP port
	MEthUdp{state}<>	Set UDP state
	MEthUdpPort{port}<>	Set UDP port
	MEthSubnet{subnet}<>	Set subnet
	MEthMode{mode}<>	Set manual or DHCP mode
	MEthApply<>	Apply Ethernet settings
InterSense System Status Record Request	*MS<>*	
Tracking Status Record Request	*MP<>*	
Genlock Synchronization	*MG[State, Rate]<>*	
Genlock Phase	*MGP[Param]<>*	
Configuration Lock	*MConfigLockMode{Mode}<>*	
SoniStrip LED Control	*ML[state]*	
Error reporting	*ME<>*	
	MEC<>	
	ME1<>	
	ME0<>	
Command Logging	*MLogOpen<>*	
	MLogClose<>	
	MLogClear<>	
	MLogState<>	
	MLogSend<>	
InterSense Station Status Record Request	*Ms{stationNum}<>*	
Prediction Interval	*Mp{stationNum},[Interval]<>*	
Perceptual Enhancement Level	*MF{stationNum},{Mode}<>*	
Compass Heading Correction	*MH{stationNum},{state}<>*	
Rotational Sensitivity Level	*MQ{stationNum},[Sensitivity Level]<>*	
Associate Fixed PSE with a Constellation	*MCF{FPSE number}, {xp, yp, zp, xn, yn, zn, IDcode}<>*	
Disassociate Fixed PSE from Constellation	*MCf[Fixed PSE number, IDcode]<>*	
Clear All Fixed PSEs (Constellation)	*MCC<>*	
Apply New Configuration	*MCe<>*	
Cancel Configuration Session	*MCx<>*	

5. Thales Visionix, Inc. End User License Agreement

Complete text of the End User License Agreement you agree to with purchase of this software is provided below:

5.1. Thales Visionix, Inc. End User License Agreement

This License Agreement (“Agreement”) is an agreement between you (“Licensee”) and Thales Visionix, Inc. (“Licensor”). It governs your use of the software supplied to you by Thales Visionix, Inc. ("the Software") and related documentation. By downloading, installing or otherwise using the Software, you agree to be legally bound by the terms of this Agreement. Please read the entire contents of this agreement. THIS SOFTWARE IS COPYRIGHT © 2016 by Thales Visionix, Inc. ALL RIGHTS RESERVED WORLD WIDE.

IMPORTANT NOTE

You should make a backup of any important information, valuable data, or Software on your PC before installing this or any other software. Also, you should make regular backups of the Software and any data files, so that in the event of data loss, your information may be restored.

1. DEFINITION

(a) “Documentation” means the standard end user manual for the Product provided by Licensor.
(b) “Product” means all Licensor equipment including, but not limited to, the InertiaCube, IS-900, IS-1200+ and IS-1500 product families.
(c) “Software” means all Licensor software code including sample source code, compiled code, and embedded firmware.

2. LICENSE

(a) Grant of Rights. Licensor hereby grants to Licensee a non-exclusive, royalty-free, perpetual license to use the Software solely with Licensor Product and solely for the operation of Product in accordance with its documentation. Licensee may redistribute Licensor libraries, isense.dll, libisense.so, and libisense.dylib as required for the operation of Licensee products.

(b) Restrictions. The Software (and any copy thereof) is licensed not sold, and Licensee receives no title to or ownership of the Software and no rights other than those specifically granted in Section 2(a) above. With the exception of Licensor sample source code, Licensee may not modify, reproduce, and create derivative works of the Software and shall not attempt to decompile or otherwise reverse engineer the Software. Licensee shall not remove any proprietary or copyright notices of Licensor in the Software or any copies thereof.

3. WARRANTY

This software and accompanying written materials (including instructions for use) are provided "as is" without warranty of any kind. Further, Licensor does not warrant, guarantee, or make any representations regarding the use, or the results of use, of the software or written materials in terms of correctness, accuracy, reliability, currentness, or otherwise. The entire risk as to the results and performance of the software is assumed by Licensee. If the software or written materials are defective, Licensee, and not Licensor or its dealers, distributors, agents, or employees, assume the entire cost of all necessary servicing, repair, or correction.

The above is the only warranty of any kind, either express or implied, including but not limited to the implied warranties of merchantability and fitness for a particular purpose, which is made by licensor on this product. No oral or written information or advice given by licensor, its dealers, distributors, agents or employees shall create a warranty or in any way increase the scope of this warranty and you may not rely on any such information or advice.

4. LIMITATION OF LIABILITY

Neither licensor nor anyone else who has been involved in the creation, production or delivery of this product shall be liable for any direct, indirect, consequential or incidental damages (including damages for loss of business profits, assets, business interruption, loss of business information, and the like) arising out of the use or inability to use such product even if licensor has been advised of the possibility of such damages.

5. TERMINATION

(a) Termination. This Agreement shall terminate automatically and immediately upon any breach of this agreement by Licensee, including but not limited to any action in violation of the license rights and restrictions set forth in Section 2.

(b) Effects of Termination. Upon termination of this Agreement, the licenses granted in Section 2(a) will terminate and Licensee will cease all use of the Product and delete all copies of any Software (including Upgrades and Updates stored on Licensee computers) in its possession or control. This Section 5(b) shall survive termination of this Agreement.

6. MISCELLANEOUS

(a) Notices. Notices to Licensor pursuant to this Agreement will be sent to 22605 Gateway Center Drive, Clarksburg, Maryland 20972, Attention: Contracts Department. Such notices will be deemed received at such addresses upon the earlier of (i) actual receipt or (ii) delivery in person, or by certified mail return receipt requested.

(b) U.S. Government Use. The Software is a “commercial item” as that term is defined at 48 C.F.R. 2.101, consisting of “commercial computer software” and “commercial computer software documentation” as such terms are used in 48 C.F.R. 12.212.

(c) Export. Licensee understands and recognizes that the Product is export-controlled and agrees to comply with all applicable export regulations. The Product may not be exported to any country outside the country where the Product was acquired without Licensor’s prior written consent.

(d) Assignment & Successors. Licensee may not assign this Agreement or any of its rights or obligations hereunder.

(e) Choice of Law & Jurisdiction. This Agreement will be governed by the laws of the United States of America.

(f) Severability. To the extent permitted by applicable law, the parties hereby waive any provision of law that would render any clause of this Agreement invalid or otherwise unenforceable in any respect. In the event that a provision of this Agreement is held to be invalid or otherwise unenforceable, such provision will be interpreted to fulfill its intended purpose to the maximum extent permitted by applicable law, and the remaining provisions of this Agreement will continue in full force and effect.

(g) Entire Agreement. This Agreement sets forth the entire agreement of the parties and may not be modified except by a written agreement signed by Licensor.