Developer Instructions for 4.237 SDK

2012-05-08

Contents

Introduction

This document describes the interface to be used by the application software to initialize and retrieve data from the 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, in the sense that software written for older versions of the DLL should generally run without recompilation using the current version.

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 can also be obtained from 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. These files should be placed 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.

Basic Usage

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 3 or 4 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 );
}
```

API Reference

ISD_OpenTracker()

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

This function is used for opening a single tracker. It may be called multiple times in order to open multiple trackers, though typically using `ISD_OpenAllTrackers()` is recommended instead for use with multiple trackers.

`hParent`	Handle to the parent window. This parameter is optional and should only be used if information screen or tracker configuration tools are to be used when available in the future releases. All included sample programs pass `NULL`.
`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`	This feature has not been implemented. Its purpose is to display an information window to show the tracker detection progress and results. Currently DLL writes only to Windows console. Most applications should pass `FALSE`.
`verbose`	Pass `TRUE` if you would like a more detailed report of the DLL activity. Messages are printed to Windows console.

ISD_OpenAllTrackers()

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

This function is used for opening multiple trackers. It returns an array of handles for all detected trackers. -1 is returned on failure.

`hParent`	Handle to the parent window. This parameter is optional and should only be used if information screen or tracker configuration tools are to be used when available in the future releases. All included sample programs pass `NULL`.
`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`	This feature has not been implemented. Its purpose is to display an information window to show the tracker detection progress and results. Currently DLL writes only to Windows console. Most applications should 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 0 is passed, all currently open trackers are closed. When the last tracker is closed, the program frees the DLL. Returns FALSE if failed for any reason.

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

`Tracker` Pointer to a structure of type **`ISD_TRACKER_INFO_TYPE`**. The structure definition is given below.

ISD_SetTrackerConfig()

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

When used with IS Precision Series (IS-300, IS-600, and IS-900) tracking devices this function call will set ultrasonic and synchronization parameters, all other fields in the `ISD_TRACKER_INFO_TYPE` structure are for information purposes only.

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

`Tracker` Pointer to a structure of type `ISD_TRACKER_INFO_TYPE`. The structure definition is given below.

ISD_GetCommInfo()

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

Get `RecordsPerSec` and `KBitsPerSec` without requesting Genlock and other settings from the tracker. Use this instead of **`ISD_GetTrackerConfig()`** 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 is a handle returned by **`ISD_OpenTracker()`** or **`ISD_OpenAllTrackers()`**.
`Tracker`	Pointer to a structure of type `ISD_TRACKER_INFO_TYPE`. The structure definition is given below.

ISD_SetStationConfig()

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

Configure station as specified in the `ISD_STATION_INFO_TYPE` structure. Before this function is called, all elements of the structure must be assigned 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.
This function is ignored if used with InterTrax products. IS-900, IS-600, IS-300 and InertiaCubes allow many parameters (including but not limited to `AngleFormat`, `Compass`, `Prediction`, `Enhancement`, and `Sensitivity`) to be changed.

`handle`	Handle to the tracking device. This is a handle returned by **`ISD_OpenTracker()`** or **`ISD_OpenAllTrackers()`**.
`Station`	Pointer to a structure of type `ISD_STATION_INFO_TYPE`. The structure definition is given below.
`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 is a handle returned by **`ISD_OpenTracker()`** or **`ISD_OpenAllTrackers()`**.
`Station`	Pointer to a structure of type `ISD_STATION_INFO_TYPE`. The structure definition is given below.
`stationID`	Number from 1 to `ISD_MAX_STATIONS`.

ISD_ConfigureFromFile()

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

When a tracker is first opened, library automatically looks for a configuration file in current directory of the application. File name convention is isenseX.ini where X is a number, starting at 1, identifying the first tracking system in the order of initialization. This function provides for a way to manually configure the tracker using an arbitrary configuration file instead.

`handle` Handle to the tracking device. This is a handle returned 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 )
```

Save tracker configuration. For devices with on-host processing, like the IS-900 PCTracker or PCI Tracker, this will write to the isenseX.cfg file. Serial port devices like IS-300, IS-600 and IS-900 save 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 is a handle returned 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. Data is places in the `ISD_TRACKING_DATA_TYPE` structure. `TimeStamp` is only available if requested by setting `TimeStamped` field to `TRUE`. Returns `FALSE` if failed for any reason.

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

`Data` Pointer to a structure of type `ISD_TRACKER_DATA_TYPE`. See below for structure definition. Orientation data order is Yaw, Pitch, and Roll for Euler angles and W, X, Y, Z for quaternions.

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 returns 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 return 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 return 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 is a handle returned 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 )
```

Activate the ring buffer. While active, all data samples are stored in the buffer. Because this is a ring buffer, it will only store 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 is a handle returned 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 )
```

Stop 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 is a handle returned 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 returns 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 is a handle returned 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.

handle	Handle to the tracking device. This is a handle returned 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 )
```

Boresight station using specific reference angles. This is useful when you need to apply a specific offset to system output. For example, if a sensor is mounted at 40 degrees relative to the 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 is a handle returned 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 )
```

Boresight, or unboresight a station. If 'set' is TRUE, all angles are reset to zero. Otherwise, all boresight settings are cleared, including those set by ISD_ResetHeading() and ISD_BoresightReferenced().

Note that the angles are reset relative to the current yaw; if the station is at 90 degrees yaw and 0 degrees pitch/roll when this function is called, rolling the sensor (relative to its current heading) will

be considered pitch, and pitch (relative to its current heading) will be considered roll; it does **not** perform a boresight 'relative' to the current orientation vector.

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

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. Commands in the script should be terminated by the newline character '\n'. The linefeed character '\r' is added by the function and is not required.

Note that this 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.

`handle`	Handle to the tracking device. This is a handle returned 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 the interface is configured to expect. If too many are specified, extra bytes are ignored.

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

ISD_NumOpenTrackers()

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

The number of currently opened trackers is stored in the parameter passed to this function.

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 is a handle returned 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,
                           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.

`handle`	Handle to the tracking device. This is a handle returned by **`ISD_OpenTracker()`** or **`ISD_OpenAllTrackers()`**.
`hwInfo`	An `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).

`handle`	Handle to the tracking device. This is a handle returned 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_GetPortWirelessInfo()

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

Retrieve wireless configuration information.

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

API Data Structures

ISD_TRACKER_INFO_TYPE

```
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 port the tracker is connected to. Starts with 1.

RecordsPerSec
Communication 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 4 values:

0 - OFF, system is in free run
1 – Not used
2 - ON, hardware genlock frequency is specified by the user
3 - ON, no hardware signal, lock 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

This data structure is used to get and set station configuration, 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    bReserved2;
    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 to be a tracking system consisting of one station, which cannot be turned off, so this field will always be `TRUE`. 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 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, the readings produced by the magnetometers inside the InertiaCube 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. Please use the new notation. This API will correctly interpret the settings.

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's 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. 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 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

This setting is only used when Perceptual Enhancement Level is set to 1 or 2. It controls the minimum angular rotation rate picked up by the InertiaCube. 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 maximum provided by this control, you must use Perceptual Enhancement level 0. For InterTrax this value is fixed to default and can't be changed.

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. Supported by IS-300, IS-600, IS-900 and InertiaCubes. 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 returned in degrees. Default is ISD_EULER.

TimeStamped

TRUE if time stamp is requested, default is FALSE.

GetInputs

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 Magnetic Environment Calibration is applied. 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 a weight to compass measurements, allowing it to reject bad measurements. Values from 0 to 3 are accepted. If CompassCompensation is set 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. If the sensor is used in an environment with significant magnetic interference this can result in drift due to insufficient compensation from the compass data. Default setting is 2.

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

ImuShockSuppression

This setting controls how the system deals with sharp changes in 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.

ImuShockSuppression

This setting controls the rejection threshold for ultrasonic measurements. Currently, it is implemented only for the IS-900 PCTracker. Default setting is 4, which results in measurements with range errors greater than 4 times the average to be rejected. Please do not change this setting without first consulting with InterSense technical support.

CoordFrame

Coordinate frame in which position and orientation data is reported. Can be ISD_DEFAULT_FRAME or ISD_VSET_FRAME. Second is used for camera tracker only. Default is ISD_DEFAULT_FRAME.

AccelSensitivity

AccelSensitivity is used for 3-DOF tracking with InertiaCube products only. It controls how fast tilt correction, using accelerometers, is applied. Valid values are 1 to 4, with 2 as default.

Level 1 reduces the amount of tilt correction during movement. While it will prevent any effect linear accelerations may have on pitch and roll, it will also reduce stability and dynamic accuracy. It should only be used in situations when 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 will reduce 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.

TipOffset

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

GetCameraData

TRUE to get computed FOV, aperture, etc. default is FALSE.

GetAuxInputs

TRUE to get values from auxiliary inputs connected to the I^2C port in the MicroTrax device.

ISD_TRACKING_DATA_TYPE

This data structure is used to return 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 {
    ISD_STATION_DATA_TYPE Station[ISD_MAX_STATIONS];
}
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;

    float   Reserved[56];
    float   MagBodyFrame[3];
}
ISD_STATION_DATA_TYPE;
```

TrackingStatus

Tracking status byte. Available only with IS-900 firmware versions 4.13 and higher, and isense.dll versions 3.54 and higher. It is a value from 0 to 255 that represents tracking quality. 0 represents lost.

NewData

`TRUE` if this is new data. Every time **`ISD_GetData()`** is called this flag is reset.

CommIntegrity

Communication integrity of wireless link (percentage of packets received from tracker, 0-100%)

BatteryState

Certain wireless devices only (0=N/A, 1=Low, 2=OK). Not currently used by MiniTrax or MicroTrax stations.

Euler

Orientation in Euler angle format (Yaw, Pitch, Roll)

Quaternion

Orientation in Quaternion format (W,X,Y,Z)

Position

Station position in meters.

TimeStamp

Timestamp in seconds, reported only if requested

StillTime

InertiaCube and PC-Tracker products only, whether sensor is still

BatteryLevel

Battery voltage, 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.

AnalogData

Only if requested. Current hardware is limited to 10 channels, only 2 are used. The only device using this is the IS-900 wand that has a built-in analog joystick. Channel 1 is x-axis rotation, channel 2 is y-axis rotation. Values are from 0 to 255, with 127 representing the center.

AuxInputs

Only if requested.

AngularVelBodyFrame

rad/sec, in sensor body coordinate frame. Reported as rates about X, Y and Z axes, corresponding to Roll, Pitch, Yaw order. This is the processed angular rate, with current biases removed. This is 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 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. This is the accelerometer measurements with calibration, current sensor orientation applied, and gravity subtracted. This is 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 due to temperature dependant gyro bias drift will remain.

MeasQuality

Ultrasonic Measurement Quality (IS-900 only, firmware >= 4.26)

TimeStampSeconds

Time Stamp in whole seconds.

TimeStampMicroSec

Fractional part of the Time Stamp in micro-seconds.

OSTimeStampSeconds
OSTimeStampMicroSec

Data record arrival time stamp based on OS time, reserved for future use, not implemented.

MagBodyFrame

3DOF sensors only. Magnetometer data along the X, Y, and Z axes. Units are nominally in Gauss, and factory calibration is applied. Note, however, that most sensors are not calibrated precisely since the absolute field strength is not necessary to for tracking purposes. Relative magnitudes should be accurate, however. Fixed metal compass calibration may rescale the values.

ISD_HARDWARE_INFO_TYPE

This data structure is used to return system hardware information using **`ISD_GetSystemHardwareInfo()`**. For more detailed descriptions of elements in the structure, please reference 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    bReserved2;
        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;
```

ISD_STATION_HARDWARE_INFO_TYPE

This data structure is used to return station (individual tracking device) hardware information using **`ISD_GetStationHardwareInfo()`**. For more detailed descriptions of elements in the structure, please reference 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    cReserved1[128];
    char    cReserved2[128];
    char    cReserved3[128];
    char    cReserved4[128];
}
ISD_STATION_HARDWARE_INFO_TYPE;
```

ISD_PORT_WIRELESS_INFO_TYPE

This data structure is used to information about the wireless hardware on a given port, using **`ISD_GetPortWirelessInfo()`**. The `radioVersion` field can be used to check the type of radio hardware:

```
2.4 GHz (Aerocomm, radio used with older MiniTrax trackers): 15 or 31
2.4 GHz (Chipcon, MicroTrax only): 128
900 MHz (MicroTrax only): 144
868 MHz (MicroTrax only): 160
```

```
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;
```