

Track Processor Library

(TPL)

for Retail Phase II

API and Integration Guide

Ver 1.0

Table of Contents

Contents

[1. Purpose 4](#_Toc467513669)

[2. Abbreviations 5](#_Toc467513670)

[3. Document History 7](#_Toc467513671)

[4. TPL API 8](#_Toc467513672)

[4.1. General description of TPL API 8](#_Toc467513673)

[4.2. TPL API Datga Structures 8](#_Toc467513674)

[4.3. API Classes and their Methods 8](#_Toc467513675)

[5. Integration Example of TPL 11](#_Toc467513676)

[5.1. Setting up the TPL 11](#_Toc467513677)

[5.2. Source code of integration examples 11](#_Toc467513678)

[Console application 11](#_Toc467513679)

[6. Reference 13](#_Toc467513680)

1. Purpose

This document is the integration guide for the Track Processor Library (TPL) for the Retail Phase II project. It contains API description and the example for integration of TPL into other system.

1. Abbreviations

The following abbreviations are used in this document:

|  |  |
| --- | --- |
| BLE | Bluetooth Low Energy |
| FP | Fingerprint |
| IRL | InvenSense Retail Library |
| MFP | Magnetic fingerprint |
| FPBL | Finger Print Builder Library |

1. Document History

|  |  |  |
| --- | --- | --- |
| **Date** | **Version** | **Comment** |
| November 17, 2016 | 1.0 | Document created |
|  |  |  |

1. TPL API
   1. General description of TPL API

TPL API contains the common structures and classes for validation of input IRL data before using them in fingerprint generation. The API is realized in namespace TrackPreProcessing.

TPL implements the following functionality:

* Validating and correcting IRL data
* Printing validation report

Main class of TPL is TrackProcessor.

Additionally, TPL API utilizes data structures defined in FPBL, such as TpnOutput and Venue.

* 1. IRL data structure description

IRL data are provided TPL by IRL data structure TpnOutput. This structure is defined in header file TpnData.hpp of FPBL.

Description of TpnOutput structure fields and correlation with IRL data entities are given in tables below.

Table: TpnOutput structure description

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Parameter name in FPBL** | **Description** | **Parameter name in IRL data** |
| 1 | timestamp | Time tag, sec | Entity: 0x00EA  Field: timetag\_ |
| 2 | position | User position information | See below |
| 3 | attitude | Device attitude information | See below |
| 4 | pdr | Additional information from PDR | See below |
| 5 | mag\_meas | Mag data | See below |

Table: position structure-member description

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Parameter name in FPBL** | **Description** | **Parameter name in IRL data** |
| 2.1 | position.lattitude | Geodetic latitude, deg | Entity: 0x0016  Field: latitude\_ |
| 2.2 | position.longitude | Geodetic longitude, deg | Entity: 0x0016  Field: longitude\_ |
| 2.3 | position.user\_heading | User heading, deg | Entity: 0x00E9  Field: heading\_ |
| 2.4 | position.sigma\_north | Position standard deviation in North direction, m | Entity: 0x00BD  Field: position\_north\_standard\_deviation\_ |
| 2.5 | position.sigma\_east | Position standard deviation in East direction, m | Entity: 0x00BD  Field: position\_east\_standard\_deviation\_ |
| 2.6 | position.sigma\_user\_heading | User heading uncertainty, deg | Entity: 0x0060  Field: heading\_standard\_deviation\_ |
| 2.7 | position.misalignment | Reserved | n/a |
| 2.8 | position.sigma\_misalignment | Reserved | n/a |
| 2.9 | position.floor | Floor number | Entity: 0x00DD  Field: floor\_number |
| 2.10 | position.altitude | Altitude, m | Entity: 0x0016  Field: height\_ |
| 2.11 | position.sigma\_altitude | Altitude standard deviation, m | Entity: 0x00BD  Field: height\_standard\_deviation\_ |
| 2.12 | position.navigation\_phase | Navigation phase flag | Entity: 0x00E7  Field: navigation\_phase\_ |
| 2.13 | position.fidgeting\_flag | Fidgeting flag | Entity: 0x008C  Field: walking\_fidgeting\_flag\_ |
| 2.14 | position.is\_valid | position structure data validity |  |

Table: attitude structure-member description

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Parameter name in FPBL** | **Description** | **Parameter name in IRL data** |
| 3.1 | attitude.orientation\_id | Orientation id flag (orientation based on pitch) | Entity: 0x008C  Field: orientation\_based\_on\_pitch\_ |
| 3.2 | attitude.roll | Device roll, deg | Entity: 0x008C  Field: attitude\_filter\_roll\_ |
| 3.3 | attitude.pitch | Device pitch, deg | Entity: 0x008C  Field: attitude\_filter\_pitch\_ |
| 3.4 | attitude.heading | Device heading, deg | Entity: 0x008C  Field: attitude\_filter\_heading\_ |
| 3.5 | attitude.sigma\_roll | Device roll standard deviation, deg | Entity: 0x008C  Field: attitude\_filter\_roll\_standard\_deviation\_ |
| 3.6 | attitude.sigma\_pitch | Device pitch standard deviation, deg | Entity: 0x008C  Field: attitude\_filter\_pitch\_standard\_deviation\_ |
| 3.7 | attitude.sigma\_heading | Device heading standard deviation, deg | Entity: 0x008C  Field: attitude\_filter\_heading\_standard\_deviation\_ |
| 3.8 | attitude.is\_valid | attitude structure data validity |  |

Table: pdr structure-member description

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Parameter name in FPBL** | **Description** | **Parameter name in IRL data** |
| 4.1 | pdr.stride\_length | Current stride length, m | Entity: 0x005E  Field: stride\_distance\_ |
| 4.2 | pder.is\_valid | mag\_meas structure data validity |  |

Table: mag\_meas structure-member description

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Parameter name in FPBL** | **Description** | **Parameter name in IRL data** |
| 5.1 | mag\_meas.mX | Calibrated x-axis magnetometer measurement, mG | Entity: 0x0062  Field: calibrated\_data\_x\_ |
| 5.2 | mag\_meas.mY | Calibrated y-axis magnetometer measurement, mG | Entity: 0x0062  Field: calibrated\_data\_y\_ |
| 5.3 | mag\_meas.mZ | Calibrated z-axis magnetometer measurement, mG | Entity: 0x0062  Field: calibrated\_data\_z\_ |
| 5.4 | mag\_meas.sigma\_mX | x-axis magnetometer measurement noise standard deviation, mG | Obtained from mag sensor specification  Default:10 mG |
| 5.5 | mag\_meas.sigma\_mY | y-axis magnetometer measurement noise standard deviation, mG | Obtained from mag sensor specification  Default:10 mG |
| 5.6 | mag\_meas.sigma\_mZ | z-axis magnetometer measurement noise standard deviation, mG | Obtained from mag sensor specification  Default:10 mG |
| 5.7 | mag\_meas.covarianceMatrix [3][3] | Mag bias error covariance matrix, mG^2 | Currently obtained from mag\_out.txt file  Columns 5-13 |
| 5.8 | mag\_meas.level\_of\_calibration | Accuracy class of mag calibration estimation | Currently obtained from mag\_out.txt file  Column 1 |
| 5.9 | mag\_meas.is\_valid | mag\_meas structure data validity |  |

* 1. API Classes and their Methods

In this section, the classes and their methods are described.

These classes are in TrackPreProcessing namespace.

**TrackProcessor.hpp**

TrackProcessor class and its methods:

/// Track processor class

class TrackProcessor

{

public:

TrackProcessor();

~TrackProcessor();

// The method adds instance of validator object to validator list

void AddValidatorInstance(Validator\_Base \* validator);

// The method clears validator list

void ClearValidatorList();

// The method generates dataset name from specified irl\_file\_name

std::string GenerateDataSetName( const std::string irl\_file\_name ) const;

// The method set dataset name

void set\_data\_set\_name( const std::string data\_set\_name );

// The method validates specified irl\_data for specified venue

bool ValidateDataSet( const std::string dataset\_name,

std::vector<TpnOutput> &irl\_data,

const Venue venue);

// The method returns integral validity of last processed data set.

bool GetDataSetValidity();

// Console logging

void echo( const std::string echo\_string );

// The method enable/disable console logging

void set\_echo\_on( const bool echo\_on );

// The method prints results of last validation to specified stream

void PrintValidationReport( std::ofstream &fout );

// The prints header of validators to specified stream

void PrintValidationReportHeader( std::ofstream &fout);

// The method of clears results of last validation

void ClearValidationReport();

};

**MagCalibrators.hpp**

struct MagBias

{

double bias[3]; ///< mag bias error covariance matrix [mT^2], column order

double covarianceMatrix[3][3]; ///< mag bias error covariance matrix [mT^2], column order

uint8\_t calibration\_level; ///< calibration level

MagBias() : calibration\_level(0) { };

};

Calibrator\_FDMC class and its methods:

class Calibrator\_FDMC

{

public:

Calibrator\_FDMC(){};

MagBias CalculateBias(const std::vector<TpnOutput> &irl\_data); ///< return magnetic biases for IRL data

};

1. Integration Example of TPL
   1. Setting up the TPL

In order to set up the TPL Fpbl::GridBuilder, Fpbl::MagneticGrid, Fpbl::WiFiGrid, Fpbl::BleGrid objects must be created.

In order to set up the TPL a TrackPreProcessing::TrackProcessor object must be created first:

TrackPreProcessing::TrackProcessor track\_processor;

All actual validators are created in the constructor and placed into validation list of track\_processor class.

Validation log header can be printed optionally as follows:

track\_processor.PrintValidationReportHeader(validation\_log);

After parsing each IRL data file a vector of TpnOutput is filled with data:

std::vector<TpnOutput> irl\_data;

Items of this vector can be sent to grid builder, but before it can be calculated the magnetic biases by TrackPreProcessing::Calibrator\_FDMC

TrackPreProcessing::Calibrator\_FDMC fdmc;

TrackPreProcessing::MagBias magbias = fdmc.CalculateBias(irl\_data);

and it can be validated by ValidateDataSet method of TPL library:

std::string data\_set\_name = track\_processor.GenerateDataSetName( path\_to\_irl\_file );

bool track\_is\_valid = track\_processor.ValidateDataSet(data\_set\_name, irl\_data, venue);

Currently each IRL data file musty be loaded and validated separately.

Note that items of irl\_data vector are modified during the validation. So it is necessary to utilize the items of processed irl\_data only in fingerprint building.

Optional validation log can be printed after that:

track\_processor.PrintValidationReport( validation\_log );

track\_processor.ClearValidationReport();

* 1. Source code of integration examples

### Console application

Good example of usage of Track Processor Library together with Fingerprint Builder Library is PC console application FP builder. The application parses IRL data from an input folder, then validates and corrects the data using TPL functions. After that it feeds the FPBL with position/attitude, IRL, magnetic, WiFi, BLE data. In return, the fingerprint builder library generates fingerprint databases.

The source code of the PC console application is available at Gift repository on GitHub by the following link: [Gift](https://github.com/InvenSenseInc/Gift)/[Applications](https://github.com/InvenSenseInc/Gift/tree/master/Applications)/fp\_builder.console/.

1. Reference
2. Fingerprint Builder Library (FPBL) for Retail Phase II. Algorithmic Design Document