

Fingerprint Builder Library

(FPBL)

for Retail Phase II

API and Integration Guide

Ver 1.1

Table of Contents

Contents

[1. Purpose 5](#_Toc460343648)

[2. Abbreviations 6](#_Toc460343649)

[3. Document History 8](#_Toc460343650)

[4. FPBL API 9](#_Toc460343651)

[4.1. General description of FPBL API 9](#_Toc460343652)

[4.2. API Data Structures 9](#_Toc460343653)

[4.2.1. Magnetic Data Structures 9](#_Toc460343654)

[4.2.2. WiFi/BLE Data Structures 10](#_Toc460343655)

[4.2.3. TPN (IRL) Data Structures 10](#_Toc460343656)

[4.2.4. Grid Data Structures 11](#_Toc460343657)

[4.3. API Classes and their Methods 13](#_Toc460343658)

[5. Integration Example of FPBL 19](#_Toc460343659)

[5.1. Setting up the FPBL 19](#_Toc460343660)

[5.2. Setting Venue Parameters 20](#_Toc460343661)

[5.3. Processing FPBL data 20](#_Toc460343662)

[5.4. Processing IRL data 20](#_Toc460343663)

[5.5. Processing WiFi/BLE data 21](#_Toc460343664)

[5.6. Alternative Interface 24](#_Toc460343665)

[5.6.1. Processing Position and Attitude data 24](#_Toc460343666)

[5.6.2. Processing Magnetic data 25](#_Toc460343667)

[5.7. Creating Fingerprint databases 26](#_Toc460343668)

[5.8. Source code of integration examples 26](#_Toc460343669)

[5.8.1. Console application 26](#_Toc460343670)

[6. Reference 27](#_Toc460343671)

1. Purpose

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

1. Abbreviations

The following abbreviations are used in this document:

|  |  |
| --- | --- |
| AP | Access point |
| BLE | Bluetooth Low Energy |
| FP | Fingerprint |
| IRL | InvenSense Retail Library |
| MFP | Magnetic fingerprint |

1. Document History

|  |  |  |
| --- | --- | --- |
| **Date** | **Version** | **Comment** |
| 07 July 2016 | 1.0 | Document created |
| 04 April 2017 | 1.1 | API Updated |

1. FPBL API
   1. General description of FPBL API

FPBL API contains the common structures for description input/output data and classes GridBuilder, MfpBuilder, WiFiBuilder, BleBuilder that realize FPBL algorithms. API is realized in namespace Fpbl.

FPBL API includes the following methods:

* Creation of FPBL
* Set Venue parameters
* Input positions with timestamps
* Input attitudes with timestamps
* Input magnetic data with timestamps
* Input WiFi data with timestamps
* Input BLE data with timestamps
* Creation of magnetic/WiFi/BLE grids
* Creation of magnetic fingerprint map
* Creation of WiFi fingerprint map
* Creation of BLE fingerprint map
  1. API Data Structures

In this item the common structures are described

* + 1. Magnetic Data Structures

/\*\* single magnetometer measurement \*/

struct MagneticVector

{

double mX; /\*\*< magnetic field on x-axis [uT] \*/

double mY; /\*\*< magnetic field on y-axis [uT] \*/

double mZ; /\*\*< magnetic field on z-axis [uT] \*/

};

/\*\* defines mag data with uncertainties \*/

struct MagneticData : public MagneticVector

{

int64\_t timestamp; /\*\*< unix time [ms] \*/

double covarianceMatrix[3][3]; /\*\*< mag meassurement covariance matrix [uT^2], column order\n

\* cov(mX, mX), cov(mY, mX), cov(mZ, mX)\n

\* cov(mX, mY), cov(mY, mY), cov(mZ, mY)\n

\* cov(mX, mZ), cov(mY, mZ), cov(mZ, mZ) \*/

};

/\*\* defines mag data with uncertainties \*/

struct MagneticMeasurement : public MagneticData

{

double sigma\_mX; /\*\*< x-axis magnetometer meassurement noise [uT] \*/

double sigma\_mY; /\*\*< y-axis magnetometer meassurement noise [uT] \*/

double sigma\_mZ; /\*\*< z-axis magnetometer meassurement noise [uT] \*/

};

* + 1. WiFi/BLE Data Structures

typedef uint64\_t BSSID; /\*\*< mac address in decimal form \*/

/\*\* common rssi measurement\*/

struct RSSIMeasurement

{

int64\_t timestamp; /\*\*< unix time [us] \*/

BSSID mac; /\*\*< MAC addres in decimal form \*/

int8\_t rssi; /\*\*< RSSI value [dbm] \*/

uint16\_t frequency; /\*\*< central channel frequency [MHz] \*/

};

/\*\* WiFi measurement \*/

typedef RSSIMeasurement WiFiMeasurement;

/\*\* BLE measurement (iBeacon) \*/

struct BleMeasurement : public RSSIMeasurement

{

uint16\_t major; /\*\*< iBeacon major number \*/

uint16\_t minor; /\*\*< iBeacon minor number \*/

uint8\_t uuid[16]; /\*\*< iBeacon uuid 128 bit value \*/

int8\_t txPower; /\*\*< iBeacon tx power level [dbm] on 1m distance \*/

bool hasMAC; /\*\*< mac address avaliability flag\*/

};

/\*\* WiFi scan result\*/

struct WiFiScanResult

{

int64\_t timestamp; /\*\*< UNIX time in [ms] \*/

std::vector<WiFiMeasurement> scanWiFi; /\*\*< WiFi observation \*/

};

/\*\* BLE scan result\*/

struct BleScanResult

{

int64\_t timestamp; /\*\*< UNIX time in [ms] \*/

std::vector<BleMeasurement> scanBle; /\*\*< BLE observation \*/

};

* + 1. TPN (IRL) Data Structures

struct TpnPosition

{

double lattitude; ///< latitude, [deg] [-90..90]

double longitude; ///< longitude, [deg] [-180..180]

double user\_heading; ///< speed vector direction [deg] [-180..180] (user heading)

double sigma\_lattitude; ///< [m] std north

double sigma\_longitude; ///< [m] std east

double sigma\_user\_heading; ///< [deg]

double misalignment; ///< user/device heading misalignment angle, [deg]

double sigma\_misalignment; ///< user/device heading misalignment std, [deg]

int16\_t floor; ///< floor number

double altitude; ///< altitude above sea level [m]

double sigma\_altitude; ///< altitude standard deviation [m]

bool is\_valid; ///< position validity flag

};

/\*\* Structure of device attitude of TPN \*/

struct TpnAttitude

{

int8\_t orientation\_id; ///< orientation based on pitch [-1: vertical up; 0: horizontal; +1: vertical down]

///< axis definition is described in RTFPPL design document

float roll; ///< device roll, [deg] [-180..180]

float pitch; ///< device pitch, [deg] [-90..90]

float heading; ///< device heading, [deg] [-180..180]

float sigma\_roll; ///< device roll standard deviation

float sigma\_pitch; ///< device pitch standard deviation

float sigma\_heading; ///< device heading standard deviation

int8\_t alignment\_status; ///< TPN alignment flag

bool is\_valid; ///< attitude validity flag

};

/\*\* defines mag measurements with uncertainties \*/

struct TpnMagneticMeasurement

{

double mX; /\*\*< magnetic field on x-axis [mG] \*/

double mY; /\*\*< magnetic field on y-axis [mG] \*/

double mZ; /\*\*< magnetic field on z-axis [mG] \*/

double sigma\_mX; /\*\*< x-axis magnetometer measurement noise [mG] \*/

double sigma\_mY; /\*\*< y-axis magnetometer measurement noise [mG] \*/

double sigma\_mZ; /\*\*< z-axis magnetometer measurement noise [mG] \*/

double covarianceMatrix[3][3]; /\*\*< mag bias error covariance matrix [mG^2], column order\n

\* cov(mX, mX), cov(mY, mX), cov(mZ, mX)\n

\* cov(mX, mY), cov(mY, mY), cov(mZ, mY)\n

\* cov(mX, mZ), cov(mY, mZ), cov(mZ, mZ) \*/

};

* + 1. Grid Data Structures

namespace Fpbl

{

/\*\*

\* Operation status codes \*/

enum class ReturnStatus

{

STATUS\_SUCCESS = 0, /\*\*< operation success \*/

STATUS\_UNKNOWN\_ERROR = -1 /\*\*< undefined error occurs due operation \*/

};

/\*\* input coordinates description (in WGS84)\*/

struct Position

{

double x; /\*\*< x [m] \*/

double y; /\*\*< y [m] \*/

double altitude; /\*\*< altitude (sea level?) [m] \*/

int16\_t floorNumber; /\*\*< discrete floor number [0..32767], must be positive or zero \*/

double covarianceXY[2][2]; /\*\*< Latitude/Longitude covariance matrix in column order [rad^2]\n

\* cov(lat,lat), cov(lon,lat)\n

\* cov(lat,lon), cov(lon,lon) \*/

double altitudeStd; /\*\*< altitude standard deviation [m] \*/

double floorStd; /\*\*< altitude standard deviation [floor] \*/

bool isValid; /\*\*< position validity flag \*/

};

/\*\* Attitude information (relative to ENU?) \*/

struct Attitude

{

double quaternion[4]; /\*\*< orientation quaternion [q0, q1, q2, q3] should be normalized \*/

double covarianceQuaternion[4][4]; /\*\*< quaternion covariance matrix, column order\n

\* cov(q0, q0), cov(q1, q0), cov(q2, q0), cov(q3, q0)\n

\* cov(q0, q1), cov(q1, q1), cov(q2, q1), cov(q3, q1)\n

\* cov(q0, q2), cov(q1, q2), cov(q2, q2), cov(q3, q2)\n

\* cov(q0, q3), cov(q1, q2), cov(q2, q3), cov(q3, q3) \*/

bool isValid; /\*\*< orientation validity flag \*/

};

/\*\* defines cell configuration in grid\*/

enum class CellType

{

CELL\_SQUARE = 0, /\*\*< use square cells \*/

CELL\_HEXAGONAL = 1 /\*\*< use hexagonal cells \*/

};

/\*\*in grid coordinates \*/

struct CoordinatesInGrid

{

double x; ///< x-coordinate [m]

double y; ///< y-coordinate [m]

int16\_t floor; ///< discrete floor number [floor], must be positive or zero

};

/\*\* defines grid configuration \*/

struct Grid

{

CellType type; /\*\*< cell type \*/

double size; /\*\*< cell size [m]\*/

CoordinatesInGrid min; /\*\*< left bottom corner of grid \*/

CoordinatesInGrid max; /\*\*< right top corner of grid \*/

bool enable; // enable/disable grid generation

};

/\*\* defines finger print configuration \*/

struct FPGrid

{

Grid grid;

int min\_data\_count; // minimal permited data number per cell

int max\_bssid\_count; // max AP number in FP

double min\_ig; // min information gain value to include AP in FP

};

/\*\* assigned magnetic data\*/

struct MagneticCell

{

CoordinatesInGrid coordinates; ///< cell coordinates

std::vector<MagneticData> magData; ///< magnetic data

};

/\*\* assigned wifi data\*/

struct WifiCell

{

CoordinatesInGrid coordinates; ///< cell coordinates

std::vector<WiFiScanResult> wifiData; ///< wifi data

};

/\*\* assigned BLE data\*/

struct BleCell

{

CoordinatesInGrid coordinates; ///< cell coordinates

std::vector<BleScanResult> bleData; ///< ble data

};

/\*\* assigned to grid magnetic data\*/

typedef std::vector<MagneticCell> MagneticGrid;

/\*\* assigned to grid wifi data\*/

typedef std::vector<WifiCell> WiFiGrid;

/\*\* assigned to grid ble data\*/

typedef std::vector<BleCell> BleGrid;

};

* 1. API Classes and their Methods

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

* + 1. GridBuilder Class and its Methods

This class is used to realize creation of magnetic/WiFi/BLE grids.

namespace Fpbl

{

class GridBuilder

{

public:

/\*\* default constructor \*/

GridBuilder();

/\*\* destructor \*/

~GridBuilder();

/\*\* \return version info\*/

VersionNumber getVersionNumber() const;

// ReturnStatus loadModule(const std::string &module);

// ReturnStatus unloadModule(const std::string &module);

// std::string getLoadedModules() const;

// ReturnStatus setModuleParameter(const std::string &module, const std::string &parameter, const double &value);

/\*\* Puts single path coordinates into processing

\* \param[in] timestamp unix time in [us]

\* \param[in] devicePosition coordinates with uncertainties

\* \return status code\*/

ReturnStatus processDevicePosition( const uint64\_t &timestamp, const Position &devicePosition );

/\*\* Puts single path device orientation into processing

\* \param[in] timestamp unix time in [us]

\* \param[in] deviceAttitude attitude with uncertainties

\* \return status code\*/

ReturnStatus processDeviceAttitude( const uint64\_t &timestamp, const Attitude &deviceAttitude );

/\*\* Puts single path device orientation into processing

\* \param[in] irlOutput irl position attitude and magnetic data

\* \return status code\*/

ReturnStatus processIrlOutput( const TpnOutput &irlOutput );

/\*\* Puts single path wifi measurements into processing

\* \param[in] wifiScan WiFi scan result

\* \return status code\*/

ReturnStatus processWiFiMeasurement( const WiFiScanResult &wifiScan );

/\*\* Puts single path ble measurements into processing

\* \param[in] bleScan BLE scan result

\* \return status code\*/

ReturnStatus processBleMeasurement( const BleScanResult &bleScan );

/\*\* Puts single path wifi measurements into processing

\* \param[in] timestamp unix time in [us]

\* \param[in] wifi WiFi scan result

\* \return status code\*/

ReturnStatus processWiFiMeasurement( const uint64\_t &timestamp, const WiFiScanResult &wifiScan );

/\*\* Puts single path ble measurements into processing

\* \param[in] timestamp unix time in [us]

\* \param[in] ble BLE scan result

\* \return status code\*/

ReturnStatus processBleMeasurement( const uint64\_t &timestamp, const BleScanResult &bleScan );

/\*\* Puts single path ble measurements into processing

\* \param[in] timestamp unix time in [us]

\* \param[in] mag magnetic measurement vector

\* \return status code\*/

ReturnStatus processMFPMeasurement( const uint64\_t &timestamp, const MagneticVector &mag );

/\*\* Setup venue params

\* \param[in] venue venue parameters

\* \return status code \*/

ReturnStatus setVenue( const Venue &venue );

/\*\* Sets temporary folder with write access

\* \param[in] folderPath full path

\* \return status code \*/

ReturnStatus setTemporaryFolder( const std::string &folderPath );

/\*\* return filled wifi grid for selected venue and grid configuration

\* \param[in] venueId unique venue id

\* \param[in] grid grid configuration

\* \param[in,out] wifiGrid grid with assigned wifi data

\* \return status code \*/

ReturnStatus updateGridWiFi( const venue\_id &venueId, const Grid &grid, WiFiGrid \* wifiGrid );

/\*\* return filled ble grid for selected venue and grid configuration

\* \param[in] venueId unique venue id

\* \param[in] grid grid configuration

\* \param[in,out] bleGrid grid with assigned ble data

\* \return status code \*/

ReturnStatus updateGridBLE( const venue\_id &venueId, const Grid &grid, BleGrid \* bleGrid );

/\*\* return filled mfp grid for selected venue and grid configuration

\* \param[in] venueId unique venue id

\* \param[in] grid grid configuration

\* \param[in,out] magGrid grid with assigned mfp data

\* \return status code \*/

ReturnStatus updateGridMFP( const venue\_id &venueId, const Grid &grid, MagneticGrid \* magGrid );

/\*\* return filled mfp grid for selected venue and grid configuration

\* \param[in] venueId unique venue id

\* \param[in] grid grid configuration

\* \param[in,out] magGrid grid with assigned mfp data

\* \return status code \*/

ReturnStatus updateGridMFP( const venue\_id &venueId, const Grid &grid, MagneticGrid & magGrid );

/\*\* create empty mfp grid for selected venue and grid configuration

\* \param[in] venueId unique venue id

\* \param[in] grid grid configuration

\* \param[out] empty magGrid grid

\* \return status code \*/

ReturnStatus createEmptyGridMFP(const venue\_id &venueId, const Grid &grid, MagneticGrid & magGrid);

/\*\* create empty WiFi grid for selected venue and grid configuration

\* \param[in] venueId unique venue id

\* \param[in] grid grid configuration

\* \param[out] empty WiFi wifiGrid

\* \return status code \*/

ReturnStatus createEmptyGridWiFi(const venue\_id &venueId, const Grid &grid, WiFiGrid & wifiGrid);

/\*\* create empty Ble grid for selected venue and grid configuration

\* \param[in] venueId unique venue id

\* \param[in] grid grid configuration

\* \param[out] empty Ble wifiGrid

\* \return status code \*/

ReturnStatus createEmptyGridBLE(const venue\_id &venueId, const Grid &grid, BleGrid & bleGrid);

};

}

* + 1. MfpBuilder Class and its Methods

This class is used to realize creation of magnetic fingerprint map.

namespace Fpbl

{

/\*\*

\* MfpBuilder process mfp-grid data and generates fingerprint

\*/

class MfpBuilder

{

public:

/\*\* fingerprint cell description, GM model \*/

struct DBRecord

{

/\*\* gaussian mix model parameters \*/

struct GaussMix

{

float mu1; /\*\*< gaussian #1 mean \*/

float s1; /\*\*< gaussian #1 std \*/

float w1; /\*\*< gaussian #1 weight \*/

float mu2; /\*\*< gaussian #2 mean \*/

float s2; /\*\*< gaussian #2 std \*/

float w2; /\*\*< gaussian #2 weight \*/

float reserved[2]; /\*\*< reserved fileds \*/

};

/\*\* field componets \*/

///@{

GaussMix x;

GaussMix y;

GaussMix z;

///@}

};

class LocalDB

{

public: typedef std::vector<DBRecord> Records;

Records records;

Grid grid;

typedef Records::const\_iterator const\_iterator;

typedef Records::iterator iterator;

public:

LocalDB( Grid grid );

const\_iterator cbegin() const;

const\_iterator cend() const;

Grid getGrid() const;

ReturnStatus getRecord( const CoordinatesInGrid &position, DBRecord \*record ) const;

iterator findRecord( const CoordinatesInGrid &position );

size\_t size() const;

size\_t size\_x() const;

size\_t size\_y() const;

size\_t size\_floor() const;

size\_t getCellsCount() const;

};

MfpBuilder();

~MfpBuilder();

ReturnStatus buildFingerprint( const MagneticGrid &mfpGrid, LocalDB \*mfpFp );

};

}

* + 1. WiFiBuilder Class and its Methods

namespace Fpbl{

/\*\*

\* WiFiBuilder generate WiFi fingerprint for specified area

\*/

class WiFiBuilder

{

public:

typedef struct

{

double mu1; /\*\*< gaussian #1, mean \*/

double sig1; /\*\*< gaussian #1, sigma \*/

double w1; /\*\*< gaussian #1, weight \*/

double mu2; /\*\*< gaussian #2, mean \*/

double sig2; /\*\*< gaussian #2, sigma \*/

double w2; /\*\*< gaussian #2, weight \*/

} GMixture;

//typedef std::vector<double> APHist; /\*\*< single AP histogram model \*/

//typedef std::map<BSSID, APHist> APList; /\*\*< location histogram model \*/

typedef std::map<BSSID, GMixture> APListGM; /\*\*< location GM model \*/

/\*\* fingerprint record \*/

typedef struct

{

//APList fingerprint; //< AP histograms

APListGM fingerprintGM; //< AP GMs

Fpbl::CoordinatesInGrid location; //< coordinates (x,y,z)

} DBRecord;

/\*\* fingerprint object \*/

typedef std::vector<DBRecord> LocalDB;

WiFiBuilder(); /\*\*< default constructor \*/

IWiFiBuilder(Fpbl::FPGrid wifi\_fp); /\*\*< constructor with finger print settings \*/

~WiFiBuilder(); /\*\*< destructor \*/

/\*\*

\* process wifi-grid and builds fingerprint for specified area

\* \param[in] wifiGrid wifi-grid from GridBuilder

\* \param[out] wifiFp generated fingerprint

\* \return success status

\*/

Fpbl::ReturnStatus buildFingerprint(const WiFiGrid &wifiGrid, LocalDB \*wifiFp);

};

}

* + 1. BleBuilder Class and its Methods

namespace Fpbl{

class BleBuilder

{

public:

typedef struct

{

double mu1; /\*\*< gaussian #1, mean \*/

double sig1; /\*\*< gaussian #1, sigma \*/

double w1; /\*\*< gaussian #1, weight \*/

double mu2; /\*\*< gaussian #2, mean \*/

double sig2; /\*\*< gaussian #2, sigma \*/

double w2; /\*\*< gaussian #2, weight \*/

} GMixture;

//typedef std::vector<double> APHist; /\*\*< single AP histogram model \*/

//typedef std::map<BSSID, APHist> APList; /\*\*< location histogram model \*/

typedef std::map<BSSID, GMixture> APListGM; /\*\*< location GM model \*/

/\*\* fingerprint record \*/

typedef struct

{

//APList fingerprint; //< AP histograms

APListGM fingerprintGM; //< AP GMs

Fpbl::CoordinatesInGrid location; //< coordinates (x,y,z)

} DBRecord;

/\*\* fingerprint object \*/

typedef std::vector<DBRecord> LocalDB;

BleBuilder(); /\*\*< default constructor \*/

BleBuilder(FPGrid ble\_fp); /\*\*< constructor with finger print settings \*/

~BleBuilder(); /\*\*< destructor \*/

/\*\*

\* process blei-grid and builds fingerprint for specified area

\* \param[in] bleGrid ble-grid from GridBuilder

\* \param[out] bleFp generated fingerprint

\* \return success status

\*/

Fpbl::ReturnStatus buildFingerprint(const BleGrid &bleGrid, LocalDB \*bleFp);

};

}

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

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

Fpbl::GridBuilder gridBuilder;

Fpbl::MagneticGrid magneticGrid; // created once, then updated for each track

Fpbl::WiFiGrid wifiGrid;

Fpbl::BleGrid bleGrid;

Fpbl::Grid mag\_grid;

Fpbl::Grid wifi\_grid;

Fpbl::Grid ble\_grid;

mag\_grid.type = CELL\_SQUARE; /\*\*< cell type \*/

mag\_grid.size = 1; /\*\*< cell size [m]\*/

/\* left bottom corner of grid \*/

mag\_grid.min.x = …;

mag\_grid.y = …;

mag\_grid.floor = …;

/\*\*< right top corner of grid \*/

mag\_grid.max.x = …;

mag\_grid.max.y = …;

mag\_grid.max.floor = …;

wifi\_grid.type = CELL\_SQUARE; /\*\*< cell type \*/

wifi\_grid.size = 1; /\*\*< cell size [m]\*/

/\* left bottom corner of grid \*/

wifi\_grid.min.x = …;

wifi\_grid.y = …;

wifi\_grid.floor = …;

/\*\*< right top corner of grid \*/

wifi\_grid.max.x = …;

wifi\_grid.max.y = …;

wifi\_grid.max.floor = …;

ble\_grid.type = CELL\_SQUARE; /\*\*< cell type \*/

ble\_grid.size = 1; /\*\*< cell size [m]\*/

/\* left bottom corner of grid \*/

ble\_grid.min.x = …;

ble\_grid.y = …;

ble\_grid.floor = …;

/\*\*< right top corner of grid \*/

ble\_grid.max.x = …;

ble\_grid.max.y = …;

ble\_grid.max.floor = …;

* 1. Setting Venue Parameters

To set venue parameters Venue object should be constructed.

For example:

Venue venue;

venue.id = …; /\*\*< venue identifier \*/

venue.origin\_lattitude = …; /\*\*< bottom left corner latitude [rad]

[-pi/2..pi/2] \*/

venue.origin\_longitude = …; /\*\*< bottom left corner longitude [rad]

[-pi..pi] \*/

venue.origin\_altitude = …; /\*\*< zero floor altitude [m] \*/

venue.origin\_azimuth = …; /\*\*< venue x-axix rotation to true north

[rad] [0..2pi] \*/

venue.floors\_count = …; /\*\*< total floor number in venue \*/

venue.size\_x = …; /\*\*< x axis venue size [m] \*/

venue.size\_y = …; /\*\*< y axis venue size [m] \*/

venue.alfa = …; /\*\*< axis transformation param (scale factor) \*/

venue.beta = …; /\*\*< axis transformation param (scale factor) \*/

And the following method must be called:

gridBuilder**.**setVenue ( venue );

* 1. Processing FPBL data

After FPBL is properly set up it can start processing input data. Input data of the actual interface contains the following items:

* IRL data
* WiFi/BLE data.

An alternative to IRL data interface is also supported for compatibility with the older version of FPBL:

* Position and Attitude data;
* Magnetic data;

The alternative interface described in the section 5.6 should not be used for processing IRL data.

* 1. Processing IRL data

The following objects should be created:

TpnOutput irlOutput**;**

irlOutput.timestamp = …\_;

// position data

irlOutput.position.lattitude = …\_;

irlOutput.position.longitude = …\_;

irlOutput.position.sigma\_north = …\_;

irlOutput.position.sigma\_east = …\_;

irlOutput.position.altitude = …\_;

irlOutput.position.floor = …\_;

irlOutput.position.sigma\_altitude = …\_;

irlOutput.position.misalignment = …\_;

irlOutput.position.user\_heading = …\_;

irlOutput.position.sigma\_user\_heading = …\_;

irlOutput.position.fidgeting\_flag = …;

irlOutput.position.navigation\_phase = …;

irlOutput.position.is\_valid = …;

// attitude data

irlOutput.attitude.roll = …;

irlOutput.attitude.pitch = …;

irlOutput.attitude.heading = …;

irlOutput.attitude.sigma\_roll = …;

irlOutput.attitude.sigma\_pitch = …;

irlOutput.attitude.sigma\_heading = …;

irlOutput.attitude.orientation\_id = …;

irlOutput.attitude.is\_valid = …;

// IRL magnetic data

irlOutput.mag\_meas.is\_valid = …\_;

irlOutput.mag\_meas.is\_valid = …;

irlOutput.mag\_meas.mX = …;

irlOutput.mag\_meas.mY = …;

irlOutput.mag\_meas.mZ = …\_;

irlOutput.mag\_meas.sigma\_mX = …;

irlOutput.mag\_meas.sigma\_mY = …;

irlOutput.mag\_meas.sigma\_mZ = …;

// Covariance matrix of magnetometer biases

for ( int i = 0; i < 3; i++ )

for ( int j = 0; j < 3; j++ )

irlOutput.mag\_meas.covarianceMatrix[i][j] = …;

Then the following method must be called:

Fpbl::ReturnStatus status = Builder.processIrlOutput( irlOutput );

Repeat this call until all the input data is processed.

Then the following methods must be called

MagneticGrid magGrid;

status = createEmptyGridMFP( venue.id, mag\_grid, magGrid );

status = createEmptyGridMFP( venue.id, mag\_grid, magGrid );

* 1. Processing WiFi/BLE data

Processing WiFi/BLE input data is similar to processing magnetic input data.

The following objects should be created:

WiFiScanResult scan\_wifi;

BleScanResult scan\_ble;

Also some number of WiFiMeasurement and BleMeasurement objects should be created.

For example there are one hundred WiFi access point (AP) and ten BLE AP.

Then one hundred WiFiMeasurement objects and ten BleMeasurement objects have to be created.

WiFiMeasurement wifiMeas[100];

BleMeasurement bleMeas[10];

These arrays are filled with input data:

wifiMeas[0].timestamp = …; /\*\*< Unix time [us] \*/

wifiMeas[0].mac = …; /\*\*< MAC address in decimal form \*/

wifiMeas[0].rssi = …; /\*\*< RSSI value [dbm] \*/

wifiMeas[0].frequency = …; /\*\*< central channel frequency [MHz] \*/

wifiMeas[1].timestamp = …; /\*\*< Unix time [us] \*/

wifiMeas[1].mac = …; /\*\*< MAC address in decimal form \*/

wifiMeas[1].rssi = …; /\*\*< RSSI value [dbm] \*/

wifiMeas[1].frequency = …; /\*\*< central channel frequency [MHz] \*/

and etc.

Similarly input data is filled for BLE:

bleMeas[0].timestamp = …; /\*\*< Unix time [us] \*/

bleMeas[0].mac = …; /\*\*< MAC address in decimal form \*/

bleMeas[0].rssi = …; /\*\*< RSSI value [dbm] \*/

bleMeas[0].frequency = …; /\*\*< central channel frequency [MHz] \*/

bleMeas[1].timestamp = …; /\*\*< Unix time [us] \*/

bleMeas[1].mac = …; /\*\*< MAC address in decimal form \*/

bleMeas[1].rssi = …; /\*\*< RSSI value [dbm] \*/

bleMeas[1].frequency = …; /\*\*< central channel frequency [MHz] \*/

Also for BLE aadditional fields should be filled with data:

bleMeas[0].major = …; /\*\*< iBeacon major number \*/

bleMeas[0].minor = …; /\*\*< iBeacon minor number \*/

bleMeas[0].uuid[0] = …; /\*\*< iBeacon uuid 128 bit value \*/

……………………

bleMeas[0].uuid[15] = …;

bleMeas[0].txPower = …; /\*\*< iBeacon tx power level

[dbm] on 1m distance \*/

bleMeas[0].hasMAC = …; /\*\*< mac address availability flag\*/

bleMeas[1].major = …; /\*\*< iBeacon major number \*/

bleMeas[1].minor = …; /\*\*< iBeacon minor number \*/

bleMeas[1].uuid[0] = …; /\*\*< iBeacon uuid 128 bit value \*/

……………………

bleMeas[1].uuid[15] = …;

bleMeas[1].txPower = …; /\*\*< iBeacon tx power level

[dbm] on 1m distance \*/

bleMeas[1].hasMAC = …; /\*\*< mac address availability flag\*/

and etc.

Then scan\_wifi and scan\_ble must be filled with data.

scan\_wifi.clear();

scan\_wifi.push\_back(wifiMeas[0]);

scan\_wifi.push\_back(wifiMeas[1]);

…………….

scan\_ble.clear();

scan\_ble.push\_back(bleMeas[0]);

scan\_ble.push\_back(bleMeas[1]);

…………….

and etc.

Then the following methods must be called:

Fpbl::ReturnStatus status = Builder.processWiFiMeasurement( wifiScan.timestamp, wifiScan );

Fpbl::ReturnStatus status = Builder.processBleMeasurement( bleScan.timestamp, bleScan );

Repeat these calls until all the input data is processed.

Then the following methods must be called

WiFiGrid wifiGrid;

BleGrid bleGrid;

status = createEmptyGridWiFi( venue.id, wifi\_grid, wifiGrid );

status = createEmptyGridBLE( venue.id, wifi\_grid, wifiGrid );

status = Builder.updateGridWiFi( venue.id, wifi\_grid, wifiGrid );

status = Builder.updateGridBLE( venue.id, ble\_grid, bleGrid );

* 1. Alternative Interface

An alternative interface is left for compatibility with the older version of FPBL. It should not be used for processing IRL data.

* + 1. Processing Position and Attitude data

The following objects should be created:

Fpbl::Position position;

Fpbl::Attitude attitude;

uint64\_t timestamp;

Then input data should be passed to it.

For example:

timestamp = …;

position.x = …; /\*\*< x [m] \*/

position.y = …; /\*\*< y [m] \*/

position.altitude = …; /\*\*< altitude (sea level?) [m] \*/

position.floorNumber = …; /\*\*< discrete floor number [0..32767], must be positive or zero \*/

position.covarianceXY[0][0] = …; /\*\*< Latitude/Longitude covariance matrix in column order [rad^2]\n

\* cov(lat,lat), cov(lon,lat)\n

\* cov(lat,lon), cov(lon,lon) \* position.covarianceXY[0][1] = …;

position.covarianceXY[1][0] = …;

position.covarianceXY[1][1] = …;

position.altitudeStd; /\*\*< altitude standard deviation [m] \*/

position.floorStd; /\*\*< altitude standard deviation [floor] \*/

position.isValid;

attitude.quaternion[0] = …; /\*\*< orientation quaternion [q0, q1, q2, q3] should be normalized \*/

attitude.quaternion[1] = …;

attitude.quaternion[2] = …;

attitude.quaternion[3] = …;

attitude.covarianceQuaternion[0][0] = …; /\*\*< quaternion covariance matrix, column order\n

\* cov(q0, q0), cov(q1, q0), cov(q2, q0), cov(q3, q0)\n

\* cov(q0, q1), cov(q1, q1), cov(q2, q1), cov(q3, q1)\n

\* cov(q0, q2), cov(q1, q2), cov(q2, q2), cov(q3, q2)\n

\* cov(q0, q3), cov(q1, q2), cov(q2, q3), cov(q3, q3) \*/

attitude.covarianceQuaternion[0][1] = …;

attitude.covarianceQuaternion[0][2] = …;

attitude.covarianceQuaternion[0][3] = …;

attitude.covarianceQuaternion[1][0] = …;

attitude.covarianceQuaternion[1][1] = …;

attitude.covarianceQuaternion[1][2] = …;

attitude.covarianceQuaternion[1][3] = …;

attitude.covarianceQuaternion[2][0] = …;

attitude.covarianceQuaternion[2][1] = …;

attitude.covarianceQuaternion[2][2] = …;

attitude.covarianceQuaternion[2][3] = …;

attitude.covarianceQuaternion[3][0] = …;

attitude.covarianceQuaternion[3][1] = …;

attitude.covarianceQuaternion[3][2] = …;

attitude.covarianceQuaternion[3][3] = …;

attitude.isValid = …; /\*\*< orientation validity flag \*/

and the following methods should be called:

Fpbl::ReturnStatus status1 = Builder.processDevicePosition( timestamp, position );

Fpbl::ReturnStatus status2 = Builder.processDeviceAttitude( timestamp, attitude );

Call these methods again until all the input data is processed.

* + 1. Processing Magnetic data

The following objects should be created:

MagneticVector mag\_vector;

uint64\_t timestamp;

Then input data should be passed to it.

For example:

timestamp = …;

vector.mX = …; /\*\*< magnetic field on x-axis [uT] \*/

vector.mY = …; /\*\*< magnetic field on y-axis [uT] \*/

vector.mZ = …; /\*\*< magnetic field on z-axis [uT] \*/

and the following method should be called:

Fpbl::ReturnStatus status = Builder.processMFPMeasurement( timestamp, mag\_vector );

Call this method again until all the input data are processed.

At last, the following method must be called:

Fpbl::ReturnStatus status = Builder.updateGridMFP( venue.id, mag\_grid, &magneticGrid );

* 1. Creating Fingerprint databases

After processing all input data, fingerprint databases can be created.

For this purpose, the following methods must be called:

* For MFP

Fpbl::MfpBuilder \*pMfp = new Fpbl::MfpBuilder();

Fpbl::MfpBuilder::LocalDB dbmfp( mag\_grid );

Fpbl::ReturnStatus result = pMfp->buildFingerprint( magneticGrid, &dbmfp );

* For WiFi FP

Fpbl::WiFiBuilder \*pWiFi = new Fpbl::WiFiBuilder();

Fpbl::WiFiBuilder::LocalDB dbwifi;

result = pWiFi->buildFingerprint( wifiGrid, &dbwifi );

* For BLE FP

Fpbl::BleBuilder \*pBle = new Fpbl::BleBuilder();

Fpbl::BleBuilder::LocalDB dbble;

result = pBle->buildFingerprint( bleGrid, &dbble );

* 1. Source code of integration examples

### Console application

Good example of usage fingerprint builder library is a PC console application. The application allows to feed the library 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