

Real Time Fingerprint Positioning Library (RTFPPL) for Retail Phase II

API and Integration Guide

Ver 1.1

Table of Contents

[1. Purpose 4](#_Toc450331581)

[2. Abbreviations 5](#_Toc450331582)

[3. Document History 7](#_Toc450331583)

[4. RTFPPL API 8](#_Toc450331584)

[4.1. API Data Structs 8](#_Toc450331585)

[4.2. API Classes and their Methods 11](#_Toc450331586)

[5. Example Integration of RTFPPL 15](#_Toc450331587)

[5.1. Creation of RTFPPL and setting the fingerprint data bases 15](#_Toc450331588)

[5.2. Enabling/disabling the use of WiFi, BLE and geomagnetic data for getting mixed position 15](#_Toc450331589)

[5.3. Setting callback for getting position 15](#_Toc450331590)

[5.4. Setting Venue Parameters 17](#_Toc450331591)

[5.5. Processing TPN data 17](#_Toc450331592)

[5.6. Processing WiFi/BLE data 18](#_Toc450331593)

[6. Reference 21](#_Toc450331594)

1. Purpose

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

1. Abbreviations

The following abbreviations are used in this document:

|  |  |
| --- | --- |
| AP | Access point |
| BLE | Bluetooth Low Energy |
| MEMS | Micro Electro-Mechanical Sensors: accelerometers, gyroscopes and magnetometer. |

1. Document History

|  |  |  |
| --- | --- | --- |
| **Date** | **Version** | **Comment** |
| 20 April 2016 | 1.0 | Document created |
| 01 November  2016 | 1.1 |  |

1. RTFPPL API

RTFPPL API contains the common structures for description input/output data, class FPEngine that realizes all RTFPPL algorithms and pure virtual class IPositionUpdate that has to be redefined in application to organize getting the output data. API is realized into namespace Fppe.

RTFPPL API methods may be divided into the following parts

* Creation of RTFPPL
* Setting the Wi-Fi, BLE, magnetic fingerprint data bases
* Enabling/disabling the use of Wi-Fi, BLE, magnetic for getting mixed position
* Setting callback for getting position
* Set Venue parameters
* Processing TPN data
* Processing WiFi data
* Processing BLE data
* Getting the position
  1. API Data Structures

In this item the common structures for description input/output data are described

/\*\* Fingerprint position engine \*/

namespace Fppe

{

/\*\* build type \*/

enum class VersionNumberReleaseId

{

VERSION\_ALPHA = 0, /\*\*< alpha version \*/

VERSION\_BETA = 1, /\*\*< beta version \*/

VERSION\_RELEASE\_CANDIDATE = 2, /\*\*< pre release \*/

VERSION\_RELEASE = 3 /\*\*< release \*/

};

/\*\* library build info \*/

struct VersionNumber

{

uint8\_t major; /\*\*< major version number \*/

uint8\_t minor; /\*\*< minor version number \*/

uint32\_t build; /\*\*< plain build number \*/

VersionNumberReleaseId release\_id; /\*\*< build type \*/

};

/\*\*

\* 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

{

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

double lattitude; /\*\*< lattitude [deg] [-90..+90]\*/

double longitude; /\*\*< longitude [deg] [-180..+180] \*/

double azimuth; /\*\*< direction to north [deg] [-180..+180] \*/

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

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

double covariance\_lat\_lon[2][2]; /\*\*< Lattitude/Longitude covariance matrix in column order [rad^2]\n

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

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

double azimuth\_std; /\*\*< azimuth standard deviation [deg] \*/

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

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

};

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

struct Attitude

{

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

double covariance\_quaternion[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, q3), cov(q2, q3), cov(q3, q3) \*/

bool is\_valid; /\*\*< orientation validity flag \*/

};

/\*\* Coordinates increment\*/

struct CoordinatesIncrement

{

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

double d\_x; /\*\*< x-coordinate increment [m] \*/

double d\_y; /\*\*< y-coordinate increment [m] \*/

double d\_floor; /\*\*< z-axis increment [floor] \*/

double covariance\_yx[2][2]; /\*\*< increment covariance matrix in column order [m^2]\n

\* cov(x,x), cov(y,x)\n

\* cov(x,y), cov(y,y) \*/

double d\_floor\_std; /\*\*< z-axis standard deviation [floor] \*/

Attitude attitude; /\*\*< device orientation relative start position \*/

bool is\_step; /\*\*< true if step detected \*/

};

typedef MagneticData MagneticCalibrationParam; /\*\*< magnetic calibration parameters (bias with cov matrix) \*/

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 \*/

};

/\*\* Particle is used for debug and visualization pirposes \*/

struct Particle

{

double x;

double y;

double z;

double w;

};

}

Parts of data structures are realized in header files in Components/Data\_Formats

**Venue.hpp**

typedef uint64\_t venue\_id; /\*\*< unique venue identifier type \*/

/\*\* Venue information, defines transformation to local frame\*/

struct Venue

{

venue\_id id; /\*\*< venue identifier \*/

double origin\_lattitude; /\*\*<

left corner lattitude [deg] [-90..+90] \*/

double origin\_longitude; /\*\*< bottom left corner longitude [deg] [-180..+180] \*/

double origin\_altitude; /\*\*< zero floor altitude [m] \*/

double origin\_azimuth; /\*\*< venue x-axix rotation to true north [deg] [-180..+180] \*/

uint16\_t floors\_count; /\*\*< total floor number in venue \*/

double size\_x; /\*\*< x axis venue size [m] \*/

double size\_y; /\*\*< y axis venue size [m] \*/

double alfa; /\*\*< axis transforamtion param (scale factor) \*/

double beta; /\*\*< axis transforamtion param (scale factor) \*/

double magX; /\*\*< average venue magnetic vector X [uT] \*/

double magY; /\*\*< average venue magnetic vector Y [uT] \*/

double magZ; /\*\*< average venue magnetic vector Z [uT] \*/

};

**TpnData.hpp**

/\*\* Structure of TPN position \*/

struct TpnPosition

{

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

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

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

double sigma\_north; ///< [m] position standard deviation in north direction

double sigma\_east; ///< [m] position standard deviation in east direction

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]

int8\_t navigation\_phase; ///< navigation phase flag; position is avaliable for navigation\_phase > 0

int8\_t fidgeting\_flag; ///< fidjeting flag

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

void print(std::basic\_ostream<char> &out)

{

out << "; " << lattitude; ///< lattitude, [deg] [-90..90]

out << "; " << longitude; ///< longitude, [deg] [-180..180]

out << "; " << user\_heading; ///< speed vector dicrection [deg] [-180..180] (user heading)

out << "; " << sigma\_north; ///< [m] position standard deviation in north direction

out << "; " << sigma\_east; ///< [m] position standard deviation in east direction

out << "; " << sigma\_user\_heading; ///< user heading uncertanties,[deg]; this field also indicates validity of itself, user\_heading and misalignment parameters

/// < these fields are invalid when sigma\_user\_heading <= 0

out << "; " << misalignment; ///< user/device heading misalignment angle [deg]

out << "; " << sigma\_misalignment; ///< user/device heading misalignment std [deg]; reserved

out << "; " << floor; ///< floor number

out << "; " << altitude; ///< altitude above sea level [m]

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

out << "; " << (int)navigation\_phase; ///< navigation phase flag; position is avaliable for navigation\_phase > 0

out << "; " << (int)fidgeting\_flag; ///< fidjeting flag

out << "; " << (int)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: horisontal; +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

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

void print(std::basic\_ostream<char> &out)

{

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

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

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

out << "; " << sigma\_roll; ///< device roll standard deviation

out << "; " << sigma\_pitch; ///< device pitch standard deviation

out << "; " << sigma\_heading; ///< device heading standard deviation

out << "; " << (int)is\_valid; ///< attitude validity flag

}

};

/\*\* defines mag meassurements 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 meassurement noise [mG]

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

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

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

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

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

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

uint8\_t level\_of\_calibration; ///< this parameter describes a consistency class (level) of bias estimation and bias covariance matrix estimation

///< Class 0 - bias unreliable (even if the covariance seems small 1-2uT) maybe only few points were used. Note that the covariance stand for the cluster covariance.

///< Class 1- bias cannot be trusted

///< Class 2- bias maybe trusted

///< Class 3- bias can be trusted

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

void print(std::basic\_ostream<char> &out)

{

out << "; " << mX; ///< magnetic field on x-axis [mG]

out << "; " << mY; ///< magnetic field on y-axis [mG]

out << "; " << mZ; ///< magnetic field on z-axis [mG]

out << "; " << sigma\_mX; ///< x-axis magnetometer meassurement noise [mG]

out << "; " << sigma\_mY; ///< y-axis magnetometer meassurement noise [mG]

out << "; " << sigma\_mZ; ///< z-axis magnetometer meassurement noise [mG]

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

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

out << "; " << covarianceMatrix[i][j]; ///< mag bias error covariance matrix [mG^2], column order

out << "; " << (int)level\_of\_calibration; ///< this parameter describes a consistency class of bias estimation and bias covariance matrix estimation

out << "; " << (int)is\_valid; ///< attitude validity flag

}

};

/\*\* Structure of TPN PDR data \*/

struct TpnPdr

{

double stride\_length; ///< stride length [m]

bool is\_valid; ///< stride length validity flag

void print(std::basic\_ostream<char> &out)

{

out << "; " << stride\_length; ///< stride length [m]

out << "; " << is\_valid; ///< stride length validity flag

}

};

/\*\* TPN output structure\*/

struct TpnOutput

{

double timestamp; ///< timestamp [sec]

TpnPosition position; ///< TPN user position information

TpnAttitude attitude; ///< TPN device attitude information

TpnPdr pdr; ///< Addidtional information from PDR

TpnMagneticMeasurement mag\_meas; ///< mag data synchronized with IRL output

void print(std::basic\_ostream<char> &out)

{

out << timestamp;

out << "; "; position.print(out);

out << "; "; attitude.print(out);

out << "; "; pdr.print(out);

out << "; "; mag\_meas.print(out);

out << std::endl;

}

};

**MagData.hpp**

/\*\* 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

{

MagneticData()

{

}

MagneticData(const MagneticData &c)

{

this->timestamp = c.timestamp;

this->mX = c.mX;

this->mY = c.mY;

this->mZ = c.mZ;

memcpy(this->covarianceMatrix, c.covarianceMatrix, sizeof(c.covarianceMatrix));

}

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. API Classes and their Methods

In this item the classes and their methods are described.

The main class is FPEngine. It is used to realize all RTFPPL algorithms.

/\*\* Forward declaration \*/

class IFPEngine;

/\*\* Main RTFPPL class. Integrates different modules to obtain fused navigation solution \*/

class DLL\_EXPORT FPEngine

{

public:

/\*\* default constructor\*/

FPEngine();

/\*\* destructor\*/

~FPEngine();

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

VersionNumber getVersionNumber() const;

/\*\*

\* resets internal object state

\* \param[in] filename output log filename

\*/

void setLogFile( const std::string &filename );

/\*\*

\* enables logging

\* \param[in] enabled control flag

\*/

void setLogEnabled( const bool &enabled );

/\*\*

\* resets internal object state, disables logging

\*/

void restart();

/\*\*

\* main method, process inertial sensors data and initiates processing pipeline

\* used for debug purposes

\* \param[in] increment position increments in local frame

\*/

void processIncrements( const CoordinatesIncrement &increment );

/\*\*

\* main method, process inertial sensors data and initiates processing pipeline

\* \param[in] tpn\_output position and attitude information it tpn like format

\*/

void processTpnOutput( const TpnOutput &tpn\_output );

/\*\*

\* pushes WiFi measurement into the processing pipeline

\* \param[in] scan\_wifi WiFi measurements vector with timestamp

\*/

void processWiFi( const WiFiScanResult &scan\_wifi );

/\*\*

\* pushes BLE measurement into the processing pipeline

\* \param[in] scan\_ble BLE measurements vector with timestamp

\*/

void processBLE( const BleScanResult &scan\_ble );

/\*\*

\* pushes Magnetic measurement into the processing pipeline

\* \param[in] mag\_data magnetic vector with timestamp

\*/

void processMFP( const MagneticVector &mag\_data, int64\_t timestamp );

/\*\*

\* Sets main filter position callback

\* \param[in] pPosCbk pointer to the callback implementation

\*/

void setPositionCallbackMixed( IPositionUpdate \*pPosCbk );

/\*\*

\* Sets WiFi only position callback

\* \param[in] pPosCbk pointer to the callback implementation

\*/

void setPositionCallbackWiFi( IPositionUpdate \*pPosCbk );

/\*\*

\* Sets BLE only position callback

\* \param[in] pPosCbk pointer to the callback implementation

\*/

void setPositionCallbackBLE( IPositionUpdate \*pPosCbk );

/\*\*

\* Sets MFP only(mfp+pdr) position callback

\* \param[in] pPosCbk pointer to the callback implementation

\*/

void setPositionCallbackMFP( IPositionUpdate \*pPosCbk );

/\*\*

\* initialize WiFi module and loads fingerprint from the array

\* \param[in] pWiFiMap fingerprint pointer to array

\* \param[in] min\_p validity metric threshold. TODO to be moved in fingerprint file

\* \return success status

\*/

ReturnStatus initializeWiFI( const char\* const pWiFiMap, const size\_t wifiFileSizeInBytes, const double min\_p );

/\*\*

\* initialize BLE module and loads fingerprint from the array

\* \param[in] pBleMap fingerprint pointer to array

\* \param[in] min\_p validity metric threshold. TODO to be moved in fingerprint file

\* \return success status

\*/

ReturnStatus initializeBLE( const char\* const pBleMap, const size\_t bleFileSizeInBytes, const double min\_p );

/\*\*

\* initialize MFP module and loads fingerprint from the array

\* \param[in] pMFPMap fingerprint pointer to array

\* \param[in] max\_X defines fingerprint size on X axis [m] [TODO to be moved into FP file]

\* \param[in] max\_Y defines fingerprint size on Y axis [m] [TODO to be moved into FP file]

\* \param[in] cellSize defines internal map discrete [m] [TODO to be moved into FP file]

\* \param[in] minFloor define minimum floor number [TODO to be moved into FP file]

\* \param[in] maxFloor define maximum floor number [TODO to be moved into FP file]

\* \return success status

\*/

ReturnStatus initializeMFP( const char\* const pMFPMap, const size\_t mfpFileSizeInBytes, const double max\_X, const double max\_Y, const double cellSize, const int minFloor, const int maxFloor );

/\*\*

\* Defines venue local frame

\* \param[in] venue venue parameters with local frame origin

\* \return success

\*/

ReturnStatus setVenueParams( const Venue &venue );

/\*\* updater WiFi control

\* \param[in] enable set wiFi update enable/disable

\*/

void setUpdateWiFi( const bool enable );

/\*\* updater BLE control

\* \param[in] enable set BLE update enable/disable

\*/

void setUpdateBLE( const bool enable );

/\*\* updater MFP control

\* \param[in] enable set magnetic update enable/disable

\*/

void setUpdateMFP( const bool enable );

/\*\* Enables floor increment usage, switches to another motion model \*/

void setFloorIncrements( const bool enable );

/\*\* Gets mg calibration params

\* \param[out] bias\_cov esimated magnetic bias with covariance

\*/

void getMagneticBias( MagneticCalibrationParam \*bias\_cov );

/\*\* Gets mg calibration params

\* \param[in] bias\_cov initial magnetic bias with covariance

\*/

void setMagneticBias( const MagneticCalibrationParam &bias\_cov );

/\*\* WiFi calibration status [obsolete]

\* \param[out] bias rssi bias

\* returns ReturnStatus::STATUS\_SUCCESS if success, otherwise error code

\*/

ReturnStatus getWiFiBias( double \*bias );

/\*\* sets WiFi calibration [obsolete]

\* param[in] bias initial rssi bias

\* param[in] enabled enables calibration

\*/

void setWiFiBias( const double &bias, const bool enabled );

/\*\*

\* Sets fine known initial position

\* \param[in] position initial position in global frame

\*/

void setStartPosition( const Position &position );

/\*\*

\* Sets output stream for corresponding log id

\* param[in] idx log index

\* param[in] os output stream

\* return succes status

\*/

bool setLogStream( const unsigned int &idx, std::ostream &os );

/\*\*

\* Return vector of logs brief descriptions

\* param[out] log\_descriptions pointer to descrpitions

\*/

void getLogDescription( std::vector<std::string> \*log\_descriptions );

};

Class IPositionUpdate is pure virtual class (interface) needed to organize getting position and other the output data. Therefore it has to be redefined in application.

namespace Fppe

{

/\*\*

\* Position callback interface

\*/

class IPositionUpdate

{

public:

virtual ~IPositionUpdate()

{

;

}

/\*\*

\* this prototype is used in PC model to visualize particle cloud

\* \param[in] X,Y coordinates in a local frame [m]

\* \param[in] Floor floor level

\* \param[in] H heading [rad]

\* \param[in] Sig estimated position deviation

\* \param[in] t timestamp [ms]

\* \param[in] state pointer to the particles array

\* \param[in] N particles count

\*/

virtual void update( double X, double Y, double Floor, double H, double Sig, double t, const Particle \*state, int N ) = 0;

/\*\*

\* This method is called when new position estimation avaliable

\* \param[in] position navigation solution

\*/

virtual void update( const Position &position ) = 0;

};

}

1. Example Integration of RTFPPL
   1. Setting up the RTFPPL and setting the fingerprint data bases

In order to set up the RTFPPL an Fppe::FPEngine object must be created first.

Fppe::FPEngine fpEngine = new Fppe::FPEngine();

Then the fingerprint data bases must be set. For this purpose the following methods should be called:

Fppe::ReturnStatus status

struct stat file\_buf;

stat(wifi\_db\_name.c\_str(), &file\_buf);

long wifiFileSizeInBytes = file\_buf.st\_size;

char \*pWiFiMap = (char \*)malloc(wifiFileSizeInBytes);

FILE \*pF = fopen(fp\_wifi\_base\_file.c\_str(), "rb");

\_ASSERT(pF);

if (fread(pWiFiMap, wifiFileSizeInBytes, 1, pF) == 0)

fclose(pF);

struct stat file\_buf;

stat(ble\_db\_name.c\_str(), &file\_buf);

long bleFileSizeInBytes = file\_buf.st\_size;

char \*pBleMap = (char \*)malloc(bleFileSizeInBytes);

FILE \*pF = fopen(fp\_ble\_base\_file.c\_str(), "rb");

\_ASSERT(pF);

if (fread(pBleMap, bleFileSizeInBytes, 1, pF) == 0)

fclose(pF);

status = FPc.initializeWiFI(pWiFiMap, wifiFileSizeInBytes, min\_p);

status = FPc.initializeBLE(pBleMap, bleFileSizeInBytes, min\_p);

where wifi\_db\_name, ble\_db\_name are files with WiFi and BLE databases,

min\_p is validity metric threshold (can be set to zero)

struct stat file\_buf;

stat(mfp\_db\_name.c\_str(), &file\_buf);

size\_t mfpMapSizeInBytes = file\_buf.st\_size;

char \*pMfpMap = (char \*)malloc(mfpMapSizeInBytes);

FILE \*pF = fopen(fp\_magnetic\_base\_file.c\_str(), "rb");

\_ASSERT(pF);

fread(pMfpMap, mfpMapSizeInBytes, 1, pF);

fclose(pF);

status = fpEngine->initializeMFP(pMfpMap, mfpMapSizeInBytes, max\_X, max\_Y, ceilSize, minFloor, maxFloor);

where mfp\_db\_name is MFP database file,

pMfpMap is file with MFP pointer to database array,

max\_X, max\_Y are maximum X and Y sizes of MFP database,

ceilSize is the size of MFP database grid cell,

minFloor, maxFloor are min and max floor number in MFP database

* 1. Enabling/disabling the use of WiFi, BLE and geomagnetic data for getting mixed position

It is possible to switch on and off the use of WiFi, BLE or geomagnetic data in calculating the mixed position. For this purpose the following methods should be called:

fpEngine->setUpdateMFP(magEnable); /\*\*< updater MFP control \*/

fpEngine->setUpdateWiFi(wifiEnable); /\*\*< updater WiFi control \*/

fpEngine->setUpdateBLE(bleEnable); /\*\*< updater Ble control \*/

Where magEnable, wifiEnable, bleEnable are a parameters that enables or disables appropriate update.

* 1. Setting callback for getting position

RTFPPL can provide four types of position:

* Magnetic only position
* WiFi only position
* BLE only position
* Mixed position

A class should be created that inherits the Fppe::IPositionUpdate interface class.

For example:

// callback class for writing the position into file

class PositionUpdate : public Fppe::IPositionUpdate

{

public:

PositionUpdate( const std::string &output\_log\_name );

virtual ~PositionUpdate();

virtual void update( double X, double Y, double Floor, double H, double Sig, double t, const Fppe::Particle \*state, int N );

virtual void update( const Fppe::Position &position );

private:

std::ofstream outputstream; /// stream for log file

};

Then four objects of PositionUpdate class must be created.

For example:

PositionUpdate magPosCbk(“mag.log”);

PositionUpdate wifiPosCbk(“wifi.log”);

PositionUpdate blePosCbk(“ble.log”);

PositionUpdate mixedPosCbk(“mixed.log”);

And the following methods should be called:

fpEngine->setPositionCallbackMFP ( &magPosCbk );

fpEngine->setPositionCallbackWiFi( &wifiPosCbk );

fpEngine->setPositionCallbackBLE( &blePosCbk );

fpEngine->setPositionCallbackMixed ( &mixedPosCbk );

* 1. Setting Venue Parameters

To set venue parameters an Venue object should be constructed.

For example:

Venue venue;

// venue parameters

venue.id = 1;

venue.size\_x = 24;

venue.size\_y = 15;

venue.floors\_count = 1;

// venue frame parameters

venue.origin\_lattitude = 55.7351107; // [-90;90]

venue.origin\_longitude = 37.64286405; // [-180;180]

venue.origin\_azimuth = 14.52754; // [-180;180]

venue.origin\_altitude = 0;

venue.alfa = 0; // 0 - for default scaling

venue.beta = 0; // 0 - for default scaling

And the following method must be called:

Fppe::ReturnStatus status = fpEngine->setVenueParams(venue);

* 1. Processing TPN data

After RTFPPL is properly set up it can start processing TPN input data.

The following object should be created:

TpnOutput tpn\_output;

Then input data should be passed to it.

For example:

tpn\_output.timestamp = …; ///< timestamp [sec]

tpn\_output.position.lattitude = …; ///< lattitude, [deg] [-90..90]

tpn\_output.position.longitude = …; ///< longitude, [deg] [-180..180] tpn\_output.position.user\_heading = …; ///< speed vector dicrection

///[deg] [-180..180] (user heading)

tpn\_output.position.sigma\_lattitude = …; ///< [m] std north

tpn\_output.position.sigma\_longitude = …; ///< [m] std east

tpn\_output.position.sigma\_user\_heading = …; ///< [deg]

tpn\_output.position.misalignment = …; ///< user/device heading

/// misalignment angle, [deg]

tpn\_output.position.sigma\_misalignment = …; ///< user/device heading

///misalignment std, [deg]

tpn\_output.position.floor = …; ///< floor number

tpn\_output.position.height = …; ///< user height, [m]

tpn\_output.position.sigma\_height = …; ///< height standard

/// deviation [m]

tpn\_output.position.is\_valid = …; ///< position validity flag

tpn\_output.attitude.orientation\_id = …; ///< orientation based on pitch

/// [-1: vertical up; 0:

/// horisontal; +1: vertical down]

///< axis definition is described in RTFPPL design document[1]

tpn\_output.attitude.roll = …; ///< device roll,

/// [deg] [-180..180]

tpn\_output.attitude.pitch = …; ///< device pitch,

/// [deg] [-90..90]

tpn\_output.attitude.heading = …; ///< device heading,

/// [deg] [-180..180]

tpn\_output.attitude.sigma\_roll = …; ///< device roll

/// standard deviation

tpn\_output.attitude.sigma\_pitch = …; ///< device pitch

/// standard deviation

tpn\_output.attitude.sigma\_heading = …; ///< device heading

/// standard deviation

tpn\_output.attitude.alignment\_status = …; ///< TPN alignment flag

tpn\_output.attitude.is\_valid = …; ///< attitude validity flag

tpn\_output.mag\_data.timestamp = …; /\*\*< UNIX time [ms] \*/

tpn\_output.mag\_data.mX = …; /\*\*< magnetic field on x-axis [uT] \*/

tpn\_output.mag\_data.mY; /\*\*< magnetic field on y-axis [uT] \*/

tpn\_output.mag\_data.mZ; /\*\*< magnetic field on z-axis [uT] \*/

and the following method should be called:

fpEngine->processTpnOutput(tpn\_output);

Feed the tpn\_output with new input data and call fpEngine->processTpnOutput(tpn\_output) again until all the input data is processed.

When RTFPPL calculates the new position the update() method of magPosCbk objects will be called.

* 1. Processing WiFi/BLE data

Processing WiFi/BLE input data is similar to processing TPN 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 addres 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 addres 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 addres 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 addres 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 avaliability 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 avaliability 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:

fpEngine->processWiFi(scan\_wifi );

fpEngine->processBLE(scan\_ble );

Repeat these calls until all the input data is processed.

When RTFPPL calculates the new position the update() method of wifiPosCbk, blePosCbk, mixedPosCbk objects will be called.

* 1. Source code of integration examples

### Simple example

This is a simple example of usage real-time fingerprint positioning library. The example is located at Gift repository on GitHub by the following link: [Gift](https://github.com/InvenSenseInc/Gift)/[Applications](https://github.com/InvenSenseInc/Gift/tree/master/Applications)/example\_positioning. A set of magnetic data is given in the file main.cpp. Expected output position can be seen at Gift\Applications\example\_positioning\Project\out.

### Console application

Good example of usage real-time fingerprint positioning library is a PC console application. The application allows to feed the library with data from TPN output. In return, the positioning library generates indoor position.

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\_positioning.console/.

1. Reference
2. Real Time Fingerprint Positioning Library (RTFPPL) for Retail Phase II. Algorithmic Design Document