

Finger Print Verification Library

(FPVL)

for Retail Phase II

API and Integration Guide

Ver 1.0

Table of Contents

[Table of Contents 3](#_Toc482204644)

[1. Purpose 4](#_Toc482204645)

[2. Abbreviations 5](#_Toc482204646)

[3. Document History 6](#_Toc482204647)

[4. FPVL API 7](#_Toc482204648)

[4.1. General description of FPVL API 7](#_Toc482204649)

[4.2. API Classes and their Methods 7](#_Toc482204650)

[5. Integration Example of FPVL 10](#_Toc482204651)

[6. Reference 12](#_Toc482204652)

1. Purpose

This document is the integration guide for the Finger Print Verification Library (FPVL) for the Retail Phase II project. It contains API description and the example for integration of FPVL into other systems.

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** |
| March 20, 2017 | 1.0 | Document created |
|  |  |  |

1. FPVL API
   1. General description of FPVL API

FPVL API contains the common structures and classes for verification of fingerprint maps. The API is realized in namespace FPVerificator.

FPVL implements the following functionality:

* Creating grid-map positions with needed interpolation interval
* Calculating number of magnetic measurements for every grid-map position
* Calculating number of WiFi measurements for every grid-map position
* Calculating number of BLE measurements for every grid-map position
* Calculating magnetic coverage for the magnetic grid
* Calculating WiFi coverage for the WiFi grid
* Calculating BLE coverage for the BLE grid

Additionally, FPVL API utilizes data structures defined in FPBL, such as MagneticGrid, WiFiGrid, BlecGrid and Venue.

Also FPVL utilizes CoordinateConverter component.

* 1. API Classes and their Methods

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

**OccupancyOfPosition.hpp**

OccupancyOfPosition is used for saving one IRL position and number of measurements for it

struct OccupancyOfPosition

{

float X;

float Y;

int32\_t number\_of\_measurements;

OccupancyOfPosition () :X(0), Y(0), number\_of\_measurements(0){}

};

inline bool compare\_occupancy\_of\_positions\_by\_x (const OccupancyOfPosition & pos1, const OccupancyOfPosition & pos2)

{

return (pos1.X < pos2.X);

}

inline bool compare\_occupancy\_of\_positions\_by\_number\_of\_measurements (const OccupancyOfPosition & pos1, const OccupancyOfPosition & pos2)

{

return (pos1.number\_of\_measurements < pos2.number\_of\_measurements);

}

**CreateIRLPositions.hpp**

create\_irl\_positions () function create IRL positions list

**CalculateNumberOfMagMeasurements.hpp**

calculate\_number\_of\_mag\_measurements () function calculate number of magnetic measurements for every IRL position

void create\_irl\_positions(

// input

Venue &venue, // venue struct with building parameters

char\* pBuffer, // buffer with IRL data

uint32\_t buffer\_size, // buffer size

double interpolation\_interval, // interval of interpolation for IRL positions

// output

std::vector<OccupancyOfPosition> &irlposlist // IRL positions with number of measurements

);

void calculate\_number\_of\_mag\_measurements (

// input

Fpbl::MagneticGrid &magneticGrid, // grid with magnetic data

double mag\_grid\_size, // size of magnetic cell

// input / output

std::vector<PositionOccupancy> &irlpositions // IRL positions with number of measurements

);

**CalculateNumberOfWiFiMeasurements.hpp**

calculate\_number\_of\_wifi\_measurements () function calculate number of WiFi measurements for every position

void calculate\_number\_of\_wifi\_measurements (

// input

Fpbl::WiFiGrid &wifiGrid, // grid with WiFi data

double wifi\_grid\_size, // size of WiFi cell

// input / output

std::vector<PositionOccupancy> &irlpositions // IRL positions with number of measuments

);

**CalculateNumberOfBleMeasurements.hpp**

calculate\_number\_of\_ble\_measurements () function calculate number of BLE measurements for every position

void calculate\_number\_of\_ble\_measurements (

// input

Fpbl::BleGrid &bleGrid, // grid with BLE data

double ble\_grid\_size, // size of WiFi cell

// input . output

std::vector<PositionOccupancy> &irlpositions // IRL positions with number of measurements

);

**CalculateCoverage.hpp**

combine\_immmapgrid\_with\_blegrid () function calculate coverage for any IRL positions list

double calculate\_coverage(

std::vector<PositionOccupancy> poslist, // IRL positions with number of measurements

int32\_t min\_meas\_num // minimal number of measurements

);

The source code of FPVL is available at Gift repository on GitHub by the following link: [Gift](https://github.com/InvenSenseInc/Gift)/[Tools](https://github.com/InvenSenseInc/Gift/tree/master/Applications)/ FP\_Verificator/Src/Lib.

Visual Studio project for Windows and makefiles for Linux of FPVL is available at Gift repository on GitHub by the following link: [Gift](https://github.com/InvenSenseInc/Gift)/[Tools](https://github.com/InvenSenseInc/Gift/tree/master/Applications)/ FP\_Verificator/Src/Project.

1. Integration Example of FPVL

The source code of the PC console application FP Verificator using FPVL is available at Gift repository on GitHub by the following link: [Gift](https://github.com/InvenSenseInc/Gift)/[Tools](https://github.com/InvenSenseInc/Gift/tree/master/Applications)/ FP\_Verificator/Src/App.

Visual Studio project and solution for Windows and makefiles for Linux of FP Verificator application are available at Gift repository on GitHub by the following link: [Gift](https://github.com/InvenSenseInc/Gift)/[Tools](https://github.com/InvenSenseInc/Gift/tree/master/Applications)/ FP\_Verificator/Src/Project.

The FP Verificator application loads IRL data from an input binary file, loads magnetic, WiFi, BLE input grids.

char \*pBuffer = NULL;

uint32\_t fileSize;

// loading IRL data

load\_irl\_map\_grid\_file(immMapGridFileName, &pBuffer, fileSize);

std::vector<OccupancyOfPosition> positions;

positions.clear();

// create IRL positions list

create\_irl\_positions(venue, pBuffer, fileSize,

mag\_interpolation\_interval, positions);

// magnetic

{

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

// loading magnetic grid

load\_mag\_grid\_file(magGridFileName, magneticGrid);

}

// WiFi

{

Fpbl::WiFiGrid wifiGrid; // created once, then updated for each track

// loading WiFi grid

load\_wifi\_grid\_file(wifiGridFileName, wifiGrid);

}

// BLE

{

Fpbl::BleGrid bleGrid; // created once, then updated for each track

// loading BLE grid

load\_ble\_grid\_file(bleGridFileName, bleGrid);

}

Then the application calls functions for calculating number of measurements, saving those into a file and calculating coverage.

// magnetic

{

calculate\_number\_of\_mag\_measurements(magneticGrid, mag\_grid\_size, positions);

int32\_t min\_mag\_meas\_num = 300;

double coverage = calculate\_coverage(positions, min\_mag\_measurements);

std::cout << "magnetic coverage = " << coverage << std::endl;

save\_position\_occupancy(out\_log\_folder + SLASH + "occupancy.mag", maglist);

}

// WiFi

{

calculate\_number\_of\_wifi\_measurements(wifiGrid, wifi\_grid\_size, positions);

int32\_t min\_wifi\_meas\_num = 30;

double coverage = calculate\_coverage(positions, min\_wifi\_measurements);

std::cout << "wifi coverage = " <<coverage << std::endl;

save\_position\_occupancy(out\_log\_folder + SLASH + "occupancy.wifi", wifilist);

}

// BLE

{

calculate\_number\_of\_ble\_measurements(bleGrid, ble\_grid\_size, positions);

int32\_t min\_ble\_meas\_num = 5;

double coverage = calculate\_coverage(positions, min\_ble\_measurements);

std::cout << "ble coverage = " << coverage << std::endl;

save\_position\_occupancy(out\_log\_folder + SLASH + "occupancy.ble", blelist);

}

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