Pervasive Positioning Standard for Fingerprint-based and Proximity-based Systems

for

Site Owners and Application Developers

Prepared by Gary W.-H. Cheung, Peter W.-L. Tsui, Mengyun Liu
Supervised by S.-H. Gary Chan

HKUST

March 15, 2023

Contents

Abstract

Pervasive positioning is to locate an object anywhere seamlessly on a country scale. Current positioning technologies are mature enough to support both indoor and outdoor environments, using site signal survey and indoor localization algorithms for indoor and GNSS for outdoor. Pervasive positioning can be realized provided that application requests location services from the correct parties using the correct formats. Pervasive Positioning Standard aims to bridge every party together. This standard specifies a set of communication protocols, data organization and format definitions for site signals, and data organization for maps using existing map standards. With this standard, existing location-based service (LBS) applications are able to operate anywhere in Hong Kong including their original supporting zone, and many novel LBS applications can be developed.

Introduction

The goal of the Pervasive Positioning Standard is to enable any applications to locate their users anywhere so as to provide LBS potentially on a country scale. It is important to be able to locate users in both outdoor and indoor environments. In outdoor open areas, localization can be done with GNSS using satellite signals. In outdoor and indoor enclosed environments, we carry out site surveys and use the site signals to locate users. However, there are no agreed formats of site signals, so applications cannot share their site signals with others easily to enlarge their supported zone. In addition, pervasive positioning on a country scale requires the management of a huge size of site signal and map data. This standard aims to provide communication protocols, data formats, and organization required for an application to locate a user anywhere efficiently.

This standard consists of three components: 1) A set of communication protocols, 2) data organization and format definitions for site signals, and 3) data organization for maps using existing map standards.

Pervasive Positioning Standard embraces existing standards, if possible, to avoid repeating definitions. The standards that are covered are: IMDF, IndoorGML, and a few image formats (JPEG, PNG, and GIF).

1 Scope

Pervasive Positioning Standard is a set of communication protocols with data organization designs on site signals and maps. It aims to help LBS applications get locations anywhere by requesting location services from the correct service providers. This standard defines:

- Communication protocols
  - SDK APIs for applications to request location service
  - Web APIs for site owners to host a server for service query
- Data organization and format definitions for site signals
  - A grid reference system for organizing site signals
  - Format definition of reference point for fingerprint-based system
  - Format definition of anchor of proximity-based system
- Data organization for maps
  - A multi-layered space model for organizing maps

This standard is for fingerprint-based and proximity-based localization only.

2 Terminologies

2.1 Data

Site signal data

Site signal data is collected from site survey. It is precomputed data that is used in indoor localization systems, for example, WIFI fingerprint, iBeacon location, etc. This standard is for fingerprint-based and proximity-based localization only.

Map data

Map data is used to display the environment, for example, a JPEG file, an IMDF archive, etc. This standard only supports image format, IMDF, and IndoorGML.

Site spatial data

Site signal and map data, collectively called site spatial data.

User signal

User signals are the surrounding sensor measurements of the user. The format is defined in this standard.

2.2 Stakeholders

End users (Applications)

Applications use this standard to provide pervasive positioning for end users. End users either upload user signals or download site signals according to their preference.

Site owners

Site owners are the data holders of the site spatial data in his/ her site(s). They either send site signals to the lookup server or host a server for location service query.
Site owners must upload map data to the lookup server.

Platform operator (Lookup server)

Lookup server is the contact point that applications connect for location services such as computing locations and transmitting site spatial data. It stores site signals and maps from site owners.

2.3 Developers

Application developers

Application developers who develop the LBS application using this standard to obtain locations.

Site owner's developers

Developers who build and set up servers for site owners. According to whether site owners are willing to share site signals, site owners may or may not hire site owner's developers to set up servers.

3 Conventions

Symbols and abbreviated terms

API	Application Programming Interface
IndoorGML	Indoor Geographic Markup Language
IMDF	Indoor Mapping Data Format
JSON	JavaScript Object Notation
JWT	JSON Web Token
LBS	Location-based Service
MLSM	Multi-Layered Space Model
NRG	Node-Relation Graph
n-D	n-Dimensional
RSSI	Received Signal Strength Indication
SDK	Software Development Kit

4 Motivations

Different applications have different methods to locate users in indoor environments, unlike GNSS for outdoor areas. Therefore, site signals are stored in different representations and styles. Let us consider when computing locations across two buildings A and B. Suppose building A uses WIFI fingerprints as site signals and JPEG images as maps, while building B uses iBeacon proximity and IMDF files. The current solution is to build two different applications, so that when you need a location, you open application A to locate yourself in building A and application B in building B. This case illustrates the inconvenience and redundancy to get a location pervasively at this moment, and it is far from the concept of pervasive positioning – to locate users in any applications.

But surprisingly, pervasive positioning can be realized with a simple concept: applications contact the right parties using the right formats. Continuing with the example, suppose building A supports edge computation only and building B supports cloud computation only. Contacting the right party means contacting the server for downloading building A's WIFI fingerprint as well as the server for computing locations in building B. Using the right formats means the data formats involved when contacting, for example, the WIFI fingerprint, user signal, location, etc.

Only two stakeholders, end users (applications) and site owners, involved in the communication cannot ensure that applications contact the right parties. In the first place, which building should be contacted when the application wants a location? Therefore, it is necessary to maintain a platform for retrieving the building location information and other useful data including metadata of site signals, maps that are needed to display the locations to users, etc.

Therefore, in order to bridge every stakeholder together to achieve pervasive positioning, this standard defines a set of communication protocols for applications, site owners and a lookup server (platform) as well as the spatial data formats and organization for running on a country scale.

5 Design and Approaches

5.1 Operation Modes and Communication Protocols

5.1.1 Overview

Communication protocols describe the necessary components for applications to acquire locations, that is, what should be done for all stakeholders. The following diagram shows a general workflow of the protocols:

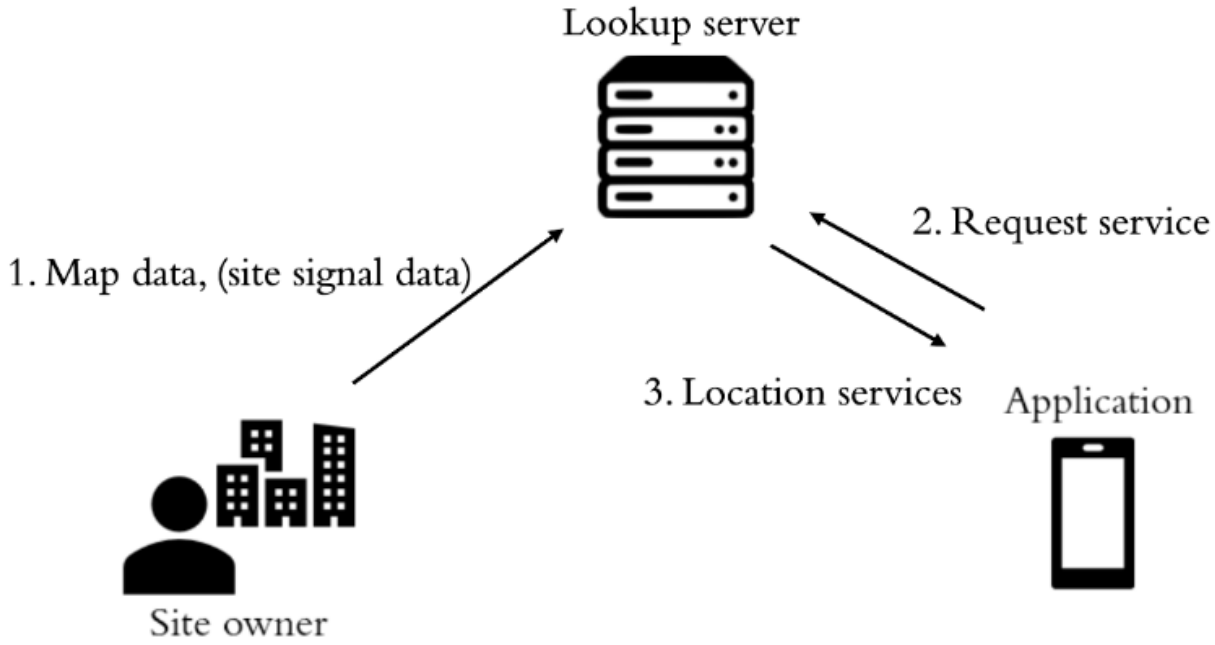

Figure 5.1: A general workflow of communication protocols

Site owners first upload map data and/or site signal data to lookup server. When applications request locations, they contact the lookup server, and the lookup server offers the location service to them, for example, sending nearby site signals to them or redirecting

them to site owner's server.

5.1.2 Four Operation Modes

End users and site owners may or may not want to share their data, which implies that only one operation mode is not enough. For example, a cloud approach is not applicable if end users do not share their user signals. Therefore, the communication protocols categorize all cases into four operation modes:

Table 5.1: Four operation modes for different pairs of applications and site owners

Application / Site owner	Not willing to share user signals with any server	Willing to share user signals with any server
Not willing to share site signals with the lookup server	Operation mode 0 [00] (Site-supported edge loc)	Operation mode 1 [01] (Site server localization)
Willing to share site signals with the lookup server	Operation mode 2 [10] (Platform-supported edge loc)	Operation mode 3 [11] (Platform localization)

Applications may or may not want to share end users' signal with any servers. This determines whether the operation mode is a cloud approach or an edge approach, since an edge approach must be chosen if applications do not share user. Besides, site owners also decide whether or not to share their site signals with the lookup server, determining if it is a centralized approach or a distributed approach. Therefore, depending on the types of applications and site owners, there are four operation modes.

The first bit of the binary form of the operation mode is whether the site owner is willing to share site signals with the lookup server while the second bit is whether the application is willing to share user signals with any server.

Operation mode 0 - Site-supported edge localization

If the site owner does not share site signals with the lookup server and the application does not share user signals with any parties, they should choose operation mode 0. The site owner has to host a server for retrieving location and tell the lookup server his/her

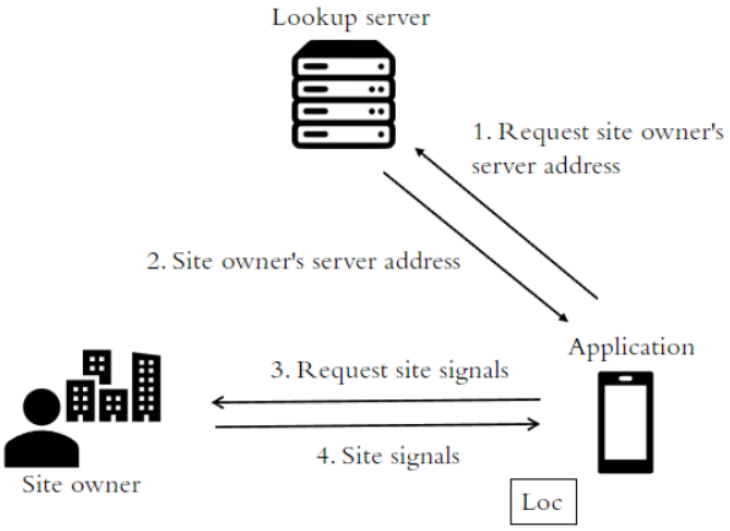

Figure 5.2: Operation mode 0

server address. Then the lookup server returns the server address upon the application's request, redirecting it to the site owner's server. After that, the application requests the site owner's site signals and computes the location locally.

Operation mode 1 - Site server localization

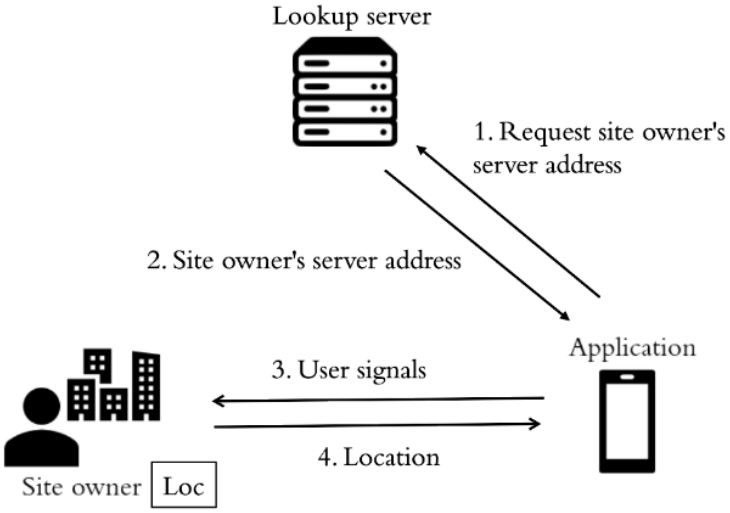

Figure 5.3: Operation mode 1

If the site owner does not share site signals with any parties but the application shares user signals, they should choose operation mode 1. This mode is similar to mode 0, but

after the application is redirected, it sends user signals to the site owner's server and receives the location.

Operation mode 2 – Platform-supported edge localization

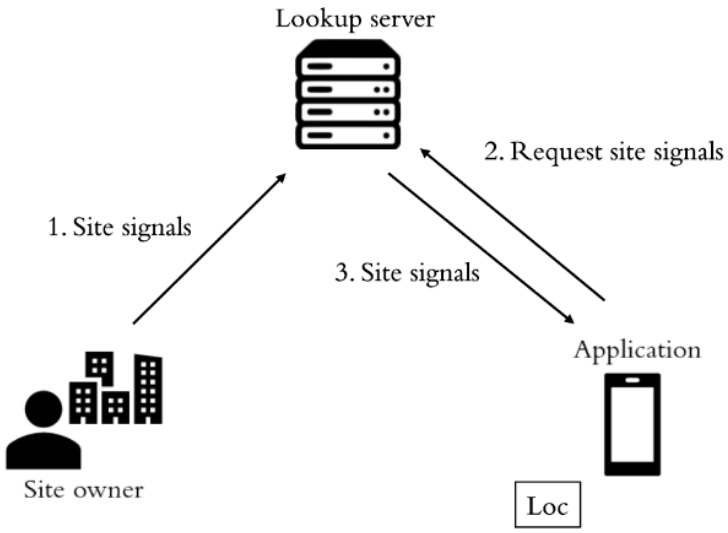

Figure 5.4: Operation mode 2

If the site owner shares site signals but the application does not share user signals, operation mode 2 is the best choice. In this mode, the site owner uploads his/her site signals to the lookup server. Then the application requests site signals from lookup server and computes location locally.

Operation mode 3 – Platform localization

If both the application and the site owner are willing to share their data, operation mode 3 is the best choice. In this mode, the site owner uploads his/her site signals to the lookup server. Then the lookup server computes the location for the application upon its request.

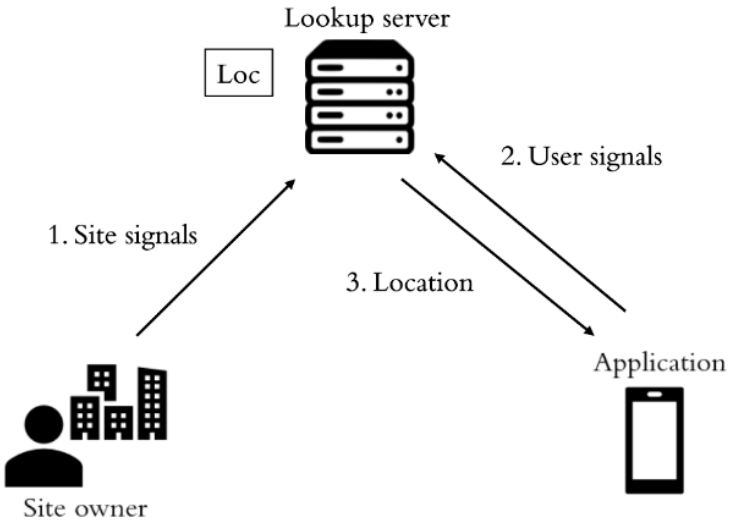

Figure 5.5: Operation mode 3

Whether or not to share data with others

For applications, sharing user signals weakens end users' location privacy and increases the latency since user signals will be uploaded to site owner's server to compute locations. However, if applications do not share user signals, it must compute the location locally, which increases the device computational and storage requirements.

For site owners, there are three levels of sharing. The first is not to share with any parties. Site owners should choose operation mode 1 so as to avoid any data privacy concerns. The second is to share with application only, potentially restricting the number of people accessing their site signals. Then they should choose mode 0. The third is to share with anyone, choosing mode 2 or 3.

However, there is a trade-off between privacy protection and cost since higher privacy protection means keeping more computation and information on their sides. Applications and site owners should choose the operation mode according to their preferences.

5.1.3 Handshaking for Choosing Operation Mode

It is important to provide the information needed for applications to choose the operation modes with the nearby building. Different applications may have different searching criteria and considerations. Therefore, a handshaking process is needed between applications and the lookup server to confirm the favored operation mode.

The handshaking consists of four steps:

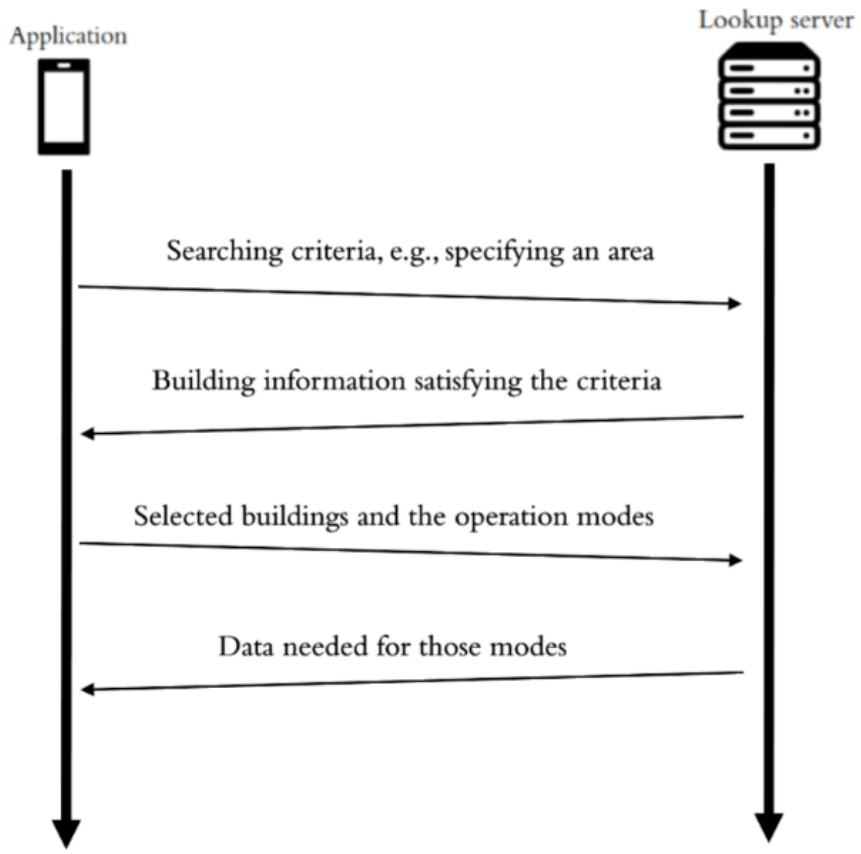

Figure 5.6: Handshaking between applications and the lookup server

1. The application specifies a (set of) searching criteria that it is interested in, for example, specifying a circle by a center and a radius, filtering out buildings not having WIFI fingerprints, etc.

2. The lookup server returns building information satisfying the criteria. This information includes everything the application needs to decide whether or not to select the operation mode for this building, for example, the supported operation modes, type of supported site signals, etc.

3. The application returns the selected building(s) and the operation mode(s).

4. After confirming the operation mode and the building, the lookup server sends the necessary data for operation to the application. For example, the application needs the site owner's server address to upload user signals for locations in operation mode 1.

5.1.4 Building Switching

Unlike traditional localization, pervasive positioning supports dynamically changing of the supported area. Therefore, before computing the location, an in-building region detection is carried out to ensure the application is inside the supported region. If the application is outside the supported region, the application needs to download the latest nearby site signals or connect to the site owner's server of the newly arrived building, which is called building switching.

The simplest way is to handshake again with the lookup server to obtain the latest data for operation. Applications are suggested to include the previous building identity in the searching criteria in step one when handshaking, which improves efficiency and accuracy of the newly arrived building.

5.1.5 Obtaining Map Data

After computing or receiving the locations, applications need a map to display the location. Since every map data is stored in the lookup server, the application can request the map data from the lookup server using the computed location. Other map retravel methods are introduced in Section 5.3.

5.1.6 Site Owners' Server

In mode 0 and 1, site owners do not share data with the lookup server so they should host their servers to deal with applications' requests. Mode 0 site owners need to host a server for downloading their site signals while mode 1 site owners need to host a server for computing location. The server should implement a specific list of APIs following this standard so as to ensure that the application can call the desired APIs in different buildings. The API specifications are in Chapter 8.

5.1.7 SDK for Applications

Applications need to contact many different servers to obtain the location services. If applications need to communicate with different parties via web APIs at a low level, it increases the developer's workload significantly. To facilitate the development of the application, this standard defines an SDK for retrieving location service to reduce the workload of application developers. Applications can handshake, download the site signals, connect to

site owner's servers, download map data via a set of APIs in the SDK. The SDK guidelines and API specifications are in Chapter 7.

5.1.8 A Holistic View of Communication Protocols

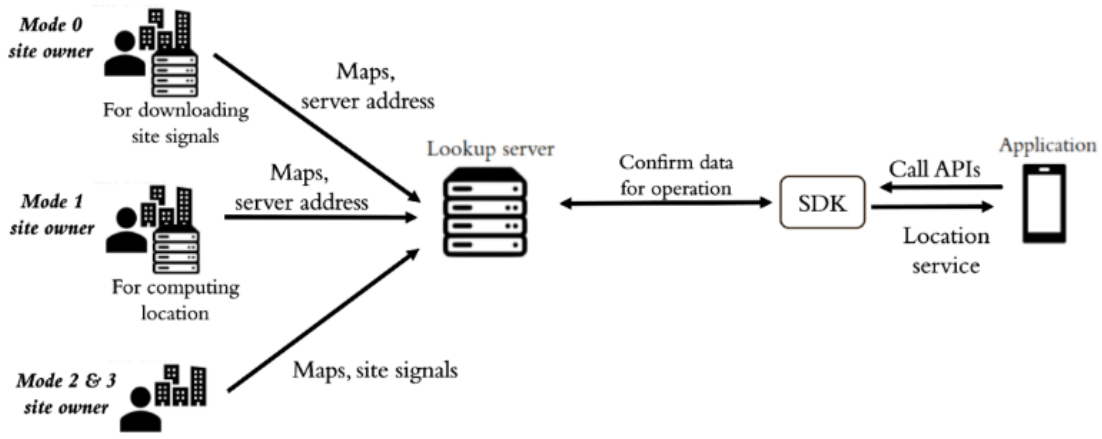

Figure 5.7: A holistic view of communication protocols

Before offering any location service, operation mode 2 and 3 site owners upload their site signals maps to the lookup server, while operation mode 0 and 1 site owners only upload maps to the lookup server, following the formats and procedures motioned in later sections. Additional works for mode 0 and 1 site owners is that mode 0 site owners host a server for querying their site signals and mode 1 site owners host a server for computing location. Then both of them send the server address to the lookup server.

The trigger point of a location service is when an application calls the SDK for location services. The SDK then contacts the lookup server to handshake to confirm data for operation, including the operation mode, the server address for uploading user signals as well as for requesting locations, the server address for downloading site signals, etc. After that, the SDK processes the data and provides the location services to the application.

5.1.9 Application Authentication and Authorization

As site owners may not want any application to access their server directly in mode 0 and 1, some authentication techniques are needed to ensure the call is from the standard SDK and the application is authorized. In this standard, JSON Web Token (JWT) [5] is used to provide both the authentication and authorization of each function call to the site owner's

server due to its simplicity and cross-domain functionality. A JWT, issued by the lookup server, is attached in each function call, so site owners can authenticate the user.

Each JWT contains the information of the issuer, the subject and expiration time, and it is digitally signed by the lookup server. Specifications of JWT claims can be found in Chapter 8.

Application developers do not need to worry about much the design but to register themselves in the lookup server and to obtain locations by procedures in following Chapter 7.

5.2 Site Signal Data Organization, Retrieval and Format

5.2.1 Overview

With communication protocols, applications can contact the right party to obtain the site signal and map data, but it is also important to know how to retrieve the data correctly and efficiently. That is, the data must follow an agreed data format so that every party can interpret the data correctly, and the data can be retrieved with flexibility to support efficient querying with a grid reference system.

5.2.2 Site Signal Formats

An agreed data format of site signals is needed to provide site signal data interoperability. Currently, there is no agreed data format of site signals so that site signals are represented in different measurements and styles by different organizations. For example, WIFI AP signal strength can be measured by RSSI, dBm or other units, and the WIFI fingerprints can be stored in a text file, a database, or other data storage systems. In this way, unless an organization shares its data formats, others cannot interpret their site signals correctly. Therefore, this standard defines the site signal data formats for WIFI fingerprint-based and proximity-based systems:

Site signals for fingerprint-based systems are stored as reference points.

In a fingerprint-based system, site signals are the fingerprints that comprise reference points. A reference point (RP) contains a location and a Received Signal Strength Indica-

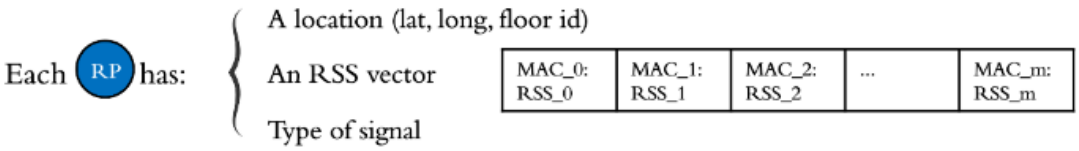

Figure 5.8: The structure of a reference point

tion (RSSI) vector that is collected in this location. A location is represented as a latitude, a longitude and a floor Id that represents the vertical information of the location. The floor Id refers to the floor Id in map organization and is defined in Section 5.3. A RSSI vector is a collection of the sensor's identifications and their strength measurement. It is stored as the MAC addresses of the sensors and the RSSI.

Site signals for proximity-based systems are stored as pairs of device Ids and locations.

In a proximity-based system, site signals are the locations of the sensors. Currently, this standard supports iBeacon only. Therefore, the beacon identification is stored as their UUID, major and minor, and the location is stored as a latitude, a longitude and a floor Id.

5.2.3 Grid Reference System

Inspired by Google map structure [14], a grid reference system is used as the data organization of site signals to support efficient querying. The major similarity of outdoor maps and site signals is that users tend to load a specific area initially and the adjacent later on. It is inefficient to load the site signals using the building as a unit, and a fine resolution using grids is the preferred way to load the site signals.

There are three enhancements to the grid reference system to adopt the nature of site signals:

1. The information of a grid is enriched since a grid does not only store a map and the coordinates. The types of supported site signals and the signal data are stored in the grid for site signals. Other location data structures, for example, the zoom level and grid length, remain unchanged.

2. Vertical information is also needed for site signals. This standard represents grids in

2.5-D. It is stored as the floor Id of the site signals. The floor Id refers to the floor Id in map organization and is defined in Section 5.3.

3. Adjacency is not the only connection between grids for site signals. Since users can travel a long distance between grids, for example, via an elevator an escalator. They can bring users from one grid to another grid that is far away, and the site signals between the two distant grids may not be useful. For example, if a user travels from 6F to GF via an elevator, the fingerprints of other floors are unnecessary. Therefore, this connection information is stored in a node-relation graph while grids are the nodes and a relation represents the connection between two grids.

Figure 5.9: A grid reference system storing the site signals of a building

5.2.4 Site Signal Data Retrieval

Site signals are stored in site owner's servers in operation mode 0 and 1 and in the lookup server in mode 2 and 3. They need to store the site signals in a grid reference system, following the data structure in section 6.1. With a common site signal data structure (in grids), applications can retrieve site signals flexibly and unambiguously in an edge approach. Grid is the basic unit of site signals, that is, applications retrieve data in grids from the lookup server and site owners' servers.

Site signals in the desired area can be retrieved by calling the predefined APIs. They provide data retrieval in a few different ways:

1. In a circle by specifying the center and the radius

2. In a rectangular region by specifying the maximum and minimum latitude and longitude

3. In a building by specifying the building Id, floor by floor Id and region by region Id

4. In the connected grids of a grid that is loaded previously

The resolution when retrieving the site signals is determined by the zoom level. The higher the zoom level, the finer grids will be retrieved. It is a tradeoff between efficiency and memory storage since if applications retrieve finer grids, they can have less memory occupied but they need to request more frequently when the users are moving.

The API specifications that the lookup server and site owners' servers should follow are in Chapter 8.

5.3 Map Data Organization and Retrieval

5.3.1 Overview

After applications compute or obtain a location, a map is needed for displaying the user location. Since every map data is uploaded to the lookup server. It is important to manage this large amount of data efficiently. In this standard, the lookup server stores the map data with a multi-layered space model while embracing existing map standards.

5.3.2 Existing Standards

Unlike site signals, there are a few mainstream indoor map standards. The following standards are supported:

1. Image formats including JPEG, PNG, and GIF. Images are well-known and common ways to store maps.

2. Indoor Mapping Data Format (IMDF) [11] is another popular map standard recently. It provides a general yet comprehensive model for indoor map data. It supports efficient and mobile-friendly computations in edge devices. In addition, the model can be used not only in map displaying but also other location-based applications such as navigation, orientation, etc.

3. GeoJSON multi-polygon type (https://datatracker.ietf.org/doc/html/rfc7946#section-3.1.7) provides a fraction of functionalities of IMDF. It is less costly than IMDF but it is also less expressive when representing the building structure. However, it can be a good balance for some companies.

4. IndoorGML [12] aims to provide comprehensive 3D indoor spaces modelling for navigation purposes. It is supported in this standard due to its potential usage in 3D LBS applications.

5.3.3 Multi-Layered Space Model

Inspired by IndoorGML [12], a multi-layered space model (MLSM) is used in order to manage complex indoor map data efficiently. A MLSM represents the same physical space in different representations. Each layer is the partitioning of spaces in one representation. In indoorGML, it is used to give the primal structure and different semantics, for example, primal space layer, topology space layer, WIFI coverage as a layer to record the spaces covered by each AP or a set of APs, etc. In this standard, the MLSM is designed in a way that suits our situation best.

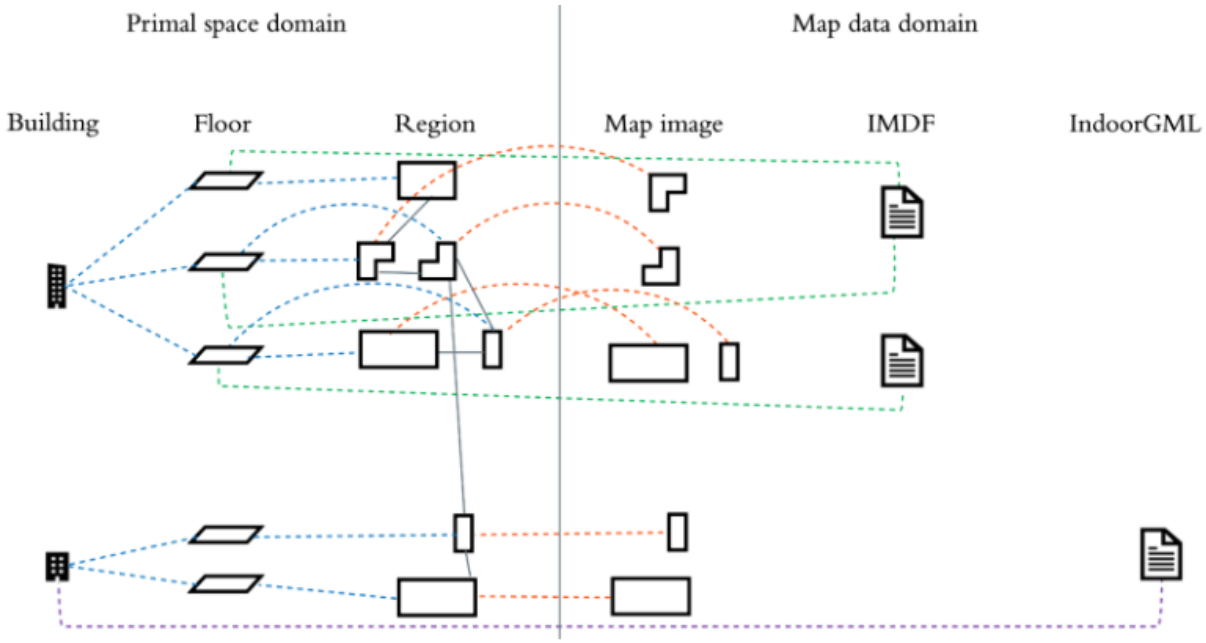

Figure 5.10: Multi-layered space model for managing map data

There are two domains in the MLSM, namely primal space domain and map data domain. Primal space domain represents the building hierarchical and topological information while map data domain represents the details of the map data.

Primal space domain consists of three layers, namely building layer, floor layer and region layer. Inter-layer connections (ILCs) represent the hierarchical structure of the building. For example, an ILC between a building and a floor means the floor belongs to

the building (the blue dotted lines in Fig. 5.10). Topological information is stored in the region layer for the finest resolution, shown as the gray lines in Fig. 5.10.

Map data also consists of three layers, namely map image layer, IMDF layer and IndoorGML layer. Each layer stores the data of its type and the attached space of the data. Each map data node contains the data that is necessary for display such as geodetic points and boundary. The ILCs between two domains are the attachment of map data and its primal space. For example, the indoorGML file in Fig. 5.10 is attached to the lower building, represented by the purple dotted line.

Site owners need to provide their maps, the hierarchical and topological information of their building to the lookup server, following the validation check in Chapter 9. Then the MLSM in the lookup server will store them and will be used to deal with applications' map requests.

5.3.4 Map Data Retrieval

With the MLSM, map data retrieval can be very flexible. There are three approaches to obtain maps:

1. By coordinates and constraints. Inputs specify a region, and the map data inside the region is returned. For example, given a location (latitude, longitude) and a radius, the map data inside that circle is returned.

2. By building, floor or region identity. Input is the Id of a specific building/ floor/ region, and the map data that is attached to this space and its children is returned. For example, given a floor Id, the map data of this floor and the child regions is returned.

3. By connections. Inputs are a source map Id and other requirements such as a direction, and the map data that is connected in the region layer is returned. For example, given a map Id, the map data that its attached space is connected to the attached space of the given map Id is returned.

More detailed descriptions and specifications are in Chapter 7.

6 Data Structures and Formats

Every party has to follow the same data formats to store, send and receive site spatial data in order to ensure every party can retrieve and interpret the desired data correctly. In this standard, we define:

1. A set of site signal data format and a grid data structure for site signal organization
2. A multi-layered space model for map data organization

Noted that this standard does not define the storing methods for the data but the data structure since different parties may have different preference of their storage system such as DBMS or simple text files. The detailed implementation for each party is described in later sections.

6.1 Site Signal Data

In this section, we first define a set of site signal data formats for different types of site signal, then we design a grid data structure to store site signals for efficient querying.

6.1.1 Site Signal Definition for Fingerprint-based System

In a fingerprint-based system, site signals are the fingerprints that comprise reference points with the measurements of surrounding sensors. A reference point defined as follows:

Table 6.1: Site signal definition for fingerprint-based system

Attribute	Data type	Description
RP Id	UUID	The Id of the reference point
Latitude	Number	The latitude of the reference point in 6 decimal places
Longitude	Number	The longitude of the reference point in 6 decimal places
Floor Id	String	The floor Id of the reference point, referencing the Id in map data structure
WIFI RSS vector	Array	The WIFI RSS vector collected in the reference point. Each element is a "mac:rssi" string, representing the mac address and rssi value of each AP. It is null if the WIFI RSS vector is not supported.
BLE RSS vector	Array	The BLE RSS vector collected in the reference point. Each element is a "uuid:major:minor:rssi" string, representing uuid, major, minor, and rssi of each beacon. It is null if the BLE RSS vector is not supported.
Magnetic signal	Array of number	An array of 3 elements that are the geomagnetic field strength along the x-, y-, and z-axis in μT.

6.1.2 Site Signal Definition for Proximity-based and range-based System

In proximity-based and range-based systems, site signals are the locations of the sensors. For BLE, each iBeacon is defined as follows:

Table 6.2: Site signal definition for proximity-based and range-based system

Attribute	Data type	Description
Beacon Id	String	The Id of this iBeacon in this standard. The format is: UUID (in hex string form) concatenates with Major and Minor (in 5 digits)
Latitude	Number	The latitude of the reference point in 6 decimal places
Longitude	Number	The longitude of the reference point in 6 decimal places
Floor Id/ Outdoor Site Id	String	In indoor environments, the floor Id should be given. It is the floor of the beacon's location, referencing the Id in the map data structure In outdoor environments, the site Id should be given, referencing to the site Id in. Chapter 9. It is the site that the beacon is in.
UUID	String	The UUID of the beacon in the form 8-4-4-4-12
Major	Number	The major value of the beacon, ranging from 1 to 65535
Minor	Number	The minor value of the beacon, ranging from 1 to 65535

6.1.3 Grid Reference System

Site signals are stored in a grid reference system for efficient retrieval of site signals. Each grid contains some site signals and topological information. A grid is defined as follows:

Table 6.3: Data structure of a grid

Attribute	Data type	Description
Grid Id	String	The Id of the grid. The format is the concatenation of zoom level, x-index, y-index and floor Id. The Grid Id for indoor site will consist of 37 digits: zoom level (2) + x-index (7) + y-index (7) + buildingId (19) + floor number (2). The Grid Id for outdoor site will consist of 35 digits: zoom level (2) + x-index (7) + y-index (7) + siteId (19). It is an extension to the Google Grid ID system.
Zoom level	Number	The zoom level of the grid, ranging from 16 to 20
X-index	Number	The x-index of the grid in this zoom level
Y-index	Number	The y-index of the grid in this zoom level
Floor Id	String	The floor Id of the beacon's location, referencing the Id in map data structure. It is empty for outdoor grids.
Connected grid Ids	Array	The array of connected grid Ids
RP Id list	Array	The array of RP Id that the reference points are in this grid
Beacon Id list	Array	The array of RP Id that the reference points are in this grid

Zoom level describes the size of a grid

The higher zoom level, the grid is smaller. Each level n grid is composed of 4 level n+1 grids. At zoom level 0, there is only one grid representing the whole world. At zoom level 1, 2x2 grids cover the world. At zoom level 2, there are 4x4 grids, and so on.

X- and y-index are the column/row number of a grid

X-index is the column number counting in ascending order of longitude. Y-index is the row number counting in ascending order of latitude. Both indexes range from 0 to $2^{(zoomlevel)-1}$. The index is a 7-digit number.

Connected grid Ids are the grids that can be directly arrived from the current grid

For example, if an escalator is from grid A to grid B, then grid B's Id is in grid A's connected grid ids. If an elevator connects grids C, D, E, then they are in each other's connected grid Ids.

6.1.4 Equations Connecting Latitude, Longitude, and Grid Reference System

Latitude ranges from –90 to 90 and relates to y-index. Longitude ranges from –180 to 180 and relates to x-index.

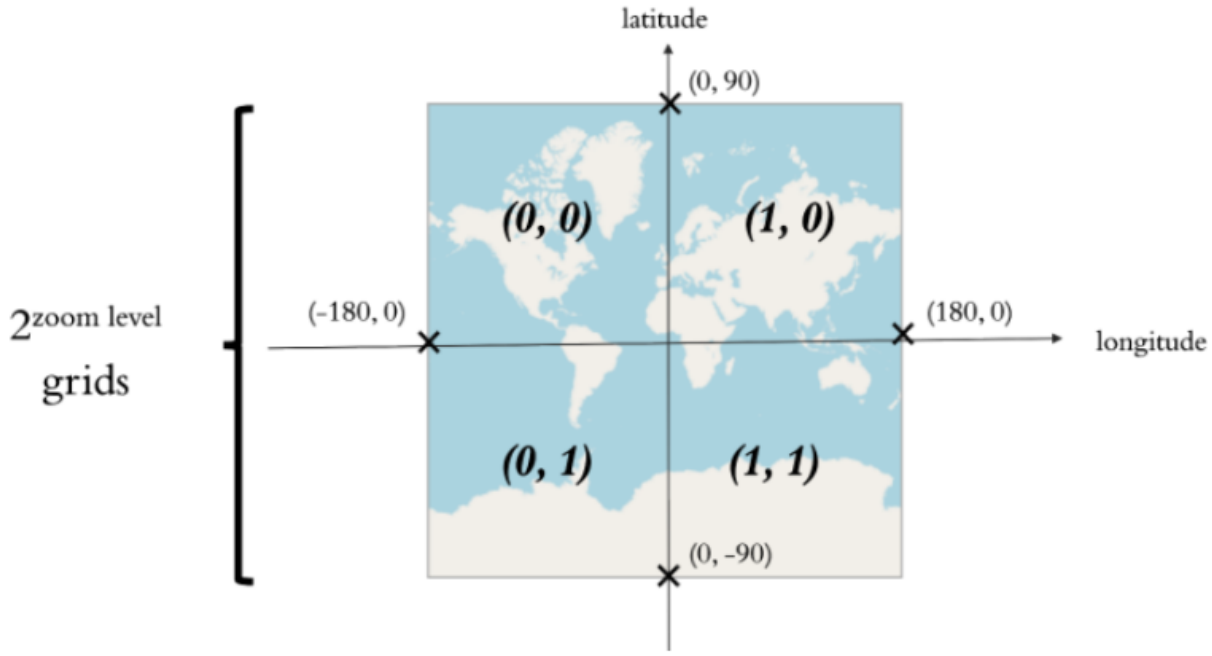

Figure 6.1: Grids at zoom level 1

This is the grids at zoom level 1. Besides the latitude and the longitude, the number of grids in a row or in a column is $2^{zoomlevel}$.

Computing corner coordinates of a grid given x-, y-index and the zoom level z

Using Mercator projection formula, we come up with 4 equations:

$$minLon = x \times grid_{width} - 180$$

$$= \frac{360x}{2^z} - 180;$$

$$maxLon = (x+1) \times grid_{width} - 180$$

$$= \frac{360(x+1)}{2^z} - 180;$$

$$minLat = \frac{180}{\pi} \times \arctan(\sinh(\pi - \frac{y+1}{2^z} \times 2\pi))$$

and

$$maxLat = \frac{180}{\pi} \times \arctan(\sinh(\pi - \frac{y}{2^z} \times 2\pi))$$

Computing x-, y-index given latitude and longitude at zoom level z

Using Mercator projection formula, we come up with 2 equations:

$$x = \left\lfloor \frac{2^z(lon + 180)}{360} \right\rfloor$$

and

$$y = \left\lfloor (1 - \frac{\ln(\tan(lat \times \frac{\pi}{180}) + \frac{1}{\cos(lat \times \frac{\pi}{180})})}{\pi}) \times 2^{z-1} \right\rfloor.$$

Computing x-, y-indexes of the next zoom level

A grid at zoom level z contains 4 grids at zoom level (z+1). Their indexes are: (2x, 2y), (2x, 2y+1), (2x+1, 2y), (2x+1, 2y+1)

6.2 Map Data

Recall that this standard does not define new map standards but embraces the existing map standards, such as images, IndoorGML, and IMDF, providing a multi-layered space

model for flexible and efficient map data retrieval from applications.

6.2.1 A Multi-layered Space Model

There are two domains in this model, namely primal space domain and map data domain. The former describes the physical space hierarchical and topological information, while the latter describes the map data. Note that the connections (edges) are stored in the nodes like adjacency list. Therefore, nodes under each layer contain the layer as well as the connection information.

Primal space consists of three layers – building, floor, region

The building layer contains hierarchical information about the building. It is defined as follows:

Table 6.4: Data structure of the building layer

Attribute	Data type	Description
Building Id	Hex string	The Building Common Spatial Unit Identifier (CSUID) of the building. Building CSUID consists of geo-reference number, polygon type and creation date. Geo-reference number: a 10-digit identifier formed by combining the Easting and Northing of the building label point within the polygon. (Easting and Northing are from HK 1980 Grid Coordinates, decimal is truncated and the first digit is removed from the coordinates.) Polygon type: 'T' or 'P'. T for Tower, P for Podium. Creation Date: YYYYMMDD e.g. BuildingId: "4520522021T20220412"
Name	String	The display name of the building
Child Ids	Array	The array of child floor Ids, referencing floor Id in the floor layer. It is sorted by elevation ascendingly.
Default floor Id	Hex string	The default floor of the building to be displayed
Data file Ids	Array	The array of data file Ids that are attached in the building, referencing the data file Id in map data domain

The floor layer contains hierarchical information about the floor. It is defined as follows:

Table 6.5: Data structure of the floor layer

Attribute	Data type	Description
Floor Id	Hex string	Building Id concatenates with the floor number
Floor number	Hex string	An 8-bit hex string (2 letters) indicating the floor number. The first bit is 0 if it is above ground, 1 if lower ground. The remaining 7 bits are the magnitude of the floor, relative to the default floor of the building. For example, if the default floor is G/F: $01 = 00000001_{(2)}$ is the 1/F, and $84 = 10000100_{(2)}$ is the LG 4/F Remarks: 00000000 is G/F while 10000000 is invalid
Name	String	The display name of the floor excluding the building name.
Parent Id	Hex string	The parent building Id
Child Ids	Array	The array of region Ids, referencing region Id in the region later
Default region Id	Hex string	The default region of the floor to be displayed
Data file Ids	Array	The array of data file Ids that are attached in the building, referencing the data file Id in map data domain

The region layer contains hierarchical and topological information about the region. It is defined as follows:

Table 6.6: Data structure of the region layer

Attribute	Data type	Description
Region Id	Hex string	Floor Id concatenates with the region number
Region number	Hex string	An 8-bit hex string (2 letters) indicating the region number
Name	String	The display name of the region excluding the floor name
Parent Id	Hex string	The parent floor Id
Data file Ids	Array	The array of data file Ids that are attached in the building, referencing the data file Id in map data domain
Connected regions	Array	This describes the transition regions and their arrival regions of this region in this form: [transition region: [arrival $region_1$, ...]] A transition region is described as [minLon, minLat, maxLon, maxLat] An arrival region is described as regionId: id, area: [minLon, minLat, maxLon, maxLat]

Map data domain stores each type of map in one layer

Currently, this standard only supports image, IMDF and IndoorGML, and they are stored in the same structure. It is defined as follows:

Table 6.7: Data structure of the map domain

Attribute	Data type	Description
Data Id	UUID	The Id of the map
Format	String	The format of the map, e.g., jpg, png, indoorgml, imdf, etc.
Geodetic points	Array	The array of geodetic points. Each geodetic point contains a coordinate in the map and a [lon,lat] pair. The array contains at least 2 geodetic points. Both the latitude and longitude are in 6 decimal places
Boundary	Array	The array of [lon,lat] of the boundary of the image
Attached primal space	Hex string	The attached primal space Id. It can be a building Id, floor Id and region Id.
Data	Byte	The data content of the image

7 SDK API Specifications for Mobile Application Developers

This section is written for mobile developers who want to implement pervasive positioning in their apps. To start off, developers should read the "Guideline for implementing pervasive positioning" to get an overview of SDK usage and read the "Detail specification of SDK" to understand classes and API interface in SDK for your application development.

7.1 Guideline for Implementing Pervasive Positioning

7.1.1 Initialization Step

The first step of pervasive positioning is to initialize the Global Positioning System to get an initial location. This GPS result will help your application to start handshaking with the pervasive positioning platform in the following steps.

Next, detect whether the user is under an indoor or outdoor environment. You can either:

1. use detectIndoorEnviroment(lat,lon,accuracy) in our Localization Assistant of SDK; or

2. implement your own indoor-outdoor environments detection algorithm.

This detection result will help your application to make decisions between using indoor positioning technology or using outdoor positioning technology.

7.1.2 Indoor Localization

If you want to implement indoor positioning, you should follow the steps below:

1. Detect which building you have entered
2. Decide whether to share user signals for localization
3. Get localization service from platform or site server
4. Compute or receive indoor location
5. Get map for display and navigation from platform
6. Check if the user entered switch zone periodically

1. Detect which building you have entered

This step helps you to discover the correct building from your GPS location or indoor location. Each building will have their own settings on providing indoor localization, some may share a variety of site signals, and some may only provide their own cloud localization service for users. Discover the building where your device is inside will help the SDK retrieve correct settings for your indoor localization.

Developers should call discoverBuilding(latitude, longitude, accuracy) in our API Manager of SDK. SDK will contact the platform to download building's localization settings and cache them in API Manager for further usage. Remember to keep the return value of discoverBuilding(latitude, longitude, accuracy), you will use it in the next step.

2. Decide whether to share user signals for localization

After you discover the current building, you will need to decide which localization approach you want to use in this building. Localization approach support by the current building is indicated by the returned string of discoverBuilding(latitude, longitude, accuracy), "cloud", "edge" or "all_available". Developer can adapt supported approach to get localization service.

This decision should be made according to your application's security concerns. As adapting cloud approach requires you to share user signals with cloud server, it will reveal user's location to the service provider. Developers who wish to keep high location security should priorities edge approach rather than cloud approach. During implementation, we recommend developers to implement your localization in a prioritize logic to support both edge localization and cloud localization. You can consider to only adapt one approach

in your app, but remember you may not be able to support buildings who only support another approach.

No matter what approaches you choose, we must acquire an authentication token through generateToken(appID,key) first. This token will grant your access to location service provided by platform and site servers. If your appID and key is authorized by the platform, a token will be generated and keep in API Manager for getting location services.

If you decide to implement edge localization and try to get a site signal package, go to 3a. For implementation of cloud approach, go to 3b.

3. Get localization service from platform or site server

Edge approach This step helps you to initialize indoor edge localization in the current building. First, call getGridIDListForEdgeLoc() API to get the gridID list by parameters. GridID list describes areas you want to download site signal. Then, call downloadSiteSignal() API to retrieve site signals data from the platform or site owner's server.

Cloud approach This step helps to you initialize indoor cloud localization in the current building. First, call getSignalTypeForCloudLoc() API to get the required signal type of cloud localization algorithm. You will need to scan these types of signals for cloud localization. Then, call uploadSignalToCloud() API to upload user ID and user signals to the platform or site owner's server.

4. Compute or receive indoor location

For edge approach, you can use the return site signals from API in your indoor edge localization algorithm, and compute a user's location.

For cloud approach, you can call getCloudLocResult() to download user location computed by cloud server.

5. Get map for display and navigation from platform

After you have gathered a user's location, you need to download a map to display the location on your application.c API to search all available maps for your location or desired floor. You should choose a mapID that matches your desired level of spatial representation and your displayable filetype. After you get your desired mapID and map filetype, use

them in getMapFile(filetype,mapID) to download actual map file. Remember to deserialize returned byte array to actual map file.

6. Detect switch condition periodically for switching localization services

In the indoor positioning, users may walk across multiple buildings or go to an outdoor environment. You will need to switch to indoor localization service of another building, or switch to outdoor localization service. Therefore, you have to implement a switch zone detect mechanism. SDK provided a reference design: detectSwitchCondition(location) API. This API is used to check if your indoor location falls inside a switch zone to another building or the outdoor environment. If the user is going into another building, the API returns a Building ID. If the user is going to an outdoor environment, the API returns an "Outdoor" tag for indication.

If switch building condition is detected, you will need to handshake with another building and use their localization service. You may go back to step 1 of indoor localization, and find building settings with connected Building ID to start again. You will need to add connected = True in some API call to indicate you wants to get localization service of connected building, so that localization service from current building won't be affect.

On the other hand, if switch outdoor condition is detected, you should switch to use outdoor localization service.

7.1.3 Outdoor localization

For outdoor localization, you can either use GPS results directly or implement your outdoor localization algorithm with outdoor signals.

If you want to get outdoor signals, such as Smart Lampposts, you should first call discoverOutdoorSite(latitude,longitude,radius) to find the correct outdoor site who have outdoor signals. Then call getOutdoorSignal(latitude,longitude,radius,siteSignalMode) to get nearby outdoor signal information from the Pervasive Positioning Platform or Site owner server.

7.2 Detailed Specifications

SDK will be responsible for handling handshaking protocol and acquiring necessary information for pervasive localization from the platform.

SDK will consist of three parts:

Data Classes	Classes for representing data in handshaking. Spatial representation, signal grids, raw signals, etc.
API Manager	Handling handshaking API calls, retrieving necessary data from platform database, returning Data class object to the caller
Localization Assistant	Reference design algorithms for pervasive localization (in-outdoor detection, switch condition detection)

7.2.1 Data Classes

Classes for handling API response data, such as Building, Floor, Region in Spatial representation, Building Signal Object, Grid Object, raw signal and Location Object in site signal standard.

These data classes assist in the handshaking process and initialization of localization service, most of the API results will return as a Data class, so developers are recommended to use these classes for data handling as well.

For handshaking convenience, some of the data will be kept in the API Manager as current state information (BuildingID, GridID, connection information)

Data Classes for Spatial Representation

public abstract class **SpatialObj** extends **Object**		
Member	Type	Description
ID	String	ID of SpatialObj
name	String	Display name of SpatialObj
mapDataID	List<String>	ID of map file that represents this SpatialObj
connectedList	List<Connection>	A list of connections represents connection area towards other SpatialObj

public class **BuildingObj** extends **SpatialObj**			
Inheritance	Member	Type	Description
Inherited member	ID	String	ID of building, CSUID of building
	name	String	Display name of building
	mapDataID	List<String>	ID of map file that represents this building
	connectedList	List<Connection>	A list of connections represents connection area towards other BuildingObj
	floorList	List<FloorObj>	A list of floors contained by this BuildingObj
	defaultFloorID	String	ID of FloorObj that is located on ground level

public class **FloorObj** extends **SpatialObj**			
Inheritance	Member	Type	Description
Inherited member	ID	String	ID of floor
	name	String	Display name of floor
	mapDataID	List<String>	ID of map file that represents this floor
	connectedList	List<Connection>	A list of connections represents connection area towards other FloorObj
	floorNo	String	An 8-bit hex string (2 letters) indicating the floor number, referencing to Section 6.2
	regionList	List<RegionObj>	A list of RegionObj contained by this FloorObj
	defaultRegionID	String	ID of RegionObj that acts as default region

public class **RegionObj** extends **SpatialObj**			
Inheritance	Member	Type	Description
Inherited member	ID	String	ID of region
	name	String	Display name of region
	mapDataID	List<String>	ID of map file that represents this region
	connectedList	List<Connection>	A list of connections represents connection area towards other RegionObj
	contraints	Constraint	Constraints contained by this RegionObj, indicating the walkable area of this region to improve the localization accuracy.
	grid	List<String>	The array of 2D grid Ids in zoom level 20 occupied by this region. The specified area should be supported with location services.

public class **MapObj** extends **Object**		
Member	Type	Description
ID	String	ID of map
mapType	String	Type of map, PNG/JPEG/IMDF/IndoorGML
geodetic	List<GeodeticPoint>	A list of geodetic points that represent projection of map coordinate system to WGS84 coordinate system
Grid	List<String>	A list of grid Ids representing occupied grids of this map.
attachedPrimal SpaceID	String	Spatial ID of building/floor/region represented by this map
filename	String	Filename of actual map file
fileContent	byte[]	Byte array of map file content

Data Classes for Site Signal Standard

public class **BuildingLocSetting** extends **Object**		
Member	Type	Description
ID	String	ID of Building, same as BuildingObj
grids	List<String>	A list of 2d grid Ids representing the area supported by this building
supportedFloors	List<String>	A list of floor numbers indicating the floors supported with localization service.
operationMode	List<String>	A list of localization operation modes that are available in this building. Modes will be represented in [0,1,2,3]
siteSignalMode	List<String>	A list of site signals available in this building.
cloudLocSignalMode	List<String>	A list of user signals needed by the cloud localization service of this building
relatedGridList	List<String>	A list of Grid ID which related to this building
cloudLocSync	Boolean	A boolean descriptor to indicate what cloud localization approach will be used in this building. (only for Mode 1)
remoteSignalDownloadURL	URL	URL for download site signal from site owner's server
remoteCloudLocUploadURL	URL	URL for uploading users' signals (required by cloudLocSignal) to site owner's cloud localization service
remoteCloudLocDownloadURL	URL	URL for download user location from site owner's cloud localization service
remoteCloudLocSyncURL	URL	URL for computing the location synchronously.
remoteCloudSignalModeURL	URL	URL for getting signal mode supported from the site server.

public class **GridObj** extends **Object**		
Member	Type	Description
ID	String	ID of Grid
boundary	List<Point>	A list of (latitude, longitude) points representing boundary of grid
connectedGridID	List<String>	A list of grid id which connected to current grid in one hop

public class **SignalObj** extends **Object**		
Member	Type	Description
ID	String	ID of SignalObj
coordinate	Point	Reference point of SignalObj in (latitude, longitude)
floorID	String	ID of FloorObj where this SignalObj locate at

public class **WiFiFingerprintObj** extends **SignalObj**			
Inheritance	Member	Type	Description
Inherited member	ID	String	ID of WiFiFingerprintObj
	coordinate	Point	Reference point of WiFiFingerprintObj in (latitude, longitude)
	floorID	String	ID of FloorObj where this WiFiFingerprintObj locate at
	rssiVecList	List<Wifi>	A list of wifi rssi signal

public class **BLELocationObj** extends **SignalObj**			
Inheritance	Member	Type	Description
Inherited member	ID	String	ID of BLELocationObj
	coordinate	Point	Reference point of BLELocationObj in (latitude, longitude)
	floorID	String	ID of FloorObj where this BLELocationObj at
	UUID	String	UUID of BLE device
	major	String	Major of BLE device
	minor	String	Minor of BLE device
	txPower	Integer	Transmission power of BLE device

public class **MagFingerprintObj** extends **SignalObj**			
Inheritance	Member	Type	Description
Inherited member	ID	String	ID of MagFingerprintObj
	coordinate	Point	Reference point of MagFingerprintObj in (latitude, longitude)
	floorID	String	ID of FloorObj where this MagFingerprintObj locate at
	magneticVecList	List<Magnetic>	A list of magnetic signal

Data Class for Raw Signal

public class **WiFi** extends **Object**		
Member	Type	Description
mac	String	Mac address of received WiFi signal
rssi	Double	Receive signal strength indicator in dBm
freq	Integer	Frequency of received WiFi signal

public class **ReceivedWiFiSignal** extends **WiFi**			
Inheritance	Member	Type	Description
Inherited member	mac	String	Mac address of received WiFi signal
	rssi	Double	Receive signal strength indicator in dBm
	freq	Integer	Frequency of received WiFi signal
	timestamp	Long	A received Unix timestamp of WiFi signal

public class **BLE** extends **Object**		
Member	Type	Description
UUID	String	UUID of received BLE signal
major	String	Major of received BLE signal
minor	String	Minor of received BLE signal
rssi	Integer	Receive signal strength indicator in dBm
txPower	Integer	Transmission power level received in BLE signal

public class **ReceivedBLESignal** extends **BLE**			
Inheritance	Member	Type	Description
Inherited member	UUID	String	UUID of received BLE signal
	major	String	Major of received BLE signal
	minor	String	Minor of received BLE signal
	rssi	Integer	Receive signal strength indicator in dBm
	txPower	Integer	Transmission power level received in BLE signal
	timestamp	Long	A received Unix timestamp of BLE signal

public class **Magnetic** extends **Object**		
Member	Type	Description
mag_x	Double	Magnetic field reading of x-axis
mag_y	Double	Magnetic field reading of y-axis
mag_z	Double	Magnetic field reading of z-axis

public class **ReceivedMagneticSignal** extends **Magnetic**			
Inheritance	Member	Type	Description
Inherited member	mag_x	Double	Magnetic field reading of x-axis
	mag_y	Double	Magnetic field reading of y-axis
	mag_z	Double	Magnetic field reading of z-axis
	timestamp	Long	A received Unix timestamp of magnetic signal

Data Class for Coordinate Related Object & Point of Interest(PoI)

public class **Point** extends **Object**		
Member	Type	Description
lat	Double	Latitude of this point in WGS84 coordinate system
lon	Double	Longitude of this point in WGS84 coordinate system

public class **GeodeticPoint** extends **Point**			
Inheritance	Member	Type	Description
Inherited member	lat	Double	Latitude of this point in WGS84 coordinate system
	lon	Double	Longitude of this point in WGS84 coordinate system
	x	Double	x-axis coordinate of map coordinate system, which is different from WGS84
	y	Double	y-axis coordinate of map coordinate system, which is different from WGS84

public class **Location** extends **Point**			
Inheritance	Member	Type	Description
Inherited member	lat	Double	Latitude of this point in WGS84 coordinate system
	lon	Double	Longitude of this point in WGS84 coordinate system
	floorID	String	ID of the Floor where this location is at

public class **Connection** extends **Object**		
Member	Type	Description
transitionArea	RegionConnector	a region connector representing a transition area.
arrivalAreaList	List <RegionConnector>	Array of RegionConnector representing all connected areas that connects to current transition area.

public class **RegionConnector** extends **Object**		
Member	Type	Description
RegionID	String	ID of region where this region connector locate at.
Name	String	Name of connector, the naming convention is the concatenation of the type of connector and a number. (e.g. Lift 1, Escalator 10)
Tag	String	The tag is an optional attribute to briefly describe the nature of this connection for potential localization optimiza- tion. Available tags are "Elevator", "Escalator", "Stair", "Door", "Ramp", and "Other"
geometry	List <Point>	a list of [lon,lat] points to describe the poly-line of the transitional area.

public class **Constraint** extends **Object**		
Member	Type	Description
outConstraint	List<Point>	A polygon representing outermost constraint of the spatial object (building/floor/region).
inConstraints	List<List< Point >>	Array of polygon representing constraints of the spatial object (building/floor/region) located inside the outConstraint.

7.2.2 API Manager

API Manager is a singleton class to provide API call interface for developers. For any needs to communicate with the platform to retrieve data for localization, the developer should call this API Manager for solutions.

API List

Constructor

1. APIManager getInstance()

Initialize handshaking

1. HashMap<String, List<String>> discoverBuildingList(double latitude, double longitude, double accuracy)
2. String initializeBuildingObjInManager(String buildingID)
3. Boolean removeBuildingObjInManager(String buildingID)
4. Boolean generateToken(String appID, String key)

Indoor edge localization

1. List<String> getGridIDListForEdgeLoc(String buildingID, double latitude, double longitude, double radius)
2. List<String> getGridIDListForEdgeLoc(String buildingID, Location location)
3. List<String> getGridIDListForEdgeLoc(String buildingID)
4. JSONArray downloadSiteSignal(String buildingID, String siteSignalMode, List<String> gridIDList)

Indoor cloud localization

1. List<String> getSignalTypeForCloudLoc(String buildingID)
2. void uploadSignalToCloud(String buildingID, String userID, JSONArray userSignal)
3. Location getCloudLocResult(String buildingID, String userID)

Outdoor localization

1. List<String> discoverOutdoorSite(double latitude, double longitude, double accuracy)
2. Boolean initOutdoorSite(String outdoorSiteID)
3. Boolean removeOutdoorSite(String outdoorSiteID)
4. JSONArray getOutdoorSignal(String outdoorSiteID, double latitude, double longitude, double radius, String siteSignalMode)

Get map data for display

1. JSONObject getMapData(@Nullable String SpatialID)
2. JSONObject getMapData(Location location)
3. MapObj getMapObj(String mapID)
4. byte[] getMapFile(String fileType, String mapID)

Get building information

1. BuildingObj getBuildingObjByID(String BuildingID)
2. BuildingLocSetting getBuildingLocSettingByID(String BuildingID)
3. List<String> getSignalModeByID(String BuildingID)

API Specification

Constructor

APIManager getInstance()

Constructor of singleton class APIManager, any developer who wants to call API to get localization service should get instance by this function.

Input Parameter			
Parameter name	Data type	Mandatory	Description
-	-	-	-
Return value			
Data type	Mandatory	Description	
APIManager	Yes	Return static instance of APIManager.	

Initialize handshaking

HashMap<String, List<String>> discoverBuildingList(double latitude, double longitude, double accuracy)

This function is used for finding all buildings near the device. This attempt will calculate all the grids covered in the GPS location, and list out all the related building ID. For example, Building P,Q and R has 3 grids covered by the GPS location, the function will return a HashMap with buildingID as key and List of gridID as value

Input Parameter			
Parameter name	Data type	Mandatory	Description
latitude	double	Yes	Latitude of GPS result
longitude	double	Yes	Longitude of GPS result
accuracy	double	Yes	Accuracy of GPS result

Return value		
Data type	Mandatory	Description
HashMap<String, List<String>>	Yes	A HashMap to indicate the buildings which near the GPS location, the HashMap will have BuildingID as key and the corresponding gridID list as value

String initializeBuildingObjInManager(String BuildingID)

This function is used for initialize BuildingSpatialObj and BuildingLocSettingObj by the given BuildingID. The APIManager will call web API hosted on platform to retrieve spatial information and localization settings of the building, then store it in its class member HashMap¡String,BuildingObj¿ Buildings and HashMap¡String,BuildingLocSetting¿ BuildingLocSettings

Input Parameter			
Parameter name	Data type	Mandatory	Description
buildingID	String	Yes	ID of a Building, the format should conform Building CSUID

Return value		
Data type	Mandatory	Description
String	Yes	A string to indicate this building support which localization approach, return ”cloud”, ”edge” or ”all_available”

Boolean removeBuildingObjInManager(String BuildingID) This function is used for removing BuildingObj and BuildingLocSetting from APIManager			
Input Parameter			
Parameter name	Data type	Mandatory	Description
BuildingID	String	Yes	ID of a Building, the format should conform Building CSUID
Return value			
Data type	Mandatory	Description	
Boolean	Yes	Return true if the obj remove successful; otherwise, return false.	

Boolean generateToken(String appID, String key) This function will authenticate your access right with appID-key pair, and generate a token if you are an authorized developer. The APIManager will manage the token internally and refresh it periodically with appID-key pair.			
Input Parameter			
Parameter name	Data type	Mandatory	Description
appID	String	Yes	The registered appID on the platform.
key	String	Yes	The registered key with appID on platform.
Return value			
Data type	Mandatory	Description	
Boolean	Yes	Return true if token generate successfully. Otherwise, return false.	

Indoor edge localization

List<String> getGridIDListForEdgeLoc(String buildingID , double latitude, double longitude, double radius)

This function is used to find Grid ID lists of the targeted building covered by (latitude, longitude, radius) area, these GridID List will later be used for getting site signals.
It is recommended to use it when you don't have an indoor location. For example, the first time to request site signals in building.

Input Parameter			
Parameter name	Data type	Mandatory	Description
buildingID	String	Yes	ID of targeted building, API manager will get Grid ID in this building.
latitude	double	Yes	Latitude of GPS result
longitude	double	Yes	Longitude of GPS result
radius	double	Yes	A radius decided by the developer, this value will determine how many signals are requested for indoor localization. For example, if you want to get site signals within 20m radius of your location, set radius = 20.0

Return value		
Data type	Mandatory	Description
List<String>	Yes	Return list of gridID covered by latitude, longitude, radius

List<String> getGridIDListForEdgeLoc(String buildingID, Location location)

This function will find a list of Grid IDs. The current grid which the user's location is located, and grids connected to the current grid in one hop, will be put in the Grid ID list. This GridID List will later be used for getting site signals.

Input Parameter			
Parameter name	Data type	Mandatory	Description
buildingID	String	Yes	ID of targeted building, API manager will get Grid ID in this building.
location	Location	Yes	Indoor location of the user

Return value		
Data type	Mandatory	Description
List<String>	Yes	Return list of gridID that have a close relation with current user location.

List<String> getGridIDListForEdgeLoc(String buildingID)

This function will find a list of Grid IDs that are related to buildingID, helping callers to get all site signals in building.
This GridID List will later be used for getting site signals.

Input Parameter			
Parameter name	Data type	Mandatory	Description
buildingID	String	Yes	A decided buildingID provided by caller

Return value		
Data type	Mandatory	Description
List<String>	Yes	Return list of gridID that are related to decided building

JSONArray downloadSiteSignals(String buildingID, String siteSignalMode, List<String> gridIDList)

This function will download site signals covered by gridIDList with one siteSignalMode filter from the targeted building .

Input Parameter			
Parameter name	Data type	Mandatory	Description
building	String	Yes	ID of targeted building, API manager will get site signals in this building.
siteSignalMode	String	Yes	siteSignalMode tag that indicates the type of site signal you want to download.
gridIDList	List<String>	Yes	List of gridID you get by calling getGridIDListForEdgeLoc()

Return value		
Data type	Mandatory	Description
JSONArray	Yes	Return site signals covered by gridIDList in JSON Object array. e.g. [{WiFiFingerprintObj}]

Indoor cloud localization

List<String> getSignalTypeForCloudLoc(String buildingID)

This function will get the types of user signals needed by the cloud localization service of desired building.

Input Parameter			
Parameter name	Data type	Mandatory	Description
buildingID	String	Yes	ID of targeted building, API manager will get signal types of this building.

Return value		
Data type	Mandatory	Description
List<String>	Yes	Return list of tags that indicate the type of user signals to be uploaded in UploadSignaltoCloud() e.g. ['WiFi','BLE','Magnetic']

void uploadSignalToCloud(String buildingID, String userID, JSONObject userSignal)

This function will upload userID and userSignal to the cloud localization server of desired building according to the decided mode.

Input Parameter

Parameter name	Data type	Mandatory	Description
buildingID	String	Yes	ID of the targeted building, API manager will upload signal to the cloud localization server of the targeted building.
userID	String	Yes	ID to indicate your identity on cloud server

userSignal	JSONObject	Yes	The user signals to be uploaded, each type of signals should be associated with their type name.
			
			e.g. {
			”wifiRssVector”: [{
			”mac”:”c412f5c5bf48”,
			”rssi”:-84,
			”freq”:2,
			”timestamp”:142081941240},...],
			”bleRssVector”: [{
			”uuid”: uuid,
			”major”: major,
			”minor”: minor,
			”rssi”: rssi,
			”txPower”: tx power,
			”timestamp”: unix timestamp
			}, ...]
			}

Return value

Data type	Mandatory	Description
-	-	-

Location getCloudLocResult(String buildingID, String userID)

This function will get the user location by userID from cloud localization server of the desired building.

Input Parameter			
Parameter name	Data type	Mandatory	Description
buildingID	String	Yes	ID of the targeted building, APIManager will get user location from the cloud localization server of the targeted building.
userID	String	Yes	ID to indicate your identity on cloud server

Return value		
Data type	Mandatory	Description
Location	Yes	User indoor location provided by cloud localization server

Outdoor localization

List<String> discoverOutdoorSite(double latitude, double longitude, double accuracy)

This function is used to find outdoor sites the device is nearby. This attempt will list all outdoor site ID in descending order of coverage in the GPS location.

Input Parameter			
Parameter name	Data type	Mandatory	Description
latitude	double	Yes	Latitude of GPS result
longitude	double	Yes	Longitude of GPS result
accuracy	double	Yes	Accuracy of GPS result

Return value		
Data type	Mandatory	Description
List<String>	Yes	Return a List of OutdoorSiteID, the ID are sorted in descending order of coverage in the GPS location.

boolean initializeOutdoorSite(String outdoorSiteID)

This function is used to initialize OutdoorSiteObj in the APIManager. APIManager will download OutdoorSite information according to the given OutdoorSiteID.

Input Parameter			
Parameter name	Data type	Mandatory	Description
outdoorSiteID	String	Yes	ID of targeted Outdoor Site

Return value		
Data type	Mandatory	Description
boolean	Yes	Return true if initialization successful; otherwise, return false.

Booelan removeOutdoorSite(String outdoorSiteID)

This function is used to remove OutdoorSiteObj and OutdoorLocSetting in the APIManager.

Input Parameter			
Parameter name	Data type	Mandatory	Description
outdoorSiteID	String	Yes	ID of targeted Outdoor Site

Return value		
Data type	Mandatory	Description
List<String>	Yes	Return true if removed successfully; otherwise, return false.

JSONArray getOutdoorSignal(String outdoorSiteID, double latitude, double longitude, double radius, @Nullable String siteSignalMode)

This function will get the outdoor signal covered by latitude, longitude, radius with the decided siteSignalMode filter from the targeted outdoor site.

Input Parameter

Parameter name	Data type	Mandatory	Description
outdoorSiteID	String	Yes	ID of the targeted outdoor site
latitude	double	Yes	Latitude of GPS result
longitude	double	Yes	Longitude of GPS result
radius	double	Yes	A radius decided by the developer, this value will determine how many signals are requested for indoor localization. For example, if you want to get site signals within 20m radius of your location, set radius = 20.0
siteSignalMode	String	No	Site signal mode tag to indicate the type of outdoor signal you want to download. Default value is "BLELocation"

Return value

Data type	Mandatory	Description
JSONArray	Yes	Outdoor signal covered by latitude, longitude, radius with the decided siteSignalMode filter. For siteSignalMode = "BLELocation", the return value should be an array of BLELocation JSON objects. An exmaple is shown below the table.

Example:


```
[
    {
        "UUID":"b19af004-7f2a-4972-8f39-37d26c29cb9a",
        "major":"10001",
        "minor":"10001",
        "txPower":0,
        "ID":"b19af004-7f2a-4972-8f39-37d26c29cb9a1000110001",
```

```
        "coordinate":{
            "lat":22.428721,
            "lon":114.209055
        },
        "OutdoorSiteID":"39845318500220220423"
    },
    {
        "UUID":"b19af004-7f2a-4972-8f39-37d26c29cb9a",
        "major":"20001",
        "minor":"10001",
        "txPower":0,
        "ID":"b19af004-7f2a-4972-8f39-37d26c29cb9a2000110001",
        "coordinate":{
            "lat":22.428721,
            "lon":114.209055
        },
        "OutdoorSiteID":"39845318500220220423"
    }
]
```


Get map data for display

JSONObject getMapData(String spatialID)

This function will search all maps that links to the family of the given spatialID, and return their map id and map type to caller. For example, given a floorID, maps links the parent buildingObj, its floorObj and its child regionObj will be consider to return their mapID and mapType to caller.

Input Parameter			
Parameter name	Data type	Mandatory	Description
spatialID	String	Yes	spatialID of the building you want to download, either buildingID, floorID or regionID.

Return value		
Data type	Mandatory	Description
JSONObject	Yes	Return JSON Object links to SpatialID. This JSON Object contain three key pairs. "building" store array of mapid, maptype under building level, all JSONObject in this level should cover all floor and region under this building. "floor" store array of mapid, maptype under floor level, all JSONObject in this level should cover all region under this floor. "region" store array of mapid, maptype under region level. An example is shown under the table.

Example:

```
{
    "building": [{"mapid":"xxxxxxxxxx","maptype":"PNG"}],
    "floor": [{"mapid":"xxxxxxxxxx","maptype":"PNG"}],
    "region": [{"mapid":"xxxxxxxxxx","maptype":"PNG"}]
}
```

JSONObject getMapData(Location location)

This function will search all map that covers location, and return their map id and map type to caller.

Input Parameter			
Parameter name	Data type	Mandatory	Description
location	Location	Yes	User indoor location

Return value		
Data type	Mandatory	Description
JSONObject	Yes	Return JSON Object links to Location. This JSON Object contain three key pairs. "building" store array of mapid, maptype under building level, all JSONObject in this level should cover all floor and region under this building. "floor" store array of mapid, maptype under floor level, all JSONObject in this level should cover all region under this floor. "region" store array of mapid, maptype under region level. An example is shown before this table.

MapObj getMapObj(String mapID)

This function will find the MapObj required by mapID, and return it to caller.

Input Parameter			
Parameter name	Data type	Mandatory	Description
mapID	String	Yes	The mapID of the targeted map

Return value		
Data type	Mandatory	Description
MapObj	Yes	If a MapObj is found by mapID, return MapObj. If MapObj is not found, return null.

byte[] getMapFile(String filetype, String mapID)

This function will download the zipped map required by filetype and mapID, and return the map to caller in byte array format.

Input Parameter			
Parameter name	Data type	Mandatory	Description
filetype	String	Yes	Map file type you want to get
mapID	String	Yes	Map ID of the map you want to get

Return value		
Data type	Mandatory	Description
Byte[]	Yes	If a map file of fileType is found, return a zipped map file that is serialized into a byte array. If fileType is not found, return null.

Get building information

BuildingObj getBuildingObjByID(String buildingID)

Getter of BuildingObj, help developers in handshaking.

Input Parameter			
Parameter name	Data type	Mandatory	Description
buildingID	String	Yes	ID of the targeted building

Return value		
Data type	Mandatory	Description
BuildingObj	Yes	Return BuildingObj of the targeted building. Otherwise, return null.

BuildingLocSetting getBuildingLocSettingByID(String buildingID) Getter of BuildingLocSetting, help developers in handshaking.			
Input Parameter			
Parameter name	Data type	Mandatory	Description
buildingID	String	Yes	ID of the targeted building
Return value			
Data type	Mandatory	Description	
BuildingLocSetting	Yes	Return BuildingLocSetting of the targeted building. Otherwise, return null.	

List<String> getSignalModeByID(String buildingID) Getter of signalMode provided by the desired building, which indicates the type of signal available for indoor localization in this building.			
Input Parameter			
Parameter name	Data type	Mandatory	Description
buildingID	String	Yes	ID of the targeted building
Return value			
Data type	Mandatory	Description	
List<String>	Yes	Return signalMode of the desired building. Otherwise, return null.	

7.2.3 Localization Assistant

This class provides reference designs for the implementation of indoor localization done by developers. They are mostly related to making decisions with the device's state (indoor or outdoor, switch condition, etc.). You can call Localization Assistant API instead of implementing your own algorithm.

Indoor outdoor detection

Boolean detectIndoorEnviroment(double latitude, double longitude, double accuracy, @Nullable double threshold)

This function is to detect whether the device is in an indoor or outdoor environment. The accuracy of GPS result will reflect the level of blockage of satellite signals. If the accuracy > threshold, we consider satellite signals were blocked by the indoor environment, and vice versa.

Input Parameter			
Parameter name	Data type	Mandatory	Description
latitude	double	Yes	Latitude of GPS result
longitude	double	Yes	Longitude of GPS result
accuracy	double	Yes	Accuracy of GPS result
threshold	double	No	Threshold to determine indoor environment, default is 30m

Return value		
Data type	Mandatory	Description
Boolean	Yes	True if accuracy > threshold, indicating indoor environment. Otherwise, it returns false to indicate the outdoor environment.

Switch zone detection

String detectSwitchCondition(Location location) This function will check if location fall inside switch zone that exits current building, and return connected buildingID or "Outdoor" tag.			
Input Parameter			
Parameter name	Data type	Mandatory	Description
location	Location	Yes	Indoor location of device
Return value			
Data type	Mandatory	Description	
String	Yes	If the location is inside the switch zone to another building, return connectedID of switch zone (buildingID). If the location is inside an existing zone to outdoor, return "Outdoor". Otherwise, return "Null".	

8 Server Specifications for Site Owners

In mode 0 and 1, site owners should host their servers to deal with applications' requests following specifications in this section. Mode 0 site owners need to host a REST API server for downloading their site signals while mode 1 site owners need to host a server for computing location.

8.1 API URL

This is the REST API structure that is used in mode 0. Each API URL consists of a scheme, a host, a path and an optional query string.

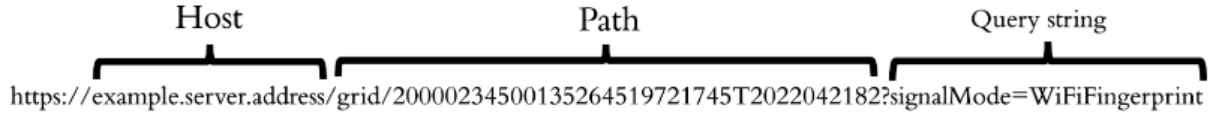

Figure 8.1: Structure of an API URL

- **Scheme:**
  The scheme is HTTPS by default.
- **Host:**
  The host is the server address of the site owner's server. Site owners need to upload this information to the lookup server after setting up the server.
- **Path:**
  The path is an identifier of the service. The path of each service is defined in this standard, and site owner's developers should follow the design to set up the server.

- **Query string:**
  The query string is one of the methods to send input parameters of the API to the server, and it is used in the GET methods.

In this standard, we specify the path and query string of an API URL. The host is determined by site owners, and the information is uploaded to the lookup server following instructions in section 9.

The path and query string are expressed as <path>?<query name, value pairs>, and the query value is denoted inside the curly braces. For example, in "/grid-id?zoomLevel=zoomLevel", "/grid-id" is the path, "zoomLevel" is the query parameter name, the curly brace denotes the value of "zoomLevel" query parameter. The curly brace is for easy reading only and will be removed in the real API call. That is, an instance of the example API could be "/grid-id?zoomLevel=19".

8.2 Mode 0 - APIs for Downloading Site Signals

Site owner's server returns their site signals upon the user's request in different searching criteria. Site owner's developers need to set up a REST API server with API URL indicated in Section 8.1.

8.2.1 GET - Request Signal Modes

GET	/signal-modes		
Return the supported signal modes that are available for applications.			
Input Parameter			
Name	Data type	Mandatory	Description
-	-	-	-
Response			
Name	Data type	Description	
signalModes	JSON array	The array of signal modes. Signal mode tags are: "WiFiFingerprint", "BLEFingerprint", "MagFingerprint", "BLELocation"	

Format:

```
{
    "signalModes": [
        "signal mode_1", "signal mode_2", ...
    ]
}
```

8.2.2 GET - Request GridIds

GET	/grid-id?zoomLevel={zoomLevel}		
Return the gridIds given the query parameters: If no query parameter is given, return all gridIds in the site. If zoom level is given, return the gridIds in that zoom level.			
Input Parameter			
Name	Data type	Mandatory	Description
zoomLevel	string	no	The desired zoom level
Response			
Name	Data type	Description	
gridIds	JSON array	The array of gridIds	

Format:

```
{
    "gridIds": [
        "gridId_1", "gridId_2", ...
    ]
}
```

8.2.3 GET - Request Site Signals by GridId

GET	/grid/{gridId}?signalMode={signalMode}
Return the site signals given the gridId(s) and optionally the signal mode If signal mode is given, return the site signals in the grid(s) with this mode. Note: API compression is highly recommended according to the "Accept-Encoding" in the HTTP header	

Input Parameter			
Name	Data type	Mandatory	Description
gridId	string	yes	The desired gridIds, seperated by commas. For example, $gridId_1, gridId_2, gridId_3$
signalMode	string	no	If this is given, only return the site signals with this mode. Signal mode tags are: "WiFiFingerprint", "BLEFingerprint", "MagFingerprint", "BLELocation"

Response		
Name	Data type	Description
grids	JSON array	The array of grids. Each element consists of gridId, fingerprints, and beaconLocations.
fingerprints	JSON array	The same format defined in Section 6.1.
beaconLocations	JSON object	The same format defined in Section 6.1.

Format:

```
{
    "grids": [
        {
            "gridId": "grid id",
            "fingerprints": [
                {
                    "rpId": "RP Id"
                    "latitude": "latitude",
                    "longitude": "longitude",
                    "floorId": "floor Id",
                    "wifiRssVector": ["mac:rss_value", ...],
                    "bleRssVector": ["uuid:major:minor:rssi", ...],
```

```
                "magneticSignal": ["magnetic field strength along x-axis", "
                    along y-axis", "along z-axis"]
            }, ...
        ],
        "beaconLocations": [
            {
                "beaconId": "beacon Id",
                "latitude": "latitude",
                "longitude": "longitude",
                "floorId": "floor Id"
            }, ...
        ]
    }, ...
]
}
```

8.3 Mode 1 – APIs for Computing Locations

Site owner's servers are required to compute user locations in either asynchronous or synchronous approaches. Both of them has a common API to request the supported signal modes. In asynchronous approach, there is one API for uploading user signals and one API for requesting location result. After receiving user signals and an id, the server computes and stores the location result. Upon users' request, the server returns the latest location. In synchronous approach, there is only one API for computing the user location given the user signal.

The API URL will not be restricted by this standard, and it is up to site owner's developers' design. This section specifies the input and output formats only. The URL information is uploaded to the lookup server following instructions in Chapter 9.

For the asynchronous approach, site owners should implement the APIs listed in Section 8.3.1, Section 8.3.2, and Section 8.3.3. For the synchronous approach, site owners should implement the APIs listed in Section 8.3.1 and Section 8.3.4.

8.3.1 GET - Request Signal Modes

Return the supported signal modes that are available for applications.

Query parameter

No

Response

```
{
    "signalModes": [
        "signal mode_1", "signal mode_2", ...
    ]
}
```

Note: signal mode tags are:
"WiFiFingerprint", "BLEFingerprint", "MagFingerprint", "BLELocation"

8.3.2 POST - Upload User Signals

Compute a location from the user signals and store it for later request.

Request body

```
{
    "userId": "string"
    "wifiRssVector": [
        {
            "mac": mac address,
            "rssi": rssi,
            "freq": frequency,
            "timestamp": unix timestamp when this AP is scanned
        },
        ...
    ],
    "bleRssVector": [
        {
            "uuid": uuid,
            "major": major,
            "minor": minor,
            "rssi": rssi,
            "txPower": tx power,
```

```
            "timestamp": unix timestamp when this beacon is scanned
        },
        ...
    ]
}
```

Response

No

8.3.3 GET - Request Latest User Location

Return the latest location given the user Id. If the location is invalid, return false in the "inBuilding" attribute, denoting the result is invalid.

Query parameter

Parameter	Type	Description
userId	String	The user Id

Response

```
{
    "inBuilding": True/False,
    "latitude": "latitude",
    "longitude": "longitude",
    "floorId": "floor id",
    "accuracy": "localization error in meters. Optional"
}
```

8.3.4 POST - Compute location

Compute and return a location from the user signals. A synchronized version of location computing API.

Request body

```
{
    "userId": "string"
```

```
    "wifiRssVector": [
        {
            "mac": mac address,
            "rssi": rssi,
            "freq": frequency,
            "ssid": ssid,
            "timestamp": unix timestamp when this AP is scanned
        },
        ...
    ],
    "bleRssVector": [
        {
            "uuid": uuid,
            "major": major,
            "minor": minor,
            "rssi": rssi,
            "txPower": tx power,
            "timestamp": unix timestamp when this beacon is scanned
        },
        ...
    ]
}
```

Response

```
{
    "inBuilding": True/False,
    "latitude": "latitude",
    "longitude": "longitude",
    "floorId": "floor id" ,
    "accuracy": "localization error in meters. Optional"
}
```

8.4 JWT for Authentication and Authorization

As site owners may want to authenticate users making requests to their server, JWT is used to provide authentication and authorization in this standard.

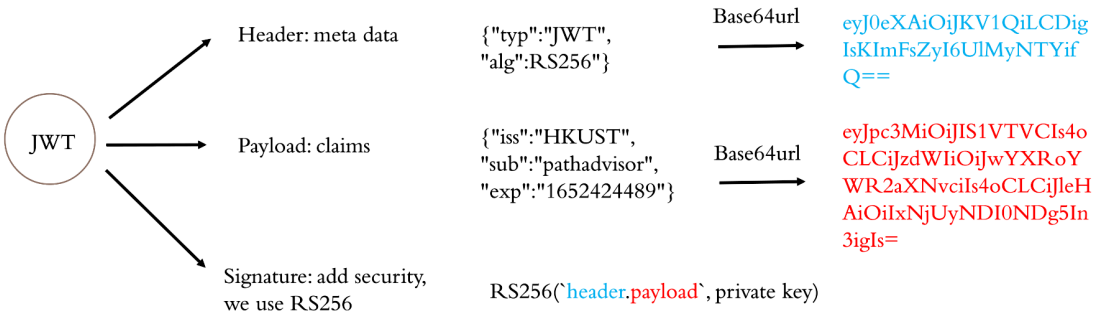

Figure 8.2: The JWT structure.

In simple words, a JWT consists of 3 parts, namely Header, Payload, and Signature. Header is the metadata of the token. Payload is the claims, providing the authentication. Signature is the digital signature of the header and the payload with the private key owned by the lookup server, providing the authorization. The detailed specifications of JWT can be found in [5], and this standard only describes the minimal part to make it works.

In our standard, we specify in the token:

1. the issuer who issues the token
2. the subject who is granted the token by the issuer
3. the issued time
4. the expired time

8.4.1 Data Specification of a JWT

Header

Parameter	Type	Description
tpy	Type of this json object which is the JWT	"JWT"
alg	Algorithm used in the signature	"RS256"

Payload

Parameter	Type	Description
iss	Issuer of the token	"HKUST Lookup Server"
sub	Subject of the token, identified by the App Id	App Id
iat	Issued time	Time in Unix time
exp	Expired time	Time in Unix time

Signature

Denote the private key held by the lookup server as k. The signature is the ciphertext from RS256 using "header.payload"(in Base64Url format) as the plaintext and k as the key. For example, in Fig. 8.2, the signature is the reesult of:
RS256("eyJ0eXAiOiJKV1QiLCDigIsKImFsZyI6UlMyNTYifQ==.eyJpc3MiOiJIS1VTVC
Is4oCLCiJzdWIiOiJwYXRoYWR2aXNvciIs4oCLCiJleHAiOiIxNjUyNDI0NDg5In3igIs="
,k).

Public Key of the Lookup Server

The RSA public key of the lookup server (2048 bits) is:

—BEGIN PUBLIC KEY—
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAs4xkpwTPqnICaD4+f7W
/Uu7rrkxsZELVR9mxiFcJsgMr37utTl0gNwl4R0tMHmCXNDOTWiQ6fEOr4ssDjpyeq1zl
dbMebR5I1MUQSV+y5IaLJSJFJlxHzp4xgWYauVT5lKUwjf11TJ+pDyOV4femTztHL8
DjgUQXuwld1q9BF4ZqYoZD0k6rTBZT09ev6jC9H0oB7qp1c984HHbaznLzUzKK0b4QC0
+r41Z9alwkA+1vUgwITiIki9LMbiPHBKiaAcWnHLAnjC7rqR+rxyWEpJQ5wr4TAIAG
dEmc27qNOjX0gbet3vR8ZZV26tCxGHpSFiGG5DLgOROizOJ9esIfHQIDAQAB
—END PUBLIC KEY—

8.4.2 Validating a JWT

The procedures to validate a JWT:

1. compute the plaintext from the ciphertext using the public key of the lookup server. That is, RS256(signature, public key)

2. do a character-wise comparison between the plaintext and the "header.payload" extracted from the token. If they are not the same, this token is invalid

3. check if every attribute is valid according to Section 8.4.1. If any attritube is invalid, the token is invalid

8.4.3 JWT Location

The JWT should be put in the header in the request, so site owner's server only need to validate the token in header if site owners want to authenticate users.

9 Site Signal and Map Data Validation for Site Owners

To enable pervasive positioning, the platform requires site owners to upload a data package including site information, spatial representation, maps, site signals and metadata for the grid. Contents in the data package should be arranged according to the standard specification, including file structure, file naming principle, file contents format and its values.

This section aims to specify the formats and requirements of the data package for site owners to prepare the data package of their site. This specification will describe the requirements of each file in the file system tree, from the utmost file SiteInfo.json to the WifiFingerprint.txt in the deepest level of the tree.

9.1 Data Package Structure – File System Tree

9.1.1 Indoor Site Data Package

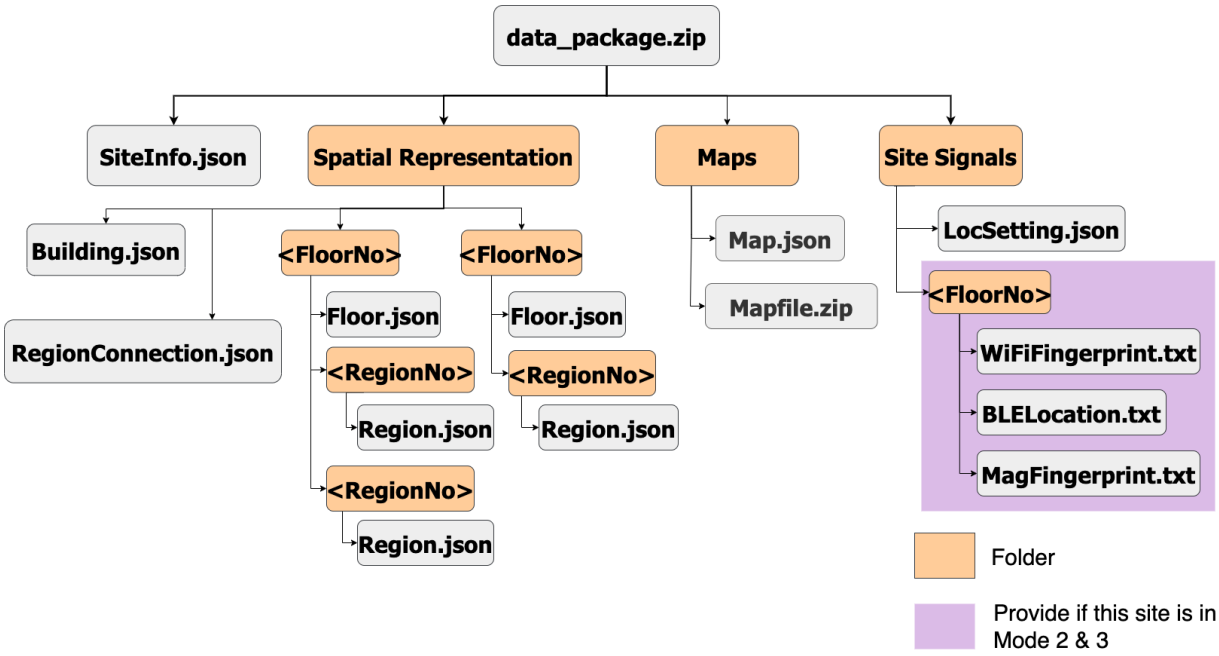

Figure 9.1: The file system tree for indoor data validation.

All blocks are named by their filename or folder name. Blocks colored with orange are folders in data package, they named by categories or tags (FloorNo or RegionNo).

Blocks colored with purple are site signals file of this building, site owner who wants to share site signal to platform should include this part.

9.2 Outdoor Site Data Package

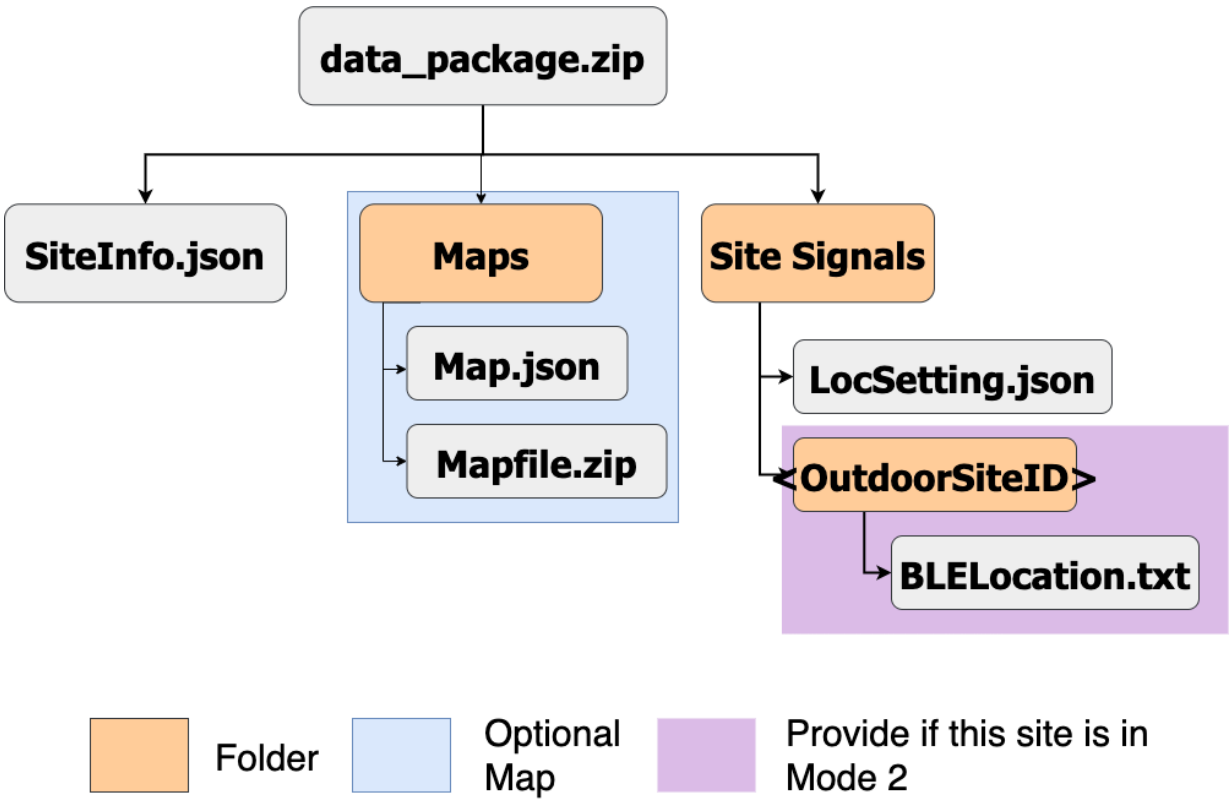

Figure 9.2: The file system tree for outdoor data validation.

Outdoor data package contains less information, as outdoor space is less complex. It contains SiteInfo.json, OutdoorLocSetting.json and some optional data: map.json, map file and outdoor site signals.

Optional folder Maps in outdoor data package allow site owner provides customize outdoor map instead of global map on the internet. The map file should be zipped.

Site owner can provide outdoor site signal to platform in operation mode 2 Platform-supported Edge Loc.

9.3 Data File Specification – Indoor Site Package

In the top level of the data package, SiteInfo.json, Spatial Representation folder and Site Signals folder will be located at this level.

9.3.1 SiteInfo.json

SiteInfo.json is a JSON file storing site Information, including site address, site owner information and site owner's contacts. They should be stored in three fields:

Attribute	Data type	Description
SiteAddress	String	Postal address of site
SiteOwner	String	Site owner description
Contacts	String	Email address for contact site owner
IndoorSite	Boolean	True if this site is an indoor site, false if it is outdoor site. For site with semi-outdoor area, site owner should also put it as indoor area.

Example:

```
{
    "SiteAddress": "HKUST, Clear Waterbay",
    "SiteOwner": "HKUST"
    "Contacts": "itsc@ust.hk",
    "IndoorSite": true
}
```

9.3.2 Spatial Representation Folder

This folder stores all spatial representations and map files. In this level, Building.json, Floor folders named by their FloorNo are located here.

Building.json

Building.json should contain all building information, including ID, Name, MapDataID, FloorList and DefaultFloorNo.

Attribute	Data type	Description
Name	String	Display name of building
FloorList	Array of String	A list of Floor number contained by this building in ascending order of elevation
DefaultFloorNo	String	The default floor number of the building to be displayed (e.g., FloorNo hexstring of G/F)

Example:

```
{
    "BuildingID": String,
    "Name": String,
    "FloorList": ["FloorNo"],
    "DefaultFloorNo":" FloorNo",
    "MapDataID": ["MapID"]
}
```

Floor Folder

This folder should be named by FloorNo, containing Floor.json, region folders under this floor.

Floor.json

This JSON file contains floor information, including FloorNo, Name, ParentID, RegionList, DefaultRegionNo and MapDataID (optional).

Attribute	Data type	Description
FloorNo	String	An 8-bit hex string (2 letters) indicating the floor number. Refer to Section 6.2.
Name	String	The display name of the floor excluding the building name
RegionList	Array of String	A list of RegionNo contained by this floor
DefaultRegionNo	String	Region number of default region

Example:

```
{
    "FloorNo": String,
    "Name": String,
    "ParentID": ["BuildingID"],
    "RegionList": ["RegionNo"],
    "DefaultRegionNo": "RegionNo",
    "MapDataID": ["MapID"]
}
```

Region Folder

This folder contains Region.json under this region.

Region.json

This file contains region information, including RegionNo, Name, ConnectedList and MapDataID (optional).

Attribute	Data type	Description
RegionNo	String	An 8-bit hex string (2 letters) indicating the region number.
Name	String	The display name of the region excluding the floor name.

RegionConnector	Array of JSON object {Name, Geometry}	An array of JSON objects that indicates the possible region transitions of this region, for example, a door or an elevator. The name is used as an identifier when specifying the relation between connectors, and the relations are described in RegionConnection.json. The naming convention is the concatenation of the type of connector and a number, for example, "Lift 1" or "Escalator 10". The tag is an optional attribute to briefly describe the nature of this connection for potential localization optimization. Available tags are "Elevator", "Escalator", "Straight stair", "Switchback stair", "Door", "Ramp", "Outdoor" (for outdoor connectors), "External" (for connecting regions in other building), and "Other". The geometry is the poly-line specified in array of lon, lat (2D-array) to describe the locations of this connector. Each JSON should be described as:
		
		{
		"Name": String,
		"Tag": String,
		"Geometry": Array of [lon, lat]
		}

Grid	Array of string	The array of 2D grid Ids in zoom level 20 occupied by this region. The specified area should be supported with location services.
Constraint	GeoJSON MultiPolygon coordinate arrays (optional)	The multi-polygon indicating the walkable area of this region to improve the localization accuracy. The detailed definition can be found in the official document of GeoJSON (https://datatracker.ietf.org/doc/html/rfc7946)

Example:

```
{
    "RegionNo": String,
    "Name": String,
    "ParentID": String,
    "MapDataID": ["MapID"],
    "ConnectedRegions": [
        {
            "TransitionArea": [
                [lon,lat], [lon,lat], [lon,lat], [lon,lat]
            ],
            "ArrivalArea": [
                {
                    "RegionID": String,
                    "Area": [
                        [lon,lat], [lon,lat], [lon,lat], [lon,lat]
                    ]
                }
            ]
        }
    ]
}
```

RegionConnection.json

This JSON file contains the topological information between regions.

Attribute	Data type	Description

TwoConnector	Array of JSON object {From, To, Bidirectional}	The connection between two connectors, for example, an escalator. "From" and "To" describe the RegionID and the name of the connector. For connections to outdoor, "Name" should be "Outdoor", and "FloorNo" and "RegionNo" should be null. "Bidirectional" is a Boolean indicator to denote whether the connection is unidirectional or bidirectional. Each JSON object is described as:
		{
		"From": {
		"FloorNo": Sting,
		"RegionNo": Sting,
		"Name": String
		}
		"To": {
		"FloorNo": Sting,
		"RegionNo": Sting,
		"Name": String
		}
		"Bidirectional": Boolean
		}

SeveralConnector	Array of JSON object {Connector}	The strongly connected connectors, for example, an elevator. Expected to have number of connectors >=3. "Connectors" describes the RegionID and the name of each connector. Each JSON object is described as:
		{
		"Connector": [{
		"FloorNo": Sting,
		"RegionNo": Sting,
		"Name": String
		}]
		}

9.3.3 Maps folder

This folder is the entry point of all maps and their metadata.

Map.json

This JSON file (in array form) is the metadata for each map file, including MapID, MapFormat, GeodeticPoints, Boundary, AttachedPrimalSpaceID and Filename. It starts with "[". Each object inside the array is described as:

Attribute	Data type	Description
Filename	String	Filename of map file. Should be unique.
MapFormat	String	The format of the map, e.g., JPG, PNG, IndoorGML, IMDF, GeoJSON, etc.

GeodeticPoint	Array of JSON object {X, Y, Lon, Lat}	Used in image formats (png, jpg, etc.). The array of geodetic points. Each geodetic point contains a coordinate in the map and a [lon,lat] pair. It is used in image format, and the geodetic points should be the 4 corners sorted in clockwise direction. JSON object of each geodetic point is described as:
		{
		"X": Double
		"Y": Double
		"Lon": Double
		"Lat": Double
		}

ImdfFloor	Array of JSON object FloorId, ImdfFloorUuid	Used for IMDF if the IMDF is in the building layer (includes multiple floors). It is the array of each floor association between this standard and the IMDF. Each element is a JSON object with attributes "FloorId" and "ImdfFloorUuid". The "FloorID" is the floodId in this standard while the "ImdfFloorUuid" is the uuid of the floor in IMDF. Each JSON object is described as:
		{
		"FloorID": String
		"ImdfFloorUuid": String
		}

Grid	Array of string	The occupied grid Ids of this map in zoom level 20. They should be outdoor grids and the superset of the union of the grid list in the attached spaces of this map.
AttachedPrimalSpaceID	Array of string	ID of Spatial Object (BuildingID/ FloorID/ RegionID/ SiteID) that this map links to.
Validation	Boolean	Declaimer of map format is valid to the file standard (e.g. IMDF, IndoorGML)

Example:

```
{
    "MapID":String,
    "MapFormat":String,
    "AttachedPrimalSpaceID ":String,
    "Filename":String,
    "GeodeticPoints": [
        {X, Y, Lon, Lat}, {X, Y, Lon, Lat}, ...
    ],
    "Boundary": [
        [lon,lat], [lon,lat], [lon,lat],[lon,lat], ...
    ],
    "Validation": true
}
```

Mapfile.zip

This is the zipped package of map files. It should contain all map files referring to each object in Map.json.

9.3.4 Site Signals Folder

This folder is the entry point of all site signals and their metadata, site signals are arranged according to the floor ID in this folder.

LocSetting.json

This JSON file stores properties related to localization, including BuildingID, boolean to indicate sharing site signal with platform, SignalMode , CloudLocSignaMode and URLs for distributed server approach.

If site owner wants to share site signal with platform, you should put "True" in ShareSiteSingal and include site signals in the data_package. Otherwise, you should put "false" and provide URLs.

Attribute	Data type	Description
BuildingID	String	ID of Building, CSUID
SupportedFloor	Array of String	An array of floor numbers that defines the floors supported with localization services.
OutdoorBaseMap	String	The outdoor base map that is used when designing the maps or localization algorithm. It can be "OSM", "Google Maps", or "Apple Maps".
ShareSiteSignal	Boolean	True if site owner shared their site signal with platform. Otherwise, set as false.
SiteSignalMode	Array of String (only for ShareSiteSignal == true)	Array of site signal tags to describe available signal for localization. Signal tags: "WifiFingerprint", "BLELocation", "MagFingerprint" For site owners not sharing their site signals, please follow Chapter 8 to provide their SiteSignalMode for applications.
RemoteSignalDownloadURL	String (only for Mode 0)	URL for downloading site signal package from site server.
RemoteCloudLocUploadURL	String (only for Mode 1)	URL for uploading user's signal to site server for cloud localization.
RemoteCloudLocDownloadURL	String (only for Mode 1)	URL for downloading user localization results from the site server.
RemoteCloudSyncLocURL	String (only for Mode 1)	URL for computing the location synchronously.
RemoteCloudSignalModeURL	String (only for Mode 1)	URL for the signal mode supported from the site server.

Example (Mode 1 site owner):

```
{
    "BuildingID": "CSUID",
    "SupportedFloors": ["00","01","02","03"],
    "Grids": [ "2007125470451263","2007125470451263" ] ,
    "ShareSiteSignal": false,
    "RemoteCloudLocUploadURL": "https://cloudLocUploadURL:port",
    "RemoteCloudLocDownloadURL": "https://cloudLocDownloadURL:port",
    "RemoteCloudSignalModeURL": "https://cloudSiteSignalModeURL:port"
```

}

Site Signal of Each Floor

Site signals should be arranged according to floor structure, each floor folder is named by FloorNo.

WifiFingerprint.txt

This file stores all Wifi fingerprints collected on the floor with FloorNo specified in the upper level. Each row in WifiFingerprint.txt stores one WifiFingerprint on a reference point, it contains Latitude, Longitude, FloorNo and an array of string that stores key-pair of Mac Address and RSSI. The format is:

latitude,longitude,FloorNo|["mac:rssi","mac:rssi",.....,"mac:rssi"]

Notice that separator "|" is used to separate location information and signal information. Spacebar should not appear in the file.

File name	WifiFingerprint.txt
Content	latitude,longitude,FloorNo|["mac:rssi","mac:rssi",.....,"mac:rssi"] latitude,longitude,FloorNo|["mac:rssi","mac:rssi",.....,"mac:rssi"] latitude,longitude,FloorNo|["mac:rssi","mac:rssi",.....,"mac:rssi"] latitude,longitude,FloorNo|["mac:rssi","mac:rssi",.....,"mac:rssi"] latitude,longitude,FloorNo|["mac:rssi","mac:rssi",.....,"mac:rssi"]

BLELocation.txt

This file stores all beacon identity and its location on the floor with FloorNo specified in the upper level. Each row in file stores one beacon identity, the format is:

latitude,longitude,FloorNo|UUID,major,minor

Notice that separator "|" is used to separate location information and beacon identity. Spacebar should not appear in the file.

File name	BLELocation.txt
Content	latitude,longitude,FloorNo|UUID,major,minor latitude,longitude,FloorNo|UUID,major,minor latitude,longitude,FloorNo|UUID,major,minor latitude,longitude,FloorNo|UUID,major,minor

MagFingerprint.txt

This file stores all magnetic field signal and its location on the floor with FloorNo specified in the upper level. Each row in file stores an array of magnetic field signal on the location, the format is:

latitude,longitude,FloorNo|["mag_x,mag_y,mag_z",.....,"mag_x,mag_y,mag_z"]

Notice that separator "|" is used to separate location information and beacon identity. Spacebar should not appear in the file.

File name	MagFingerprint.txt
Content	latitude,longitude,FloorNo|["mag_x,mag_y,mag_z",.....,"mag_x,mag_y,mag_z"] latitude,longitude,FloorNo|["mag_x,mag_y,mag_z",.....,"mag_x,mag_y,mag_z"] latitude,longitude,FloorNo|["mag_x,mag_y,mag_z",.....,"mag_x,mag_y,mag_z"]

9.4 Data File Specification – Outdoor Site Package

9.4.1 SiteInfo.json

siteInfo.json is a JSON file storing site Information, including site address, site owner information and site owner's contacts. They should be stored in three fields:

Attribute	Data type	Description
SiteAddress	String	Postal address of site
SiteOwner	String	Site owner description
Contacts	String	Email address for contact site owner
IndoorSite	Boolean	True if this site is an indoor site, false if it is outdoor site. For site with semi-outdoor area, site owner should also put it as indoor area.

Example:

```
{
    "SiteAddress": "HKUST, Clear Waterbay",
    "SiteOwner": "HKUST"
    "Contacts": "itsc@ust.hk",
```

```
    "IndoorSite": true
}
```

9.4.2 Maps folder

Please refer to Section 9.3.3.

9.4.3 Site Signals Folder

This folder is the entry point of all site signals and their metadata, site signals are arranged in this folder.

LocSetting.json

This JSON file stores properties related to localization, including OutdoorSiteID, boolean indicate sharing site signal with platform, SignalMode and URLs for distributed server approach.

If site owner wants to share site signal with platform, you should put "True" in ShareSiteSingal and include site signals in the data_package. Otherwise, you should provide download site signal URL.

Attribute	Data type	Description
OutdoorSiteID	String	Outdoor CSUID, it consists of geo-reference number, polygon type and creation date. Geo-reference number: a 10-digit identifier formed by combining the Easting and Northing of the label point within the outdoor site boundary. (Easting and Northing are from HK 1980 Grid Coordinates, decimal is truncated and the first digit is removed from the coordinates.) Polygon type: 'O' for Outdoor. Creation Date: YYYYMMDD e.g. OutdoorSiteID: "4520522021O20220412"
Grid	Array of string	The grid Ids of the supported area of this site.
OutdoorBaseMap	String	The outdoor base map that is used when designing the maps or localization algorithm. It can be "OSM", "Google Maps", or "Apple Maps".
ShareSiteSignal	Boolean	True if site owner shared their site signal with platform. Otherwise, set as false.
SiteSignalMode	Array of String (only for ShareSiteSignal == true)	Array of site signal tags to describe available signal for localization. Signal tags: "BLELocation".
RemoteSignalDownloadURL	String (only for Mode 0)	URL for downloading site signal package from site server.

Example (Mode 0 site owner):

```
{
    "OutdoorSiteID": String,
    "Boundary": [
        [lon,lat], [lon,lat], [lon,lat], [lon,lat]
    ],
    "ShareSiteSignal": false,
    "RemoteSignalDownloadURL": "https://RemoteSignalDownloadURL:port"
}
```

<OutdoorSiteID> Folder

A folder named by the OurdoorSiteID.

BLELocation.txt

This file stores all beacon identity and its location on the outdoor site. Each row in file stores one beacon identity, the format is:

latitude,longitude,OutdoorSiteID|UUID,major,minor

Notice that separator "|" is used to separate location information and beacon identity. Spacebar should not appear in the file.

File name	BLELocation.txt
Content	latitude,longitude,OutdoorSiteID|UUID,major,minor latitude,longitude,OutdoorSiteID|UUID,major,minor latitude,longitude,OutdoorSiteID|UUID,major,minor latitude,longitude,OutdoorSiteID|UUID,major,minor

9.5 Data Validation on the Platform

When the data package is uploaded by the site owner, it will pass through a validation process by the platform operator to check if it conforms to the standard. If it passes the validation, it will be managed and recorded into the database for web API requests. Otherwise, the platform operator will contact the site owner through email to revise their data package.

The data validation tool operates on the platform only checks the file structure, data format and necessary information for communication, data accuracy inside the data file will not be checked. For example, the existence of field name "BuildingName" will be checked in file Building.json, but the value of "BuildingName" : "dummy building" will not be checked. Site owners and developers should be aware of the data accuracy.

10 User Journey Examples

10.1 Setting

This setting is used throughout this section:

There are two buildings, namely building A and building B, and the first floor of building A and B are connected by a skyway.

Building A's site owner is not willing to share his site signals with the lookup server while building B's site owner is. Building A's site owner hosts a server for computing location and a server for downloading site signals, he sends his server addresses and maps to the lookup server. Buidling B's site owner sends his site signals and maps to the lookup server.

Building A's site signals are iBeacon locations while building B's site signals are WiFi fingerprints.

Application uses reference designs in SDK by default.

10.2 For Each Operation Mode

10.2.1 Operation Mode 0

Say the user is in building A, and the application is initializing a location.

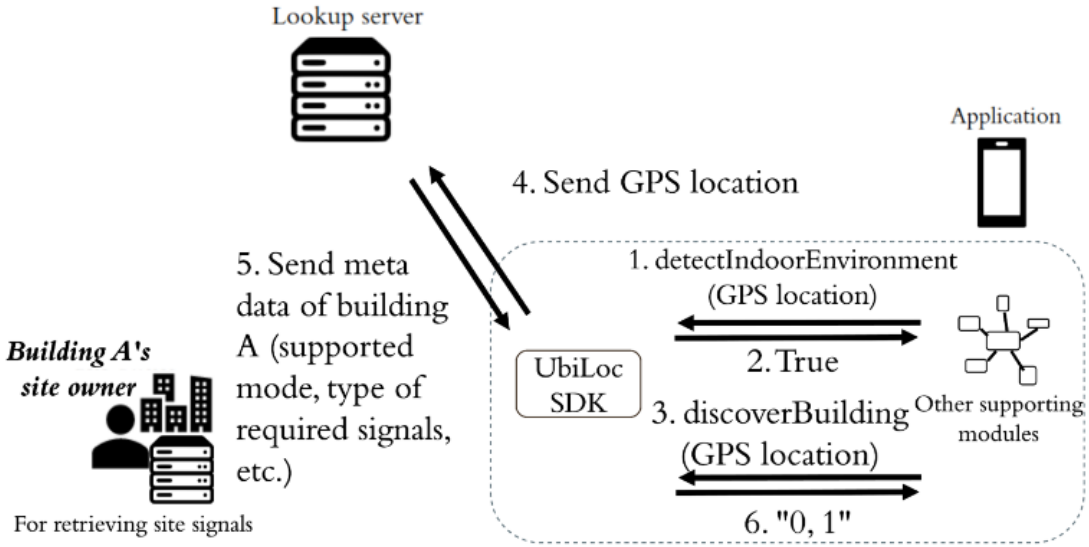

Figure 10.1: Detecting indoor environment and handshaking for operation mode of building A

First, the application initiates a Localization Assistant and an API Manager using the SDK, and call the detectIndoorEnvironment() in Localization Assistant by inputting the location result from GPS. It should return true since the user is inside the building.

Then call the discoverBuilding() in API Manager by inputting the location result from GPS. The API Manager then contacts the lookup server to find the nearest building of the given location which is building A in this case, and returns the supported modes of building A. The result should be "0, 1" since building A supports both operation mode 0 and 1.

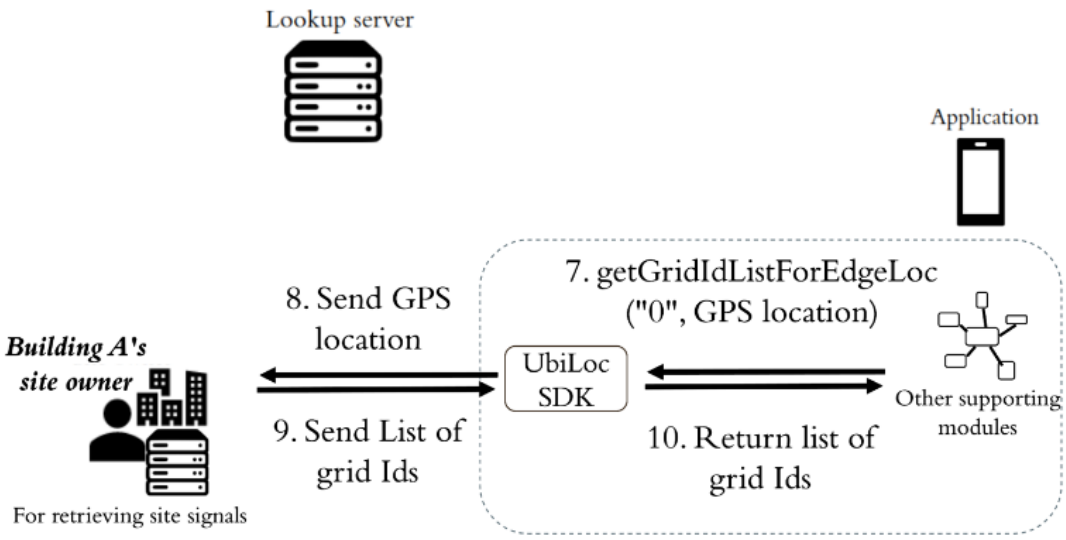

Figure 10.2: Requesting grid Ids from the site owner's server

Then the application calls the getGridIdListForEdgeLoc() in the API Manager to get the grid Ids of grids that are close to the GPS location from the site owner's server.

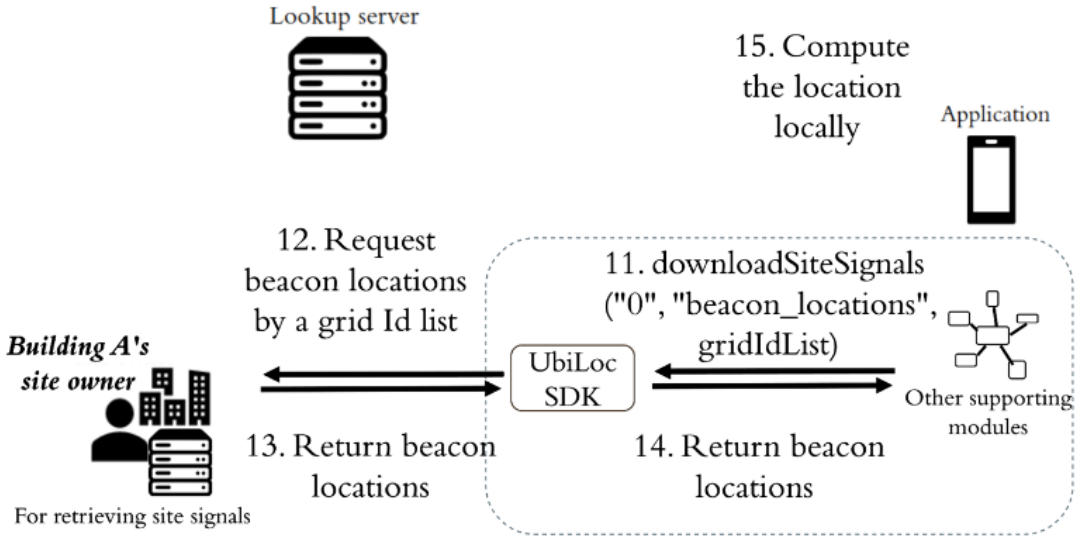

Figure 10.3: Requesting grids from the site owner's server and computing location locally

Since the application chooses mode 0 which is an edge approach, the application calls the downloadSiteSignals() in the API Manager to download the site signals that are the beacon locations in this case from the site owner's server. The location is then computed locally using those downloaded beacon locations.

In every location computation, the application first checks if the downloaded beacon locations can be used to provide a valid location. If not, meaning that the user moves away from the region covered by the downloaded beacon locations, then new beacon locations need to be downloaded by handshaking again or manipulating the grid Id list and requesting again for advanced developers.

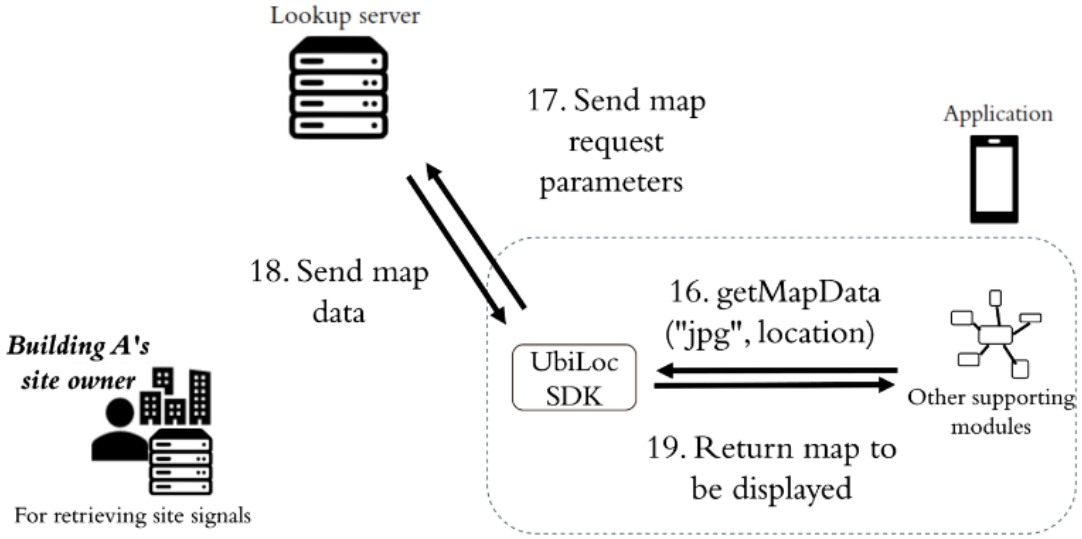

Figure 10.4: Requesting maps from the lookup server

With the location, the application calls the getMapData() in the API Manager to obtain the map in its preferred format, .jpg in this case, from the lookup server.

10.2.2 Operation Mode 1

Say the user is in building A, and the application is initializing a location.

Steps 1 – 6 are the same as operation mode 0 to confirm the building and the operation mode.

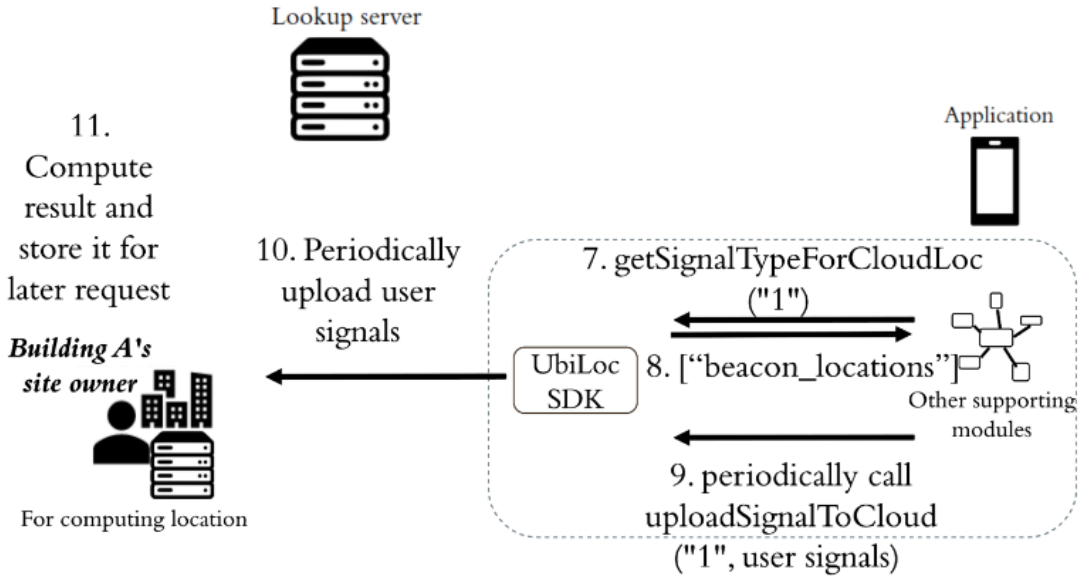

Figure 10.5: Confirming required signal types from and uploading user signals to the site owner's server

Since the application chooses mode 1 which is a cloud approach, it needs to know what signals are supposed to be uploaded by calling the getSignalTypeForCloudLoc() in API Manager. The result should be ["beacon_locations"]. Next is to periodically, say 10s, collect the beacon signals and call the uploadSignalToCloud() in the API Manager. After receiving the user signals, building A's server needs to compute and store the result for future location requests.

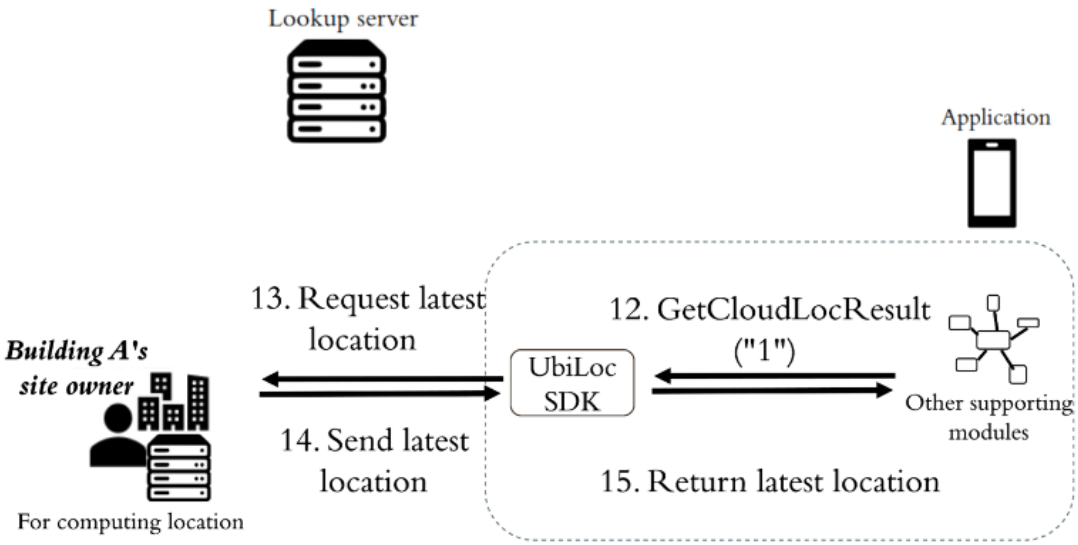

Figure 10.6: Requesting latest location from the site owner's server

The application then calls the getCloudLocResult() in the API Manager to retrieve the location from the site owner's server, and the server returns the latest location of this user.

With the location, the application calls the getMapData() in the API Manager to obtain the map from the lookup server, similar to operation mode 0.

10.2.3 Operation Mode 2

Say the user is in building B, and the application is initializing a location.

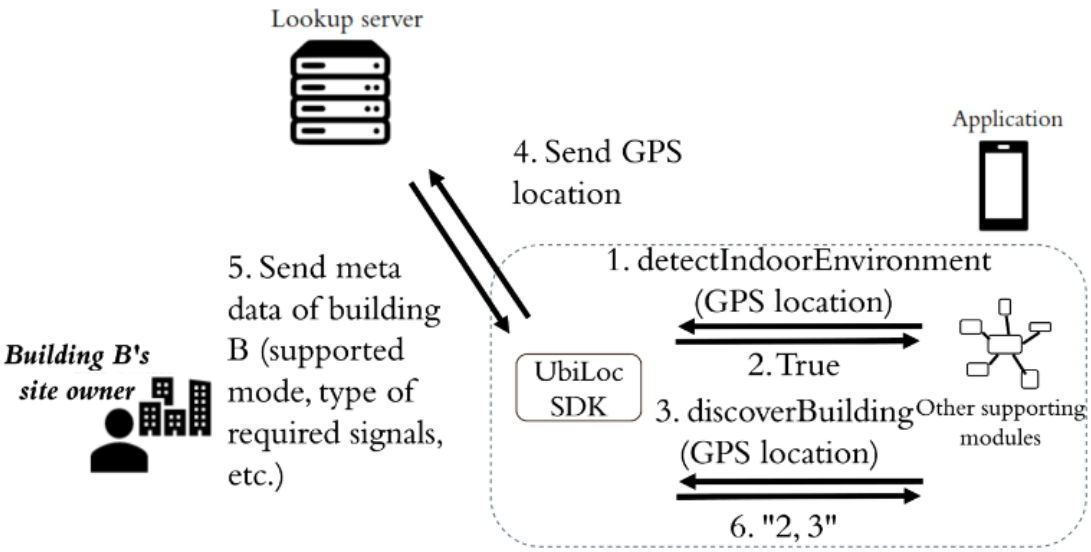

Figure 10.7: Detecting indoor environment and handshaking for operation mode of building B

First, the application initiates a Localization Asistant and an API Manager using the SDK, and call the detectIndoorEnvironment() in Localization Asistant by inputting the location result from GPS. It should return true since the user is inside the building.

Then call the discoverBuilding() in API Manager by inputting the location result from GPS. The API Manager then contacts the lookup server to find the nearest building of the given location which is building B in this case, and returns the supported modes of building B. The result should be “2, 3” since building B supports both operation mode 2 and 3.

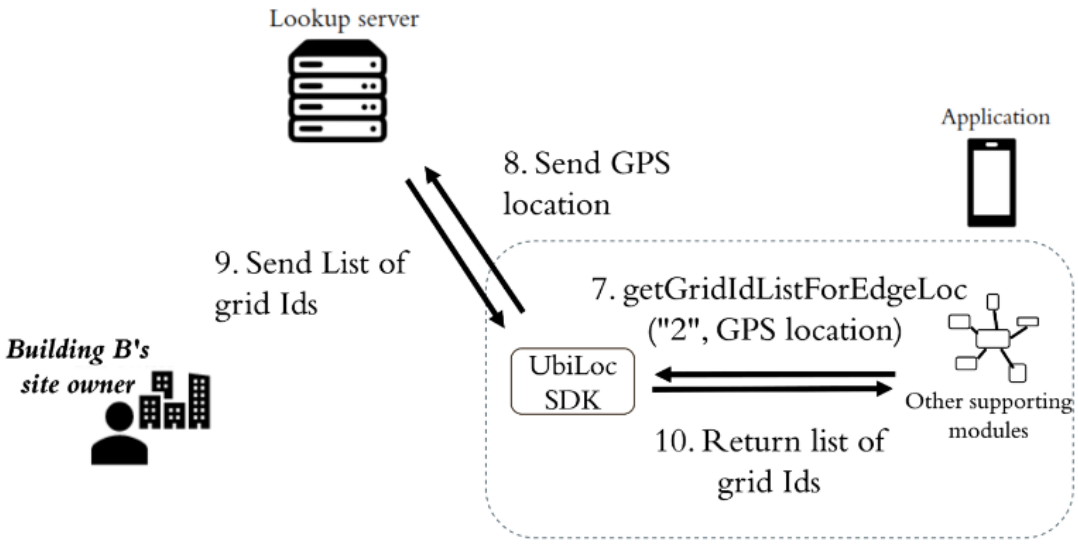

Figure 10.8: Requesting grid Ids from the lookup server

Then the application calls the getGridIdListForEdgeLoc() in the API Manager to get the grid Ids of grids that are close to the GPS location from the lookup server.

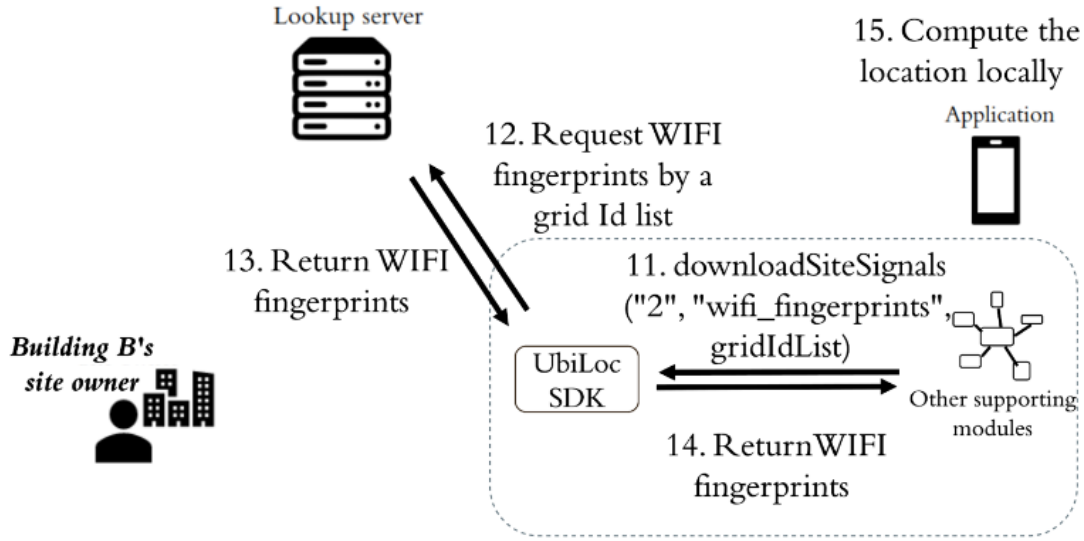

Figure 10.9: Requesting grids from the lookup server and computing location locally

Since the application chooses mode 2 which is an edge approach, the application calls the downloadSiteSignals() in the API Manager to download the site signals that are the WIFI fingerprints in this case from the lookup server. The location is then computed locally using those downloaded fingerprints.

In every location computation, the application first checks if the downloaded fingerprints can be used to provide a valid location. If not, meaning that the user moves away from the region covered by the downloaded fingerprints, then new fingerprints need to be downloaded by handshaking again or manipulating the grid Id list and requesting again for advanced developers.

With the location, the application calls the getMapData() in the API Manager to obtain the map from the lookup server, similar to other operation modes.

10.2.4 Operation Mode 3

Say the user is in building B, and the application is initializing a location.

Step 1 – 6 are the same as operation mode 2 to confirm the building and the operation mode.

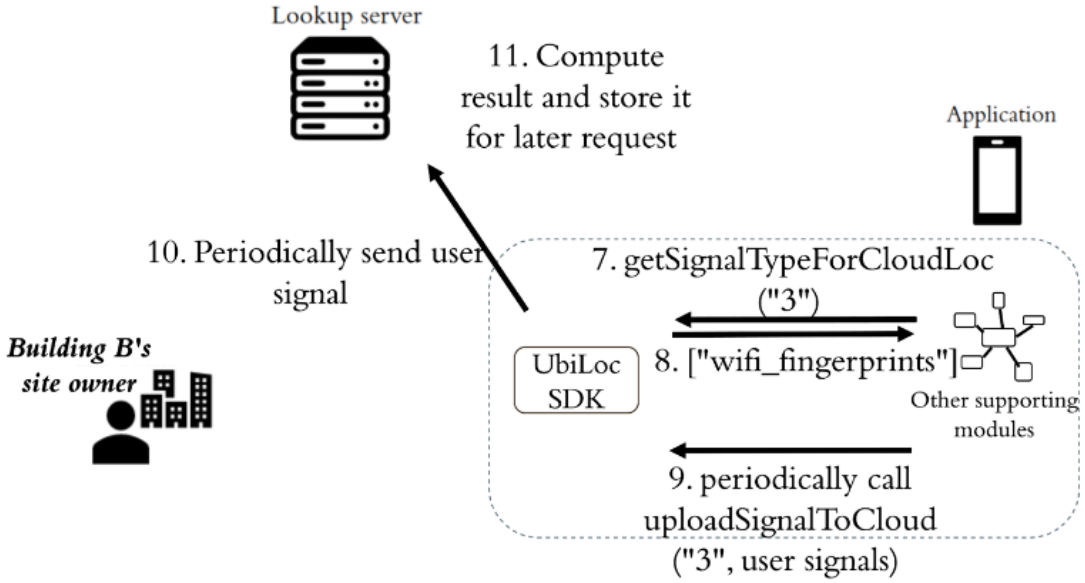

Figure 10.10: Confirming required signal types from and uploading user signals to the lookup server

Since the application chooses mode 3 which is a cloud approach, it needs to know what signals are supposed to be uploaded by calling the getSignalTypeForCloudLoc() in API Manager. The result should be [“wifi_fingerprints”]. Next is to periodically, say 10s, collect the WIFI user signals and call the uploadSignalToCloud() in the API Manager. After receiving the user signals, the lookup server needs to compute and store the result for future location requests.

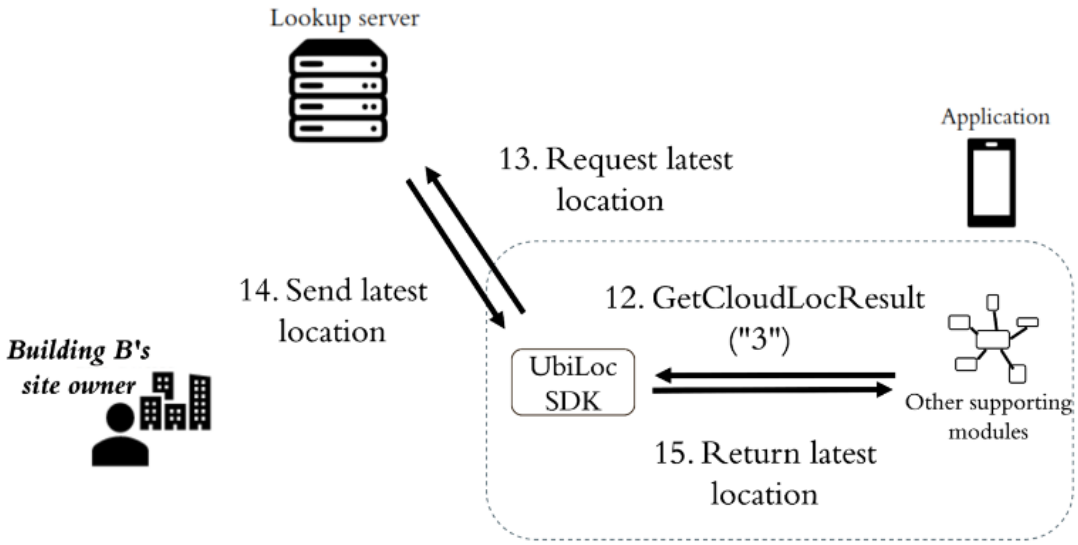

Figure 10.11: Requesting latest location from the lookup server

The application then calls the getCloudLocResult() in the API Manager to retrieve the location from the lookup server, and the server returns the latest location of this user.

10.3 Switching Floor and Mode

10.3.1 Switching Floor

In cloud approaches, switching floor does not require application developers to initiate. Since the location computations are in the site owner's server (mode 1 in building A) or the lookup server (mode 3 in building B), they are responsible for detecting if the user enters a new floor and returning the correct location result. Application developers only need to check if the floorId is different from the previous one, indicating new map needs to be downloaded.

In edge approaches, application developers are responsible for detecting if the user enters a new floor. With the downloaded beacon locations (building A) or fingerprints (building B), the application can detect if the user is in the supported region of the downloaded signals, using the location result and the grid information of the downloaded grids. If the user is in the boundary grids or outside the grids, the application should load new site signals, and the simplest way is to handshake again. More advanced developers may want to manipulate with the grid Ids by their understanding of the data format designs, e.g.,

BuildingObj, FloorObj, GridObj, etc. and retrieve the suitable grids from the site owner's server (mode 0 in building A) and the lookup server (mode 2 in building B).

10.3.2 Switching Mode

Switching mode is rather straightforward. The application should periodically call the detectSwitchCondition() in the Localization Asistant. Say the user is moving from building A to building B. The detectSwitchCondition() should return building B's Id at some point. When this happens, the application can handshake again with the lookup server, and steps 1-6 in operation mode 2/3 in the previous section should be processed to locate the user in building B with a minor difference that the application calls the discoverBuilding() with the building B's Id instead of the GPS location.

10.4 Outdoor Localization

Say the application is initializing a location outside building A and B. The application first calls the detectIndoorEnvironment() in the Localization Asistant, and the result should be false. The application can then use GPS result directly or call its outdoor localization algorithm with the outdoor signals, for example, the smart lampposts. The application calls the getOutdoorSignal() in the API Manager, and, in this case, it returns an empty array since no outdoor signals are available. Therefore, the application uses the GPS result directly.

10.5 In-Outdoor Transition

10.5.1 Indoor to Outdoor

This is similar to switching mode. Say the user is moving from building B to outdoors. The detectSwitchCondition() should return "outdoor" at some point. When this happens, the application switches to outdoor localization like in Section 10.4

10.5.2 Outdoor to Indoor

When the application is in outdoor environments, it should call detectIndoorEnvironment() periodically. Say the user enters building B. The detectIndoorEnvironment() should return

true at some point. When this happens, steps 1-6 in operation mode 2/3 in the previous section are processed to locate the user in building B.

Bibliography

[1] Rubén Cantarero Navarro et al. "A Proposal for Modeling Indoor–Outdoor Spaces through IndoorGML, Open Location Code and OpenStreetMap". In: *ISPRS International Journal of Geo-Information* 9.3 (2020). ISSN: 2220-9964. URL: https://www.mdpi.com/2220-9964/9/3/169.

[2] *Geographic information — Positioning services (ISO 19116:2019)*. International Organization for Standardization. 2019.

[3] Thomas Gilbert et al. "Built environment data standards and their integration: an analysis of IFC, CityGML and LandInfra". In: 2020.

[4] *Information technology — Real time locating systems — Test and evaluation of localization and tracking systems (ISO/IEC 18305:2016)*. International Organization for Standardization. 2016.

[5] J. Jones M. amd Bradley and N. Sakimura. *JSON Web Token (JWT)*. May 2015.

[6] Tatjana Kutzner, Kanishk Chaturvedi, and Thomas Kolbe. "CityGML 3.0: New Functions Open Up New Applications". In: *PFG – Journal of Photogrammetry Remote Sensing and Geoinformation Science* 88 (Feb. 2020). DOI: 10.1007/s41064-020-00095-z.

[7] Ki-Joune Li et al. "Survey on Indoor Map Standards and Formats". In: *2019 International Conference on Indoor Positioning and Indoor Navigation (IPIN)*. 2019, pp. 1–8. DOI: 10.1109/IPIN.2019.8911796.

[8] Liu Liu et al. "Indoor navigation supported by the Industry Foundation Classes (IFC): A survey". In: *Automation in Construction* 121 (2021), p. 103436. ISSN: 0926-5805. DOI: https://doi.org/10.1016/j.autcon.2020.103436. URL: https://www.sciencedirect.com/science/article/pii/S0926580520310165.

[9] Marc-O. Löwner et al. “Proposal for a new LOD and multi-representation concept for CityGML”. In: *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences* (2016), pp. 3–12.

[10] *OGC City Geography Markup Language (CityGML) Encoding Standard version 2.0, OGC 12-019*. OGC. 2012.

[11] *OGC IMDF version 1.0.0, OGC 20-094*. OGC. 2021.

[12] *OGC IndoorGML version 1.0.3, OGC 14-005r5*. OGC. 2018.

[13] *OGC KML version 2.3, OGC 12-007r2*. OGC. 2015.

[14] *Tilesets: Google maps structure*. `https://www.microimages.com/documentation/TechGuides/78googleMapsStruc.pdf`. MicroImages, Inc.

[15] Jinjin Yan. “Seamless Navigation in Indoor and Outdoor based on 3D Spaces”. PhD thesis. Nov. 2020.

Revision History

Revision	Date	Author(s)	Description
0.1	22.04.06	Gary W.-H. Cheung, Peter Tsui, Mengyun Liu, S.-H. Gary Chan	First draft