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

Contents

[Abstract 2](#_Toc2090123994)

[Introduction 3](#_Toc1342984074)

[SDK 4](#_Toc566114450)

[MongoDB 5](#_Toc1227276541)

[Data Specifications 5](#_Toc299958752)

[Reference Design for the Grid Reference System 11](#_Toc1830714648)

[Web REST API Specifications 14](#_Toc785164043)

[Requesting Building and Localization Information 14](#_Toc850903047)

[Requesting Grids and Site Signals for Edge Localization (Mode 2) 18](#_Toc297239690)

[Uploading User signals and Requesting Latest Location for Cloud Localization (Mode 3) 20](#_Toc316950493)

[Requesting Site Signals for Outdoor Localization 21](#_Toc2105043317)

[Requesting Maps 25](#_Toc167421212)

[Token Management 27](#_Toc1308789311)

[Data Validation Tool 27](#_Toc611974961)

[System Structure Flow Diagram 28](#_Toc629478659)

[JWT specification 28](#_Toc973994355)

[Header 28](#_Toc1915531487)

[Payload 28](#_Toc384541001)

[Signature 28](#_Toc377650977)

[Generating the Token 28](#_Toc398110411)

[Validating the Token 29](#_Toc574849400)

[RSA Key pair (2048bits) 29](#_Toc1632810688)

[Storing and Validating Password in MongoDB 30](#_Toc1037767022)

[Data Format 30](#_Toc2085528595)

[Validating 30](#_Toc989667329)

# 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 indoors and GNSS for outdoor. Pervasive positioning can be realized provided that the 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.

This is the internal version of the standard specifications for developers to build the systems responsible for the platform operator.

# Introduction

This internal version of the standard specifications includes all the components that bridge site owners and applications together. The communication protocols, data organization for site signals and maps are described in the external version. Therefore, this document describes:

* The UbiLoc SDK
* The HBase of the lookup server
* The web REST API server of the lookup server
* The validation tool in the lookup server

# SDK

# MongoDB

## Data Specifications

|  |  |  |  |
| --- | --- | --- | --- |
| **SiteInfo** | | | |
| Field | Format | Detail | Source |
| BuildingID | string | CSUID provided by site owner , filled only if indoorSite=true | datapakage.zip->SiteInfo.json |
| OutdoorSiteID | string | CSUID-like, filled only if indoorSite=false |
| SiteAddress | string | portal address |
| SiteOwner | string | owner's description |
| Contacts | string | email address |
| IndoorSite | boolean | is it an indoor site (use an indoor algorithm or outdoor algorithm in this site) |
| CreateTime | date | create time | lookup server generate |
| UpdateTime | date | update time |

|  |  |  |  |
| --- | --- | --- | --- |
| **Building** | | | |
| Field | Format | Detail | Source |
| BuildingID | string | CSUID provided by site owner | datapakage.zip->Spatial Representation->Building.json |
| Name | string |  |
| MapDataID | array of string | each mapId is referencing to id of the Map table |
| FloorList | array of string | each element is a floor number referencing to a floor number of the Floor table |
| DefaultFloorID | string | the default floor to be shown |
| CreateTime | date | create time | lookup server generate |
| UpdateTime | date | update time |

|  |  |  |  |
| --- | --- | --- | --- |
| **Floor** | | | |
| Field | Format | Detail | Source |
| FloorID | string | **parentID** concat with **floorNo** | lookup server generate |
| FloorNo | string | a string following the format in section 6.2 representing the floor number name (string) | datapakage.zip->Spatial Representation->Floor.json |
| Name | string |  |
| MapDataID | array of string | each mapId is referencing to id of the Map table |
| ParentID | string | Id of the parent building |
| RegionList | array of string | each element is the region number referencing to a region number of the Region table |
| DefaultRegionID | string | the default region to be shown |
| CreateTime | date | create time | lookup server generate |
| UpdateTime | date | update time |

|  |  |  |  |
| --- | --- | --- | --- |
| **Region** | | | |
| Field | Format | Detail | Source |
| RegionID | string | **parentID** concat with **regionNo** | lookup server generate |
| RegionNo | string |  | datapakage.zip->Spatial Representation->FloorID->Region.json |
| Name | string |  |
| MapDataID | array of string | each mapId is referencing to id of the Map table |
| ParentID | string | Id of the parent floor |
| ConnectedRegions | array of json obj | JSON array [minLon, minLat, maxLon, maxLat], arrivalArea: JSON array [minLon, minLat, maxLon, maxLat], arrivalRegionId: string} |
| CreateTime | date | create time | lookup server generate |
| UpdateTime | date | update time |

|  |  |  |  |
| --- | --- | --- | --- |
| **Map** | | | |
| Field | Format | Detail | Source |
| MapID | string | UUID provided by site owner | datapakage.zip->Spatial Representation-> Map.json |
| MapFormat | string | format such as JPG, PNG, ... |
| GeodeticPoints | array of json obj | JSON obj {x, y, lon, lat}, each element is two pairs of points: one in the coordinate system of the map, one in physical world (lon, lat) |
| ~~Boundary~~ | ~~array of float~~ | ~~JSON [lon, lat],array of lat, lon representing the utmost boundary of map~~ |
| OccupiedGridId | Array of string | The occupied grid Ids of this map |
| AttachedPrimalSpaceID | string | Id of the attached space, siteId if outdoor |
| FileName | string |  |
| Validation | boolean | declaimer of this map is valid |
| Data | byte array | Compressed packages with map files are converted to byte array(binary data) | datapakage.zip->Mapfile.zip->binary data |
| CreateTime | date | create time | lookup server generate |
| UpdateTime | date | update time |

|  |  |  |  |
| --- | --- | --- | --- |
| **LocSetting** | | | |
| Field | Format | Detail | Source |
| BuildingID | string | filled if this is a building, a CSUID-like id provided by site owner | datapakage.zip->Site Signal->BuildingLocSetting.json |
| OutdoorSiteID | string | filled if this is an outdoor site, a CSUID-like id provided by site owner |
| ShareSiteSignal | boolean | share to lookup server or not (if yes, mode 2, 3 are available) |
| SiteSignalMode | array of string | available signal modes/types to be used such as “WIFIfingerprint”, “BLElocation” |
| ~~CloudLodSignalMode~~ | ~~array of string~~ | ~~for mode 1, signal modes/type to be uploaded~~ |
| Boundary | array of float | For buildings, it is a rough boundary for checking if this building site is inside a specific region;  For outdoor sites, it is the utmost boundary of this site, it is the same as the 'Boundary' in OutdoorSite.json  Format is:  [[lon1, lat1], [lon2, lat2], ...]  (will be outdated in v2, probably after Aug) |
| SiteAreaGridId | Array of string | The grid Ids of the supported area of this site (will be used in v2, after Aug probably) |
| RemoteCloudLocUploadURL | string | for mode 1, URL to upload user signals |
| RemoteCloudLocDownloadURL | string | for mode 1, URL to obtain location result |
| RemoteCloudSignalModeURL | string | For mode1, URL to obtain supported signal mode |
| RemoteSignalDownloadURL | string | for mode 0, server to download site signals |
| SiteSignalGridID | array of string | Exist only if ShareSiteSignal==True  The GridID of this building or outdoor site | Generated from site signals under Site Signals Folder |
| CreateTime | date | create time | lookup server generate |
| UpdateTime | date | update time |

|  |  |  |  |
| --- | --- | --- | --- |
| **OutdoorSite** | | | |
| Field | Format | Detail | Source |
| SiteID | string | a CSUID-like id provided by site owner | datapakage.zip->Spatial Representation-->OutdoorSite.json |
| Name | string |  |
| Boundary | array of float | the utmost boundary of this site  (will be outdated in v2, probably after Aug) |
| SiteAreaGridId | Array of string | The grid Ids of the supported area of this site (will be used in v2, after Aug probably) |
| MapDataID | string | maps in this site, referncing mapId in Map |
| CreateTime | date | create time | lookup server generate |
| UpdateTime | date | update time |

|  |  |  |  |
| --- | --- | --- | --- |
| **Fingerprint** | | | |
| Field | Format | Detail | Source |
| RPID | string | Version 4 UUID, Id of the reference point | lookup server generate |
| FloorID | string | FloorID, concat BuildingID with FloorNo |
| Latitude | float |  | datapakage.zip->  Site Signals -> <FloorNo>  -> <XXFingerprint>.txt |
| Longitude | float |  |
| WIFIRssVector | Json array of “mac:rssi” | WiFi rssi vector, [“mac:rssi”,...] | datapakage.zip->  Site Signals -> <FloorNo> ->  WiFIFingerprint.txt |
| BLERssVector | Json array of " “uuid:major:minor:rssi” | BLE rssi vector, [“uuid:major:minor:rssi”,...] | datapakage.zip->  Site Signals -> <FloorNo> ->  BLEFingerprint.txt |
| MagneticSignal | Json array of [mag\_x, mag\_y,mag\_z] | [“mag\_x,mag\_y,mag\_z”] | datapakage.zip->  Site Signals -> <FloorNo> ->  MagFingerprint.txt |
| **Fingerprint file entry** | | | | |
| Fingerprint filename | Entry Format | | | |
| WiFiFingerprint.txt | latitude,longitude,FloorNo|[”mac:rssi”,”mac:rssi”,......,”mac:rssi”] | | | |
| BLEFingerprint.txt | latitude,longitude,FloorNo|[”mac:rssi”,”mac:rssi”,......,”mac:rssi”] | | | |
| MagFingerprint | latitude,longitude,FloorNo|[”mag x,mag y,mag z”,......,”mag x,mag y,mag z”] | | | |

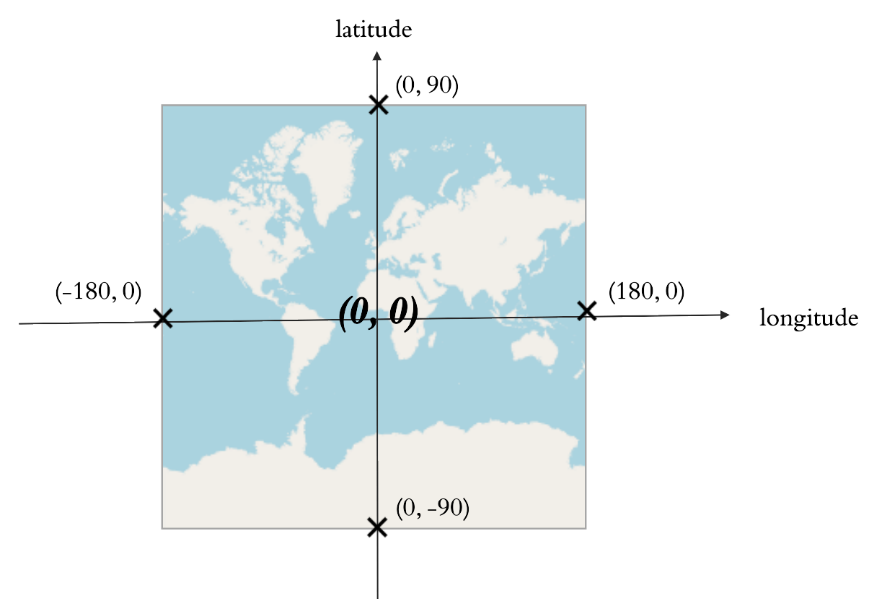
|  |  |  |  |
| --- | --- | --- | --- |
| **BLELocation** | | | |
| Field | Format | Detail | Source |
| BeaconID | string | The Id of this iBeacon in this standard. The format is: UUID (in hex string form) concatenates with Major and Minor (in 5 digits) | lookup server generate |
| Indoor | Boolean | Boolean Flag to indicate indoor beacon(true) / outdoor beacon(false) |
| FloorID | string | Exist only if Indoor==True  FloorID, concat BuildingID with FloorNo |
| SiteID | string | Exist only if Indoor==False  SiteID of this beacon |
| Latitude | float |  | datapakage.zip->  Site Signals -> <FloorNo>  ->BLELocation.txt |
| Longitude | float |  |
| UUID | string | The UUID of the beacon in the form  8-4-4-4-12 |
| Major | Integer | The major value of the beacon, rang-  ing from 1 to 65535 |
| Minor | Integer | The minor value of the beacon, rang-  ing from 1 to 65535 |
| **BLELocation file entry** | | | | |
| filename | Entry Format | | | |
| BLELocation.txt | latitude,longitude,FloorNo|UUID,major,minor | | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Grid** | | | |
| Field | Format | Detail | Source |
| GridID | string | Concatenation of zoom level, x-, y- index and floorId  \*We only store grids at zoom 20. For zoom levels 16-19, we aggregate the smallest grids (level 20) and return | lookup server generate |
| RPIDList | array of string | Array of RPID in this grid |
| BeaconList | array of string | Array of BeaconID in this grid |
| SiteID | string | Exist only if this is an outdoor grid since no SiteID info in GridID |
| ConnectedGridID | array of string | Array of connected GridID | Generated from Region.json files in datapackage.zip |

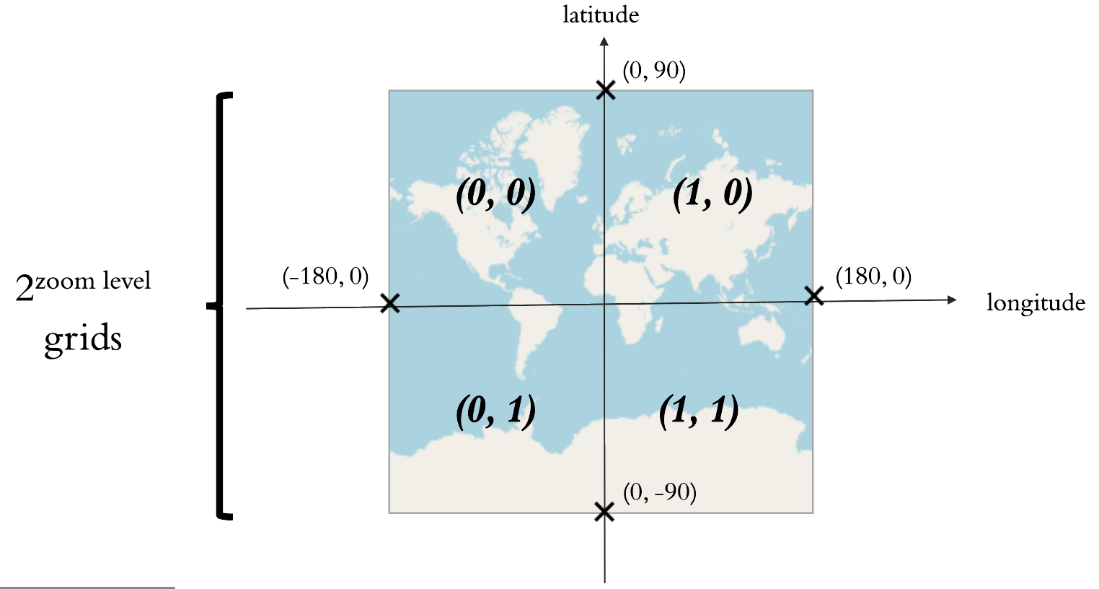
|  |  |  |  |
| --- | --- | --- | --- |
| **Account** | | | |
| Field | Format | Detail | Source |
| username | string | App Id | Registration by email. Probably hard code for now |
| salt | string | 8-byte salt to hash the password |
| hash | string | Hash of the password |
| email | string | Email of the app dev |

## Reference Design for the Grid Reference System

The grid reference system is like:

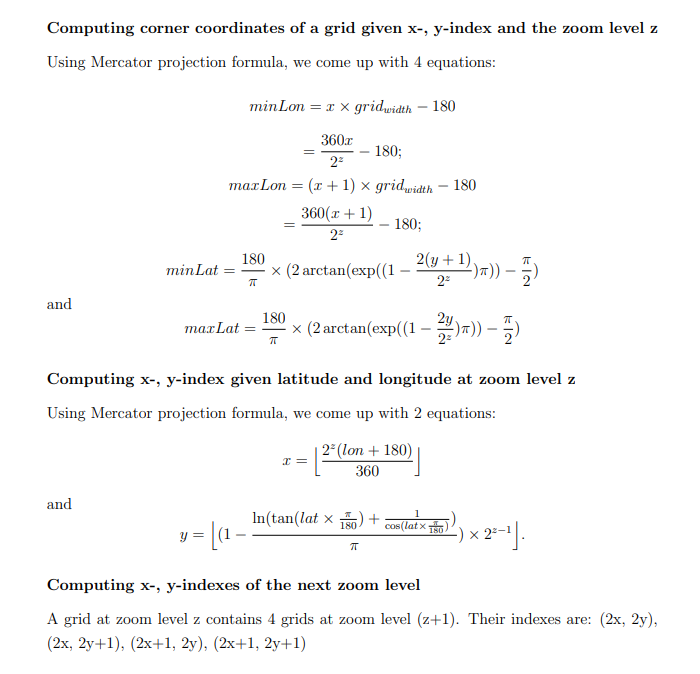


At zoom level 0



At zoom level 1

There are a few equations that are very useful when designing the grid reference system:



Then we can just store site signals in 20-zoom level grids. For example, for each grid, use a .txt to store the site signals as well as the connected gridIds. For example:

|  |
| --- |
| “connected\_grid\_ids”: [“connected grid id”, ...],  “fingerprints”: [  {  “rp\_id”: “RP Id”  “latitude”: “latitude”,  “longitude”: “longitude”,  “floor\_id”: ”floor Id”,  “wifi\_rss\_vector”: {“mac”: “rss value”, …},  “ble\_rss\_vector”: {“mac”: “rss value”, …},  “magnetic\_signal”: [“magnetic field strength along x-axis", “along y-axis", “along z-axis"]  }, …  ],  “beaconLocations”: [  {  “beacon\_id”: “beacon Id”,  “latitude”: “latitude”,  “longitude”: “longitude”,  “floor\_id”: “floor Id”  }, ...  ] |

You may also want to store the gridIds at all zoom level (16-20)

Upon requests on gridIds: return the gridIds

Upon requests on site signals with a gridId: calculate all 20-zoom level grids, aggregate them and return

# Web REST API Specifications

The web REST API server aims to support data queries from the SDK on building information, site signals, and maps. It is implemented in Python Flask due to high compatibility with the latest technologies and smaller size of the codebase. The following subsections describe the specifications of each API.

The server endpoint is: http://16.162.42.168/api

(last updated: 21/6/2022)

Note:

1. The spatial representation JSON file definitions can be found in external standard specification.
2. Binary data is sent in base64 format

## Requesting Building and Localization Information

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| GET | /buildingId-and-boundary/{longitude}/{latitude}/{accuracy} | | | | |
| Find the buildings inside or intersecting with the circle given a pair of latitude and longitude as the center and the accuracy as the radius. Return the buildingId and the boundary of these buildings. | | | | | |
| Parameters | | | | | |
| Name | | | Data type | Mandatory | Description |
| longitude | | | float | yes | The longitude |
| latitude | | | float | yes | The latitude |
| accuracy | | | float | yes | The estimated horizontal accuracy radius in meters |
| Response | | | | | |
| {  “data”: [  {  “buildingId”: "buildingId”,  “boundary”: [[lon1, lat1], [lon2, lat2], [lon3, lat3], ...],  },  ...  ]  } | | | | | |
| Name | | | Data type | Description | |
| buildingId | | | string | The buildingId of the building | |
| boundary | | | JSON array | The array of latitude and longitude of each vertex of the building boundary | |
| Implementation | | | | | |
| Reference：  <http://www.doc88.com/p-0897691589078.html>,  <https://www.docin.com/p-762635225.html>，  <https://www.jianshu.com/p/ba03c600a557>,    Algorithm for **Detect\_Intersection([lon1, lat1], [lon2, lat2])**:  {  [X1, Y1] = [lon1, lat1] – [longitude, latitude]  [X2, Y2] = [lon2, lat2] – [longitude, latitude]  R = accuracy  A = (X2 - X1) ^ 2 + (Y2 - Y1) ^ 2  B = X1 \* (X2 - X1) + Y1 \* (Y2 - Y1)  C = X1 ^ 2 + Y1 ^ 2 – R ^ 2  Delta = B \* B – A \* C  Intersection = False  Case 1: if (Delta <= 0)  {  //No intersect.  }  Case 2: if (Delta > 0)  {  t1 = (-B + sqrt(Delta)) / A; t2 = (-B – sqrt(Delta)) / A;  Case 2a: t1 <0, 0< t2.  {  Intersection = True  Break.  }  Case 2b: 0<= t1 <1.  {  Intersection = True  Break.  }  Case 2c: 1<= t1 <t2  {  //No intersect.  }  Return Intersection  }    Algorithm for **Check\_Inside(boundary):** //check if vertice within the polygon  {  Inside = False  //Draw a horizontal line to left and check num of edge intersect.  //Num is odd: Inside = true.  //Otherwise: Inside = False.  Return Inside  }    **Pseudocode:**  Data = []  For buildingId, boundary in DB:  {  Intersect = False  Inside = False  For num in (boundary.length - 1):  {  If(Detect\_Intersection(boundary[num], boundary[num+1])):  Intersect = true  }  If (Not Intersect)  {  Inside = Check\_Inside (boundary)  }  If (Intersect || Inside)  {  Data.Append (buildingId, boundary)  }  }  Return Data | | | | | |
| GET | /buildingId-and-gridId/{longitude}/{latitude}/{accuracy} | | | | |
| Return the buildings if one of their SiteAreaGridId contains the circle formed by the given lat, lon, acc | | | | | |
| Parameters | | | | | |
| Name | | Data type | | Mandatory | Description |
| longitude | | float | | yes | The longitude |
| latitude | | float | | yes | The latitude |
| accuracy | | float | | yes | The estimated horizontal accuracy radius in meters |
| Response | | | | | |
| {  “data”: [  {  “buildingId”: "buildingId”,  “gridId”: [gridId1, gridId2...],  },  ...  ]  } | | | | | |
| Name | | Data type | | Description | |
| buildingId | | string | | The buildingId | |
| gridId | | JSON array | | The array of SiteAreaGridId of this building | |
| Implementation | | | | | |
| Algorithm for detect grid that contains the circle:  2 cases:   1. One of the four corner of the grid is inside of the circle 2. One of the four corner of the largest square formed in circle is in the rectangle.     Algorithm for checking case 1:  Return (x – center\_x)^2 + (y – center\_y)^2 <= Radius of circle^2  Algorithm for checking case 2:  For pts in each square corner in circle:  {  If (pts.x < grid.max\_x && pts.x > grid.min\_x && pts.y < grid.max\_y && pts.y > grid.min\_y)  Return true  }  Data = []  For buildingId in DB:  For SiteAreaGridID in buildingId[“SiteAreaGridId”]:  IsIntersect = False  Extract zoomLevel, x ,y from GridID  Grid\_pts = Convert x y and zoomlevel to four corner in terms of longitude and latitude  if Check\_circle\_rectangles\_intersect\_grid(longitude,latitude,accuracy,x,y,zoomLevel):  isIntersect = True  For pts in Grid\_pts:  If Check\_Point\_inside\_circle(pts[0],pts[1],longitude,latitude,accuracy):  IsIntersect = True  if(isIntersect):  New['buildingId'] = building['BuildingID']  New['gridId'] = building['SiteAreaGridID']  Data.append(New)  Return Data | | | | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| GET | /building-loc-setting/{buildingId} | | | |
| Return the BuildingLocSetting JSON file of the building given a buildingId. | | | | |
| Parameters | | | | |
| Name | | Data type | Mandatory | Description |
| buildingId | | string | yes | The buildingId |
| Response | | | | |
| {  “data”: {  “BuildingLocSetting”: BuildingLocSetting.json,  }  } | | | | |
| Name | | Data type | Description | |
| BuildingLocSetting | | JSON object | The JSON object of the LocSetting specified in section 9 for indoors of the external standard specification | |
| Implementation | | | | |
| Grab data from DB according to buildingId  Return BuildingLocSetting | | | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| GET | /building-spatial-representation/{buildingId} | | | |
| Return the building spatial representation of the building given a buildingId | | | | |
| Parameters | | | | |
| Name | | Data type | Mandatory | Description |
| buildingId | | string | yes | The buildingId |
| Response | | | | |
| {  “data”: {  “buildingId”: buildingId,  “building”: Building.json,  “mapJson”: [Map.json files of the building layer],  “floors”: [  {  “floorId”: floorId,  “floor”: Floor.json,  “mapJson”: [Map.json files of this floor layer],  }, ...  ],  “regions”: [  {  “regionId”: regionId,  “region”: Region.json,  “mapJson”: [Map.json files of this region layer],  }, ...  ],  }  } | | | | |
| Name | | Data type | Description | |
| buildingId | | string | The buildingId | |
| building | | JSON object | The Building.json of the building | |
| mapJson | | Array of JSON object | All Map.json of the corresponding layer | |
| floors | | JSON array | The array of floors of this building | |
| floorId | | string | The floorId | |
| floor | | JSON object | The Floor.json of the floor | |
| regions | | JSON array | The array of regions of this building | |
| regionId | | string | The regionId | |
| region | | JSON object | The Region.json of the region | |
| Implementation | | | | |
| Return | | | | |

## Requesting Grids and Site Signals for Edge Localization (Mode 2)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| GET | /grid-id?buildingId={buildingId} &zoomLevel={zoomLevel} | | | |
| Return the gridIds of the building given the buildingId  If zoom level is given, return the gridIds in that zoom level | | | | |
| Parameters | | | | |
| Name | | Data type | Mandatory | Description |
| buildingId | | string | yes | The buildingId |
| zoomLevel | | string | no | The desired zoom level |
| Response | | | | |
| {  "gridIds”: [  “gridId\_1”, “gridId\_2”, ...  ]  } | | | | |
| Name | | Data type | Description | |
| gridIds | | JSON array | The array of gridIds | |
| Implementation | | | | |
| Return | | | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| GET | /grid/{gridId}?signalMode={signalMode} | | | |
| Return the site signals given the gridId and optionally the signal mode.  If signal mode is given, return the site signals in this grid with this mode | | | | |
| Parameters | | | | |
| Name | | Data type | Mandatory | Description |
| gridId | | string | yes | The gridId |
| signalMode | | string | no | If this is given, only return the site signals  with this mode. Signal mode tags are:  ”WiFiFingerprint”, ”BLEFingerprint”,  ”MagFingerprint”, ”BLELocation” |
| Response | | | | |
| {  "connectedGridIds": ["connected grid id", ...],  "fingerprints": [  {  "rpId": "RP Id"  "latitude": "latitude",  "longitude": "longitude",  "floorId": "floor Id",  "wifiRssVector": [“mac:rssi\_value”, ...],  "bleRssVector": [“uuid:major:minor:rssi\_value”, ...],  "magneticSignal": ["magnetic field strength along x-axis", "along y-axis", "along z-axis"]  }, ...  ],  "beaconLocations": [  {  "beaconId": "beacon Id",  "latitude": "latitude",  "longitude": "longitude",  "floorId": "floor Id"  }, ...  ]  } | | | | |
| Name | | Data type | Description | |
| connectedGridIds | | JSON array | The array of gridIds that are connected to this grid | |
| fingerprints | | JSON object | The same format defined in Section 6.1 in external spec | |
| beaconLocations | | JSON object | The same format defined in Section 6.1 in external spec | |
| Implementation | | | | |
| Return | | | | |

## Uploading User signals and Requesting Latest Location for Cloud Localization (Mode 3)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| POST | /upload-user-signal/ | | | |
| Upload user signals to the lookup server | | | | |
| Parameters | | | | |
| Name | | Data type | Mandatory | Description |
| userId | | string | yes | The userId |
| wifiRssVector | | JSON array | no | The array of scanned AP information. Each element is in format: {“mac”, “rssi”, “freq”, “timestamp”} |
| bleRssVector | | JSON array | no | The array of scanned beacon information. Each element is in format: {“uuid”, “major”, “minor” “rssi”, “txPower”, “timestamp”} |
| Response | | | | |
| no | | | | |
| Implementation | | | | |
| Pass the userId and user signal to the location computing server host in <link to be provided later> | | | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| GET | /user-location/userId/ | | | |
| Return the BuildingLocSetting JSON file of the building given a buildingId. | | | | |
| Parameters | | | | |
| Name | | Data type | Mandatory | Description |
| userId | | string | yes | The userId |
| Response | | | | |
| {  “data”: {  “inBuilding”: True/False,  “latitude”: “latitude”,  “longitude”: “longitude”,  “floorId”: “floorId”,  }  } | | | | |
| Name | | Data type | Description | |
| inBuidling | | Boolean | Boolean indicating if user is in building supported by the lookup server | |
| latitude | | float | The latitude of the user location | |
| longitude | | float | The longitude of the user location | |
| floorId | | string | The floorId of the user location | |
| Implementation | | | | |
| Return the user latest location. (need to forward only) | | | | |

## Requesting Site Signals for Outdoor Localization

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| GET | /outdoor-siteId-and-boundary/{minLon}/{minLat}/{maxLon}/{maxLat} | | | |
| Find the outdoor sites inside the rectangular boundary given two pairs of latitudes and longitudes as the boundaries. Return the siteIds and boundaries of the outdoor sites inside. | | | | |
| Parameters | | | | |
| Name | | Data type | Mandatory | Description |
| minLat | | float | yes | The bottom boundary of the rectangle in latitude |
| minLon | | float | yes | The left boundary of the rectangle in longitude |
| maxLat | | float | yes | The upper boundary of the rectangle in latitude |
| maxLon | | float | yes | The right boundary of the rectangle in longitude |
| Response | | | | |
| {  “data”: [  {  “siteId”: siteId,  “boundary”: [[lon1, lat1], [lon2, lat2], [lon3, lat3]), ...]  }  ...  ]  } | | | | |
| Name | | Data type | Description | |
| siteId | | string | The siteId | |
| boundary | | JSON array | The array of latitude and longitude of each vertex of the site boundary | |
| Implementation | | | | |
| Reference:  https://www.cnblogs.com/kfarvid/archive/2011/06/27/2091400.html.    Algorithm for **Check\_Inside([lon, lat], boundary):** //check if vertice within the polygon  {  Inside = False  //Draw a horizontal line to left and check num of edge intersect.  //Num is odd: Inside = true.  //Otherwise: Inside = False.  Return Inside  }  Algorithm for **Check\_Inside([lon, lat]):** //Check if vertice inside Rect  {  Inside = False  If (minLon<lon<maxLon && minLat<lat<maxLat):  {  Inside = True  }  Return Inside  }    Algorithm for **Check\_Rect\_Line\_H ([lon1, lat1], [lon2, lat2], Lat):** // Horizontal line of Rect  {  If (lat1>Lat && lat2>Lat): //both points over line  {  Return False  }  If (lat1<Lat && lat2<Lat): //both points below line  {  Return False  }  If (lat1 == lat2): // parallel  {  If (lat1 == Lat): // at same line  {  If (lon1<minLon && lon2<minLon): //on the left  {  Return False  }  If (lon1>maxLon && lon2>maxLon): //on the right  {  Return False  }  Return True  }  Else // not at same line  {  Return False  }  }  // not parallel  Lon = (lon2-lon1)\*(Lat-lat1) / (lat2 – lat1) + lon1 //intersect point’s lon  Return ((Lon >= minLon) && (Lon <= maxLon))  }    Algorithm for **Check\_Rect\_Line\_V ([lon1, lat1], [lon2, lat2], Lon):** // Vertical line of Rect  {  If (lon1<Lon && lon2<Lon): //both points before line  {  Return False  }  If (lon1>Lon && lon2>Lon): //both points behind line  {  Return False  }  If (lon1 == lon2): // parallel  {  If (lon1 == Lon): // at same line  {  If (lat1<minLat && lat2<minLat): //on the upper  {  Return False  }  If (lat1>maxLat && lat2>maxLat): //on the bottom  {  Return False  }  Return True  }  Else: // not at same line  {  Return False  }  }  // not parallel  Lat = (lat2-lat1)\*(Lon-lon1) / (lon2 – lon1) + lat1 //intersect point’s lat  Return ((Lat >= minLat) && (Lat <= maxLat))  }    Algorithm for **Check\_Rect\_Line ([lon1, lat1], [lon2, lat2]):**  {  Result = False  If (Check\_Inside([lon1, lat1]) || Check\_Inside([lon2, lat2])): //line inside Rect  {  Result = True  }  Else: //check intersect with Rect’s 4 edges  {  Result |= Check\_Rect\_Line\_H([lon1, lat1], [lon2, lat2], minLat)  Result |= Check\_Rect\_Line\_H([lon1, lat1], [lon2, lat2], maxLat)  Result |= Check\_Rect\_Line\_V([lon1, lat1], [lon2, lat2], minLon)  Result |= Check\_Rect\_Line\_V([lon1, lat1], [lon2, lat2], maxLon)  }  Return Result  }    Algorithm for **Boundary\_Intersect\_Rect (boundary):** //check polygon intersect Rect  {  If (boundary.length<2 || minLon == maxLon || minLat == maxLat):  {  Return False  }  For num in boundary.length:  {  If (Check\_Rect\_Line(boundary[num], boundary[num+1]):  {  Return True  }  }  Return False  }    **Pseudocode:**  Data = []  For siteId, boundary in DB:  {  Intersect = Boundary\_Intersect\_Rect (boundary)  Inside = False;  If (Not Intersect)  {  Inside |= Check\_Inside ([minLon, minLat], boundary)  Inside |= Check\_Inside ([minLon, maxLat], boundary)  Inside |= Check\_Inside ([maxLon, minLat], boundary)  Inside |= Check\_Inside ([maxLon, maxLat], boundary)  }  If (Intersect || Inside)  {  Data.Append (siteId, boundary)  }  }  Return Data | | | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| GET | /outdoor-loc-setting/{siteId} | | | |
| Return the LocSetting given the siteId | | | | |
| Parameters | | | | |
| Name | | Data type | Mandatory | Description |
| siteId | | string | yes | The siteId |
| Response | | | | |
| {  “data”: {  “OutdoorLocSetting”: OutdoorLocSetting.json  }  } | | | | |
| Name | | Data type | Description | |
| OutdoorLocSetting | | JSON object | The JSON object of the LocSetting specified in section 9 for outdoors of the external standard specification | |
| Implementation | | | | |
|  | | | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| GET | /outdoor-grid/{gridId}/{siteId}?signalMode={signalMode} | | | |
| Return the site signals given the gridId, buildingId, and the optional query parameters:  If no query parameter is given, return all site signals in this grid.  If signal mode is given, return the site signals in this grid with this mode | | | | |
| Parameters | | | | |
| Name | | Data type | Mandatory | Description |
| gridId | | string | yes | The gridId |
| siteId | | string | yes | The siteId |
| signalMode | | string | no | If this is given, only return the site signals  with this mode. Signal mode tags are:  ”WiFiFingerprint”, ”BLEFingerprint”,  ”MagFingerprint”, ”BLELocation” |
| Response | | | | |
| {  "connectedGridIds": ["connected grid id", ...],  "fingerprints": [  {  "rpId": "RP Id"  "latitude": "latitude",  "longitude": "longitude",  "floorId": "floor Id",  "wifiRssVector": {"mac": "rss value", ...},  "bleRssVector": {"mac": "rss value", ...}, (need to modify)  "magneticSignal": ["magnetic field strength along x-axis", "along y-axis", "along z-axis"]  }, ...  ],  "beaconLocations": [  {  "beaconId": "beacon Id",  "latitude": "latitude",  "longitude": "longitude",  "floorId": "floor Id"  }, ...  ]  } | | | | |
| Name | | Data type | Description | |
| connectedGridIds | | JSON array | The array of gridIds that are connected to this grid | |
| fingerprints | | JSON object | The same format defined in Section 6.1 in external spec | |
| beaconLocations | | JSON object | The same format defined in Section 6.1 in external spec | |
| Implementation | | | | |
| Return | | | | |

## Requesting Maps

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| GET | /map-metadata?buildingId={buildingId}&floorId={floorId}&regionId={regionId}&outdoorSiteId={outdoorSiteId}&lat={latitude}&lon={longitude} | | | |
| Return the map metadata given the query parameters:  If buildingId/ floorId/ regionId/ outdoorSiteId is given, return the Map.json files of the building/ floor/ region/ site  ~~If latitude AND longitude are given, find all maps whose boundary contain the request point, and return all the Map.json files of them.~~  If latitude AND longitude are given, find all maps whose OccupiedGridId contain the request point, and return all the Map.json files of them.  If multiple parameters are given, fulfill only the leftmost one. That is, the result will only depend on the leftmost parameter and ignore the rest.  If no data is available, no “mapJson” will be returned. That is, the return value is {data: []} | | | | |
| Parameters | | | | |
| Name | | Data type | Mandatory | Description |
| buildingId | | string | no | The buildingId |
| floorId | | string | no | The floorId |
| regionId | | string | no | The regionId |
| outdoorSiteId | | string | no | The outdoorSiteId |
| lat | | float | no | The latitude |
| lon | | float | no | The longitude |
| Response | | | | |
| {  “data”: {  “mapJson”: [Map.json]  }  } | | | | |
| Name | | Data type | Description | |
| mapJson | | Array of JSON object | All Map.json files that satisfy the query | |
| Implementation | | | | |
|  | | | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| GET | /map/{mapId} | | | |
| Return the map data given a mapId | | | | |
| Parameters | | | | |
| Name | | Data type | Mandatory | Description |
| mapId | | string | yes | The mapId |
| Response | | | | |
| {  “data”: {  “mapData”: map data  }  } | | | | |
| Name | | Data type | Description | |
| mapData | | byte array | The byte array of the map data in base64 format | |
| Implementation | | | | |
|  | | | | |

## Token Management

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| POST | /generate-token | | | |
| Check if the account information is correct. If yes, generate and return the JWT. Steps to generate a token are in section JWT Specification. | | | | |
| Parameters | | | | |
| Name | | Data type | Mandatory | Description |
| username | | string | yes | The username |
| password | | string | yes | The password |
| Response | | | | |
| {  “data”: {  “success”: success,  “token”: token  }  } | | | | |
| Name | | Data type | Description | |
| success | | boolean | True if the account information is correct, false otherwise | |
| token | | string | JWT, specified in section JWT specification.  Exist only if success == true | |
| Implementation | | | | |
|  | | | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| POST | /refresh-token | | | |
| Check if the token is valid. If yes, refresh the token by generating a new token with new “exp” and “iat” attributes | | | | |
| Parameters | | | | |
| Name | | Data type | Mandatory | Description |
| token | | string | yes | The token to be refreshed |
| Response | | | | |
| {  “data”: {  “success”: success,  “token”: token  }  } | | | | |
| Name | | Data type | Description | |
| success | | boolean | True if the input token is valid, false otherwise | |
| token | | string | JWT, specified in section JWT specification.  Exist only if success == true | |
| Implementation | | | | |
|  | | | | |

# Data Validation Tool

An API in the lookup server for checking the integrity of the uploaded data\_package.zip in section 9 of the external specification.

# System Structure Flow Diagram

Link: <https://app.diagrams.net/#G1j3hgnHOGBUfpIpi6WLfWYo0w7BqYS8Kt>

# JWT specification

The JSON Web Token (JWT) is used to authenticate and authorize users when calling APIs, and it can be used in cross-domain calls. In our case, it can provide auth for applications to obtain location services from site owner’s servers and the lookup server. Detailed specifications are in the official documentation [<https://datatracker.ietf.org/doc/html/rfc7519>]. In this section, we define the necessary claims in our case.

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.

## Header

|  |  |  |
| --- | --- | --- |
| Attribute | Description | Value |
| tpy | Type of this json object which is the JWT | “JWT” |
| alg | Algorithm used in the signature | “RS256" |

## Payload

|  |  |  |
| --- | --- | --- |
| Attribute | Description | Value |
| 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” as the plaintext and k as the key. Therefore, when other site owners validate the token, they compute the plaintext “header.payload” by RS256 using ciphertext as input and the public key of the lookup server.

## Generating the Token

Denote the private key held by the lookup server as k\_pri

Steps:

1. Encode the header and the payload in Base64url, denoted as H and P
2. Compute the digital signature S = RS256(H+’.’+P, k\_pri)
3. The token is H+’.’+P+’.’+S

## Validating the Token

Denote the private key held by the lookup server as k\_pub

Steps:

1. Compute the plaintext H+’.’+P from RS256(token, k\_pub)
2. Transform the plaintext from Base64url to JSON objects
3. Check if every attribute is valid. If at least one attribute is invalid, the token is invalid. Else it is valid.

## RSA Key pair (2048bits)

Private Key:

-----BEGIN RSA PRIVATE KEY-----

MIIEowIBAAKCAQEAs4xkpwTPqnICaD4+f7W/Uu7rrkxsZELVR9mxiFcJsgMr37ut

Tl0gNwl4R0tMHmCXNDOTWiQ6fEOr4ssDjpyeq1zldbMebR5I1MUQSV+y5IaLJSJF

JlxHzp4xgWYauVT5lKUwjf11TJ+pDyOV4femTztHL8DjgUQXuwld1q9BF4ZqYoZD

0k6rTBZT09ev6jC9H0oB7qp1c984HHbaznLzUzKK0b4QC0+r41Z9alwkA+1vUgwI

TiIki9LMbiPHBKiaAcWnHLAnjC7rqR+rxyWEpJQ5wr4TAIAGdEmc27qNOjX0gbet

3vR8ZZV26tCxGHpSFiGG5DLgOROizOJ9esIfHQIDAQABAoIBAAaLhpvEBCHteAL+

yd5z7J6EURkQdh8NhKLS37LdoV/Y17+XItEpeeQntze6GfM9iCWB/8wYZ/1hpiFl

TyLiuLH8d+zbc698nCENcfOKxoWoFehroGOul3uDrsBNzYmV/GcbwnJam+0IIOyJ

WoxbL7fRrF2kRWbP6aGKzN3nk5Lc5tmXvIaoav5NZ6MZB0HI/Ni0SndQtxz4o5UY

u2oZhja0eOebHBLvwvnqYFrEn+8yMG8m+uRlQleVg+QYbf62mob4gptJ9J1BTk8p

u8yZI3Hd2WHs02bXfHgWwkBAJZyGKqLsew648hZSDTSDwkkDjI8BaayaN1ODKc7w

sKSgE4ECgYEA406NKvAW4tbHyPjr2T8XGTRc4iGwcYiBvL+TsLn9noPfeDJNIDfS

97BHbgfqXl+ThT9sSb0YDjZ0UBlTNdCPt/SMvTfc+MOsIOTEMpz13l+V3ut8TUtM

irTNeUaBzpWvrduIYYysUcCuLGt9G43fFw2IFpG3B9uFREynazZ9xqECgYEAyjaF

rO75xqqg9Ae/wjihIJNBp0pU8kWeXDVP+2CMA+GTMeeOi/M5NC4y7dQkHVD4Hm+z

IpyYop6uKQEXET3kPiVULVFQyDXtdfR3cg6LzPPzZeb8nthcYmWpiZibqohCNqxH

u2wbSbD/WF4BGBWAKl1dQBl80TcItYB/sm8Nkv0CgYAU1aNc1yrR2evwANjlTcIZ

PlJxK7NXhLKXkVvBLBsSB3ZZNwsYs+UzbMXvUOxrgAkugnSzgZCN5PHeUaPt39L8

gn7P7Snk51KEpVNSbE6g9lPvXdhD4W3/nLjkr9DAFbpHjkK+5jz5NVcBnUyXVPao

z0W1A+Ap6TYa34fM7O60YQKBgQC2N0td+jq5+yfuvszdh4oqgIe46BKD/pglHyVj

sCx7ktnBLpgVJKd3jen2xiOFB2GOi7kwh0xDvhnSeVkAAHWcSqb/7zaMM4w3roO0

E8eGeI3sValuuikFwfZ0N0CO+xovii9ZhKFSVlzClHMH8Nf9VQSirlT7ckw2wgRQ

FiJ1QQKBgHgp0+6Wbonb2UlxDhYZHC1eHKibVdOEuAWPC8jeUqT5B6E/P4Rr2QP+

q8UUR7TAH1i52eZixLmjvuIckK+ZGNOoIGKTjTqyF56cWG3KYREgKv2WeyzoVi7p

QAl/Lf+cHan0Xh9xuSLQSKE7i0HzgKXYNmVd7TEE48qdr3RV98rG

-----END RSA PRIVATE KEY-----

Public Key:

-----BEGIN PUBLIC KEY-----

MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAs4xkpwTPqnICaD4+f7W/

Uu7rrkxsZELVR9mxiFcJsgMr37utTl0gNwl4R0tMHmCXNDOTWiQ6fEOr4ssDjpye

q1zldbMebR5I1MUQSV+y5IaLJSJFJlxHzp4xgWYauVT5lKUwjf11TJ+pDyOV4fem

TztHL8DjgUQXuwld1q9BF4ZqYoZD0k6rTBZT09ev6jC9H0oB7qp1c984HHbaznLz

UzKK0b4QC0+r41Z9alwkA+1vUgwITiIki9LMbiPHBKiaAcWnHLAnjC7rqR+rxyWE

pJQ5wr4TAIAGdEmc27qNOjX0gbet3vR8ZZV26tCxGHpSFiGG5DLgOROizOJ9esIf

HQIDAQAB

-----END PUBLIC KEY-----

Cipher type: “RSASSA-PKCS1-v1\_5 with SHA-256" Or “RS256” in short

# Storing and Validating Password in MongoDB

Our aim is to verify a user given a username and a plaintext password. Since storing the plaintext password is really bad in terms of security, we store the hash of the password and a salt that is used when hashing. In our design, each entry of the Account table has a username, a salt, a hash, and an email-address. “username”, “salt”, and “hash” will be used to verify the user.

## Data Format

“salt” is a randomly generated 8-byte data, and “hash” is the base64 of the result of SHA-256(plaintext\_password + salt)

## Validating

Given a username and a password, get the salt and the hash from the database. Then compute SHA-256(password + salt) and compare it to the hash. If they are the same, the user is verified, otherwise not.

# Version 2 Updates

## Reduce Site Owners’ Confusions

### Site and Map Boundaries Represented as Grids Instead

Why – Site owners are not technical people. Often, they don’t know how many points they should include in boundaries.

How – Replace boundaries with grids (grid Id). A few of clicks can define the site and map coverage for querying purposes.

### Simplification of data\_package.zip

Why – There are multiple duplicated items, and some of them can be generated by us, for instance, the “parentID” of Floor.json. Entering them multiple times gain no advantages but human input mistakes so better not.

How – The definitions of files in data\_package.zip will be revised and simplified to reduce site owners’ effort.

### Rework on Region Transition Area

Why – Just like site and map boundaries, the v1 region transition area seems to be too hard for site owners to decide.

How – Grids is not the best answer this time. Instead, site owners may want more intuitive “things” to present their site, for example, doors, elevators, etc. Rework on the region transition area is needed. More objects are to be defined in layman terms to reduce the complexity for site owners inputting their data.

## More Expression Power

### Map constraints

Why – Many localization algorithms make good use of the map constraints by specifying where the user can be (the user cannot be in the air, right?). This increases the location accuracy significantly. The map constraints can be standardized also.

How – The “coordinates” of GeoJson MultiPolygon (an example: https://geojson.org/geojson-spec.html#id7) will be used to present map constraints in the Region layer/ Map layer

### Multiple Buildings Supported in One Site

Why – Each site may have more than one building IRL as shown in 1st demo.

How – Revision on spatial representation and loc setting are needed.

### Supported Floors

Why – Site owners may or may not support the localization services of the whole building. It can be a subset of all the floors.

How – A new attribute to present the supported floors will be added in LocSetting.json. The hierarchical structure of site signals will also be revised.

## For Application Developers

### More Reference Designs

Why – As the standard provides more and more flexibilities to site owners and application developers, too many options for prevailing app dev to choose may result in them getting lose when using the standard.

How – More reference designs can be given to provide a minimal way to obtain a location. Nite that it is not a must for every app dev. For advanced app dev, they can use their own designs.

### Localization SDK API definition

Why – To support localization SDK developed for specific sites, a common set of APIs is essential to reduce app dev’s effort

How – Definition of a set of APIs that are necessary in a localization SDK, with standardized data classes.