Open Location Platform - Data User's Guide

User Guide

Version 2.3

Legal Notices

Trademark Acknowledgements

Disclaimer

Document Information

Product	
Name:	Open Location Platform - Data User's Guide
Version:	Version 2.3

Document	
Name:	Open Location Platform - Data User's Guide
ID:	529f33b2-1484-431e-b725-180a30e7934f
Status:	FINAL
Date:	2019-02-21T21:07:00.695Z

Table of Contents

- Introduction
- Concepts
  - Catalogs
  - Layers
  - Schemas
  - Partitions
  - HERE Resource Names
  - Data Limits and Cost
  - Data Security
- How To
  - Catalogs
    - Create a Catalog
    - Delete a Catalog
    - Share a Catalog
    - Edit Metadata
    - Work With GeoJSON Data
  - Layers
    - Create a Layer
      - Stream Layer Settings
      - Versioned Layer Settings
      - Volatile Layer Settings
      - Index Layer Settings
    - Reconfigure a Layer
  - Schemas
    - Update a Schema
    - Browse Schemas
    - Create a Schema
    - Share a Schema
    - Delete a Schema

Introduction

In the HERE Open Location Platform, data consists of both maps and location information that HERE provides, such as Real-Time Traffic and Weather, as well as data that you and other users provide. The HERE Open Location Platform Portal provides access to many activities related to data, including discovering, managing, and visualizing the data.

For the terms and conditions covering this documentation, see the HERE Documentation License.

Discover Data

The HERE Open Location Platform Portal is where you can start exploring data. The Portal helps business analysts, data scientists, and developers to explore data by browsing, filtering, and searching for catalogs as well as layers. Click **Data** in the navigation bar to see a list of catalogs and layers that you have permission to read. When you find data of interest, you can perform a deeper analysis using Notebooks.

A *catalog* is a collection of layers that are logically managed as a single set. Click on a catalog to view its details. The catalog details page presents all the catalog metadata, such as description, tags, data coverage, creation date, HERE Resource Name (HRN), and a list of the layers in the catalog. For more information, see Catalogs.

A *layer* contains semantically related data and can be overlaid spatially to construct a complete digital map. For example, a catalog might contain a layer which contains road sign data and another layer containing road topology data. A layer can be used in one or more catalogs. For more information, see Layers.

Each layer is divided into partitions. A *partition* divides the data in a layer into reasonable units for caching and processing. For more information, see Partitions.

The Portal contains rich metadata about both catalogs and layers. This metadata shows the origin, provider, coverage, and freshness of the data. You can view geographic data coverage on a map and, for certain data types, view the data itself on a map. You can also examine data more closely by reviewing and downloading schemas and decoded data. In summary, the Portal is the best path for exploring data when you are not sure which data you are interested in using.

Manage Data

Apart from data discovery, you can use the Portal to create and manage different data assets. Catalogs, layers, and pipelines can all be created, configured, managed, and shared using the Portal. To create schemas you must use the schema archetype in the HERE Open Location Platform SDK.

Note

You must have the HERE Workspace Plan to create pipelines and schemas.

Visualize Data

The Portal is a useful tool for investigating or visualizing data processing output on a map. Click **Data** and navigate to the data layer, then review the layers details page. On this page, you can also see the list of partitions in the layer. The partitions are geographic tiles if latitude/longitude information is included.

Other Ways of Working With Data

Aside from the Portal, the HERE Open Location Platform provides other ways of working with data.

Command Line Interface

The command line interface (CLI) is part of the HERE Open Location Platform SDK. It offers the same data management functions that are available in the Portal (such as creating catalogs and layers, listing catalogs and layers, getting metadata) but the HERE Open Location Platform CLI enables you to execute these commands from the command line. Further, commands can be stored in scripts so you can run repetitive commands easily. The HERE Open Location Platform CLI will most likely be your interface of choice when working with the HERE Open Location Platform for this reason.

The CLI is useful for:

- Developers who want programmatic access to the functions available in the Portal
- Developers and administrators who want to automate data management tasks in scripts

For more information, see the HERE Open Location Platform CLI Guide.

Note

You must have the HERE Workspace Plan to access the CLI.

Data Client Library

The Data Client Library is a Java/Scala library you can use to read and write data to the HERE Open Location Platform. It provides a layer of abstraction from the REST APIs which shields you from changes that may occur to HERE Open Location Platform APIs as they evolve. The Data Client Library also reduces the complexity involved in reading and writing data, reducing the time it takes to add HERE Open Location Platform data interaction to your applications.

The Data Client Library is useful for:

- Developers who want to incorporate HERE Open Location Platform data into applications written in Java or Scala
- Developers who want to write less code than is required by the REST APIs

For more information, see the Data Client Library Developer's Guide Guide.

Note

If you are a Marketplace user and you do not have the HERE Workspace Plan, you do not have access to the SDK. See the *Marketplace Consumer User's Guide* or the *Marketplace Provider User's Guide* for instructions on downloading the Data Client Library.

Data API

The Data API is a REST API that provides access to data and data management functions. The Data API is useful for:

- Developers who want to incorporate HERE Open Location Platform data into applications written in something other than Java or Scala
- Developers who want access to the HERE Open Location Platform APIs

For more information, see the Data API Developer's Guide.

Catalogs

The HERE Open Location Platform stores data in catalogs. A catalog is a collection of data that is managed as a single set. Catalogs contain layers that represent different types of data. In the case of map data, layers can be overlaid spatially to construct a complete digital map. For example, a catalog may contain a map that includes layers for road attributes, topology, and signs. While catalogs often contain geospatial data, catalogs can contain any kind of data.

A catalog can contain any combination of layer types. For more information, see Layers.

The image below shows the structure of catalogs.

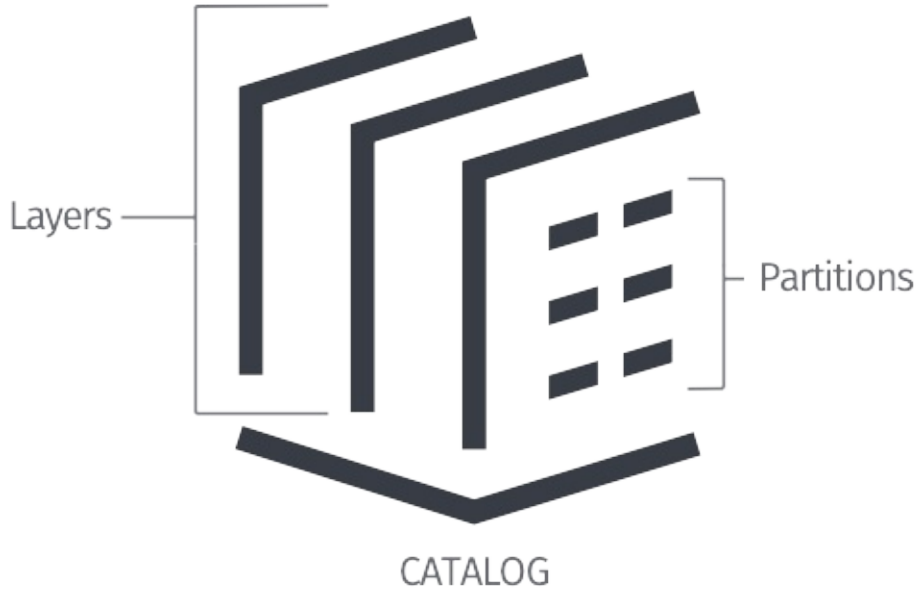


Figure: Catalog Structure

Data sharing is controlled at the catalog level. You can share an entire catalog, including all its layers.

Note

- For information about limits and cost considerations, see Data Limits and Cost.
- For information about data security and durability, see Data Security and Durability.

Layers

A layer is a set of partitions of a specific data type, functional property, and structure. You can use layers to segment data based on semantics. For example, in a catalog you can have one layer for road signs and another layer for road topology. You can also use layers to segment data based on schema. Layers can be overlaid spatially to construct a complete digital map.

There are four types of data layers: versioned, volatile, index, and stream.

Note

- For information about limits and cost considerations, see Data Limits and Cost.
- For information about data security and durability, see Data Security and Durability.

Versioned Layers

A versioned layer stores slowly-changing data that must remain logically consistent with other layers in the catalog. When you want to update a catalog of versioned layers, all the layers related to the update (and partitions within a layer) must be updated in one publication so that they can be versioned together. For example, the HERE Map Content catalog contains several versioned layers, including Topology (road topology), Road (road attributes), and Place (points of interest). In each version of the catalog, these layers represent a consistent view of the world at that point in time. If a new road is built and there are new buildings containing new businesses along the new road, all three layers would need to be updated together in one publication so that in the new version of the catalog the Topology layer contains the new road, the Road layer contains the attributes for the new road, and the Place layer contains the names of the new businesses. If only the Place layer were to be updated with the new businesses, the layers would no longer represent a consistent view of the world because the Topology and Road layers would be missing the new road.

To achieve consistency between layers, any update that affects multiple layers must be published together in a publication. Updating multiple partitions of a versioned layer also happens in a publication to preserve the consistency and integrity of intra-layer and inter-layer references. A new catalog version is available only when all layers have been updated and the publication has been finalized.

Note

It is the data publisher's responsibility to ensure that a publication results in a consistent set of layers that accurately represent the world. Extensive support is provided to produce consistent content for versioned layers in the HERE Open Location Platform Data Validation Library. You must have the HERE Workspace Plan to use the Data Validation Library.

You can access data as it existed at different points in time by referencing the version you want. Once a version has been published, the data in that version cannot be changed and can be removed only by removing the whole catalog version. Data within a version is immutable and consistent.

How Data is Versioned

The initial version of a catalog, before any data has been published to it, is -1. When data in a versioned layer is updated:

- The catalog version is incremented by one
- All layers and partitions that are updated in the publication have their versions updated to match the new catalog version
- All layers and partitions that are not updated retain their existing version number
- Layers and partitions that are not updated keep their existing version numbers

It is important to note that only those layers and partitions that are updated have their version updated to the catalog's new version number. So, the version of a layer or partition represents the catalog version in which the layer or partition was last updated.

Requesting a Version

When you request a particular version of data from a versioned layer, the partition that gets returned may have a lower version number than you requested. The following example illustrates this concept:

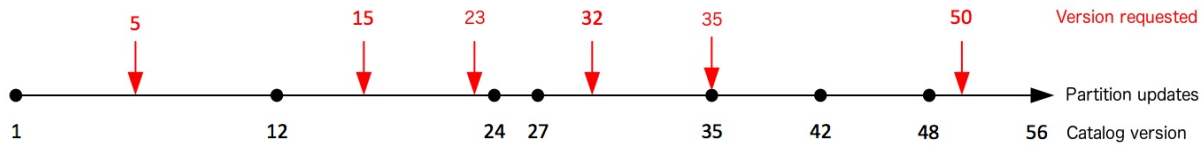


Figure: Image showing an example of partition versions returned for different catalog versions

The red arrows show requests for data from particular versions. The dots along the line represent changes to a partition over time. The partition has been updated at catalog versions 1, 12, 24, 27, 35, 42, and 48. The current catalog version is 56.

The table below shows which partition version is returned for requests to different catalog versions.

REQUESTED VERSION	PARTITION VERSION
5	1
15	12
23	12
32	27
35	35
50	48

Volatile Layers

A volatile layer is a key/value store where values for a given key can change and only the latest value is retrievable. As new data is published, old data is overwritten.

Volatile layers use in-memory storage. Storing data in memory helps reduce data access latency and provides applications with consistently high throughput.

Consider using a volatile layer when you don't need older versions of the data. For example, if you want to make the latest weather information available, you could use a volatile layer to store the latest observations. When new weather observations are written to the layer, the old one is overwritten so that only the latest observations are

available for data consumers.

Another use for a volatile layer is as a cache for applications requiring fast response times and consistently high throughput. When running complex, time-consuming computations, it is valuable to cache the computation results for future use. Correctly caching values that are accessed frequently not only reduces the load on the rest of the components in the cloud but also helps speed up responses to other clients requesting the same data. For example, say a client application performs a complex query that requires fetching data from five different versioned layers, sorting the data, matching the data, and running statistical analysis on the data to compute an optimized parameter. If it is likely that another client will request the same query, then the result of this complex query is a good candidate for caching in a volatile layer. This way, millions of clients can benefit from fast response times when submitting the same request. At the same time, the load on versioned layers and pipelines is significantly reduced.

Index Layers

An index layer is part of an overall solution that enables you to index and store metadata and data in a way that is optimized for batch processing. The index layer itself is, as its name suggests, an index of the catalog's data by attributes that you can later query. For example, if you want to run a batch process daily to find all pothole detection events recorded that day in the area surrounding a given city, you can use an index layer to index the pothole detection events by event time, event type, and location. You can then query the data every 24 hours for pothole events in the area of the city as part of your batch process. Index layers provide the flexible solution you need to easily store attributes of pothole events along with the location/time the event took place.

Like versioned layers, index layers are useful when you want to access historical data. The difference is that an index layer can be used when you do not need to maintain logical consistency across layer versions in the way versioned layers do. The other difference is that you can define your own attributes by which you want to index and query the data, whereas you cannot define your own attributes in versioned layers.

You can use an index layer in combination with a pipeline to append and get late events by event time. This ability to properly handle late events is important when your end user devices are online and offline at different times and where batches of data sent can include events with varying timestamps that need to be indexed appropriately with other events already received.

Index layers work in combination with the Data Archiving Library, which is available in the SDK. For more information about the Data Archiving Library, see the Data Archiving Library Developer's Guide.

Stream Layers

A stream layer is a queue that streams data to data consumers in real time. Consumers read the data in the order it is added to the queue. Once a consumer reads the data, the data is no longer available to that consumer, but the data remains available to other consumers.

Stream layers can be configured with a retention time, or time-to-live (TTL) which results in unconsumed data being removed after a specified period of time.

An example use of a stream layer is to handle data from vehicle sensors.

Schemas

Schemas define the organization of data in each partition of a layer, both the structure of the data and its content. In the HERE Open Location Platform, schemas are defined using Protocol Buffers. If you are reading or writing data to a layer, use the layer's schema to understand the structure of the data you receive, and the way your data should be structured to write it to the layer. Schemas are also used by pipelines to operate on data.

If you are creating your own catalog and layers, including a schema for each layer enables you to share data with others by defining how others should consume the data.

Changing Schemas

Once you assign a schema to a layer the only change you can make to the schema assignment is to change the schema to a newer minor version or patch version of the same schema.

When changing the schema assigned to a layer, you cannot:

- Change to a different schema
- Upgrade to a newer major version
- Downgrade to a prior version
- Change the schema to "None"

For example, if you assign the schema RT Traffic 1.1.0 to a layer, you can update the layer to a newer minor version such as 1.2.0 or a newer patch version such as 1.1.1. You cannot upgrade it to 2.0.0 or downgrade it to 1.0.0.

The HERE Open Location Platform includes services that enable you to create, store, and share schemas, as well as leverage HERE-provided schemas. To view schemas, log in to the Portal, click **Data** then click **Browse Schemas**.

HERE Schemas

HERE provides several complex schemas which are used in HERE data and are available for you to use with your own data layers. Examples of HERE schemas include Sensor Data Ingestion Interface (SDII), Weather, Real Time Traffic, and several map data schemas. Using these schemas provides advantages, including:

- Standardization: HERE schemas enable users with similar data to leverage a standard format, reducing disparities in what is provided, and providing common attribute nomenclature.
- Sharing: Data sharing between users and systems is made easier with HERE schemas in the following ways. First, some functions like showing data coverage on a map or visualizing event-level details may work more smoothly when you use platform schemas. Second, standardized descriptions included in HERE schemas help users better understand the data and how to read it.

User-Defined Schemas

User-defined schemas are ones that describe the structure of data in user-created catalogs. To create a custom schema, download the SDK and leverage the Schema Archetype project. Schemas are held to the same privacy standards as the data itself, which means that all schemas and data are private by default. Only the schema creator can access the schema until it is shared or used within the HERE Open Location Platform to operate on the data or visualize the data on a map.

Note

You must have the HERE Workspace Plan to create a schema.

Schema-less Data

It is possible to ingest schema-less data into the HERE Open Location Platform. Schema-less data may be appropriate if the data is only used by the data producer and never shared. Also, if you are working with GeoJSON data, you do not need a schema.

Ingesting schema-less data is also useful if you want to reduce the initial overhead of getting your data into the HERE Open Location Platform. You can ingest the data without a schema, then transform the data using a schema later.

Partitions

Catalogs are divided into layers which in turn are divided into partitions, making partitions the smallest unit of data in the system. Partitions are key values within a catalog and layers are a namespace for partitions. This makes the catalog (with layer and partition) a key-key-value store.

Partitions can store any binary data. The HERE Open Location Platform does not need to understand the structure of the data in a partition since it does not need to decode it. Data is published and retrieved without modification. However, you can define the structure and encoding of the data by associating a schema with the layer. This defines the data structure and encoding for the partitions in the layer so that data producers and consumers know how to encode and decode the data.

Partitions are named according to one of two partitioning schemes:

- Generic partitioning
- HERE tile partitioning

There are a couple reasons why understanding partitioning schemes is important. First, when you are creating a catalog, you need to select the partitioning scheme to use for the catalog, and you need to choose the one best suited to the data you are going to store in the catalog. Second, when you are querying a catalog or publishing data to a catalog, you need to know the partitioning scheme used so that you can understand how to reference partitions by their name.

Generic Partitioning

Generic partitioning is the simplest form of partitioning. Partition names have no semantic meaning. Generic partitioning is best suited to data other than map data, such as search index data.

HERE Tile Partitioning

HERE Tile partitioning is a method for storing map data. In HERE Tile partitioning, layers contain rectangular geographic tiles that represent an area of the map. These tiles are also known as partitions. If you use HERE Tile partitioning, you can take advantage of the HERE Open Location Platform libraries and APIs to perform geo-related tasks.

To use HERE Tile partitioning, you need to know how map data is partitioned so that you can read and write data.

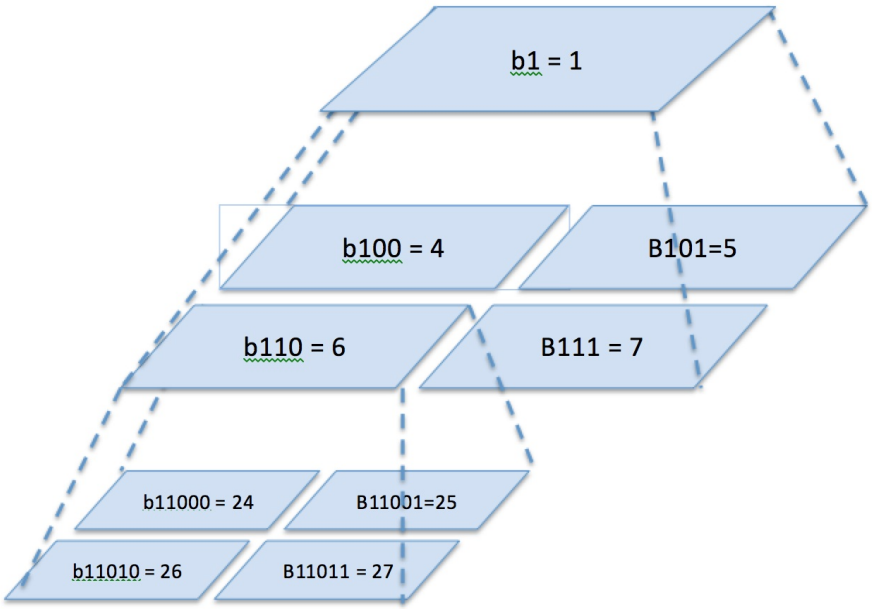
Map Tiling

The process of dividing map data into partitions is called tiling. The HERE Tile tiling scheme is based on quadtrees. A quadtree is a tree data structure in which each internal node has exactly four children. Quadtrees partition a two-dimensional space by recursively subdividing it into four tiles. The child tiles are numbered 0-3 in a fixed reverse "Z" pattern:

- Tile 0 is the southwest sub-tile
- Tile 1 is the southeast sub-tile
- Tile 2 is the northwest sub-tile
- Tile 3 is the northeast sub-tile

Tile Level

A tile's level refers to how many tiles were subdivided to produce the tile. For example, in the following diagram, tile 4 (`100` in binary) is at level 1 and tile 24 (`11000` in binary) is at level 2.



The maximum tile level is 31. At this level, each tile is approximately 0.0187 meters square near the equator.

HERE Tile tiling is based on raw, non-projected WGS84 latitude/longitude coordinate values, so each child tile covers exactly half its parent's latitude/longitude range per side.

Note

This scheme results in non-square tiles when viewed on a common Mercator projected 2D map, with the effect more pronounced further from the equator.

HERE Tile IDs

Each tile in the map has an identifier called a HERE Tile ID. A HERE Tile ID is a 64-bit unsigned integer computed from the tile's quadkey. A quadkey is a string of numbers (0-3) which captures the hierarchy of parent-child tiles from level 1 to the target tile level.

For example, for the level 5 tile containing San Francisco in the map below, the quadkey would be `02123` because the parent tile is `0212`, and the child tile containing San Francisco is `3`:

In this second example, the quadkey for the child tile containing the Berlin Hauptbahnhof (central train station) is `122012031202200` because the parent tile's quadkey is `12201203120220` and the quadkey of the child tile containing the Berlin Hauptbahnhof is `0`.

You can determine the level of a tile by the number of digits in the quadkey. For example, the quadkey for a level 14 tile will have 14 digits.

To determine a tile's HERE Tile ID from its quadkey, use the following algorithm. To illustrate the algorithm, the example values use the level 14 tile of Berlin as shown above.

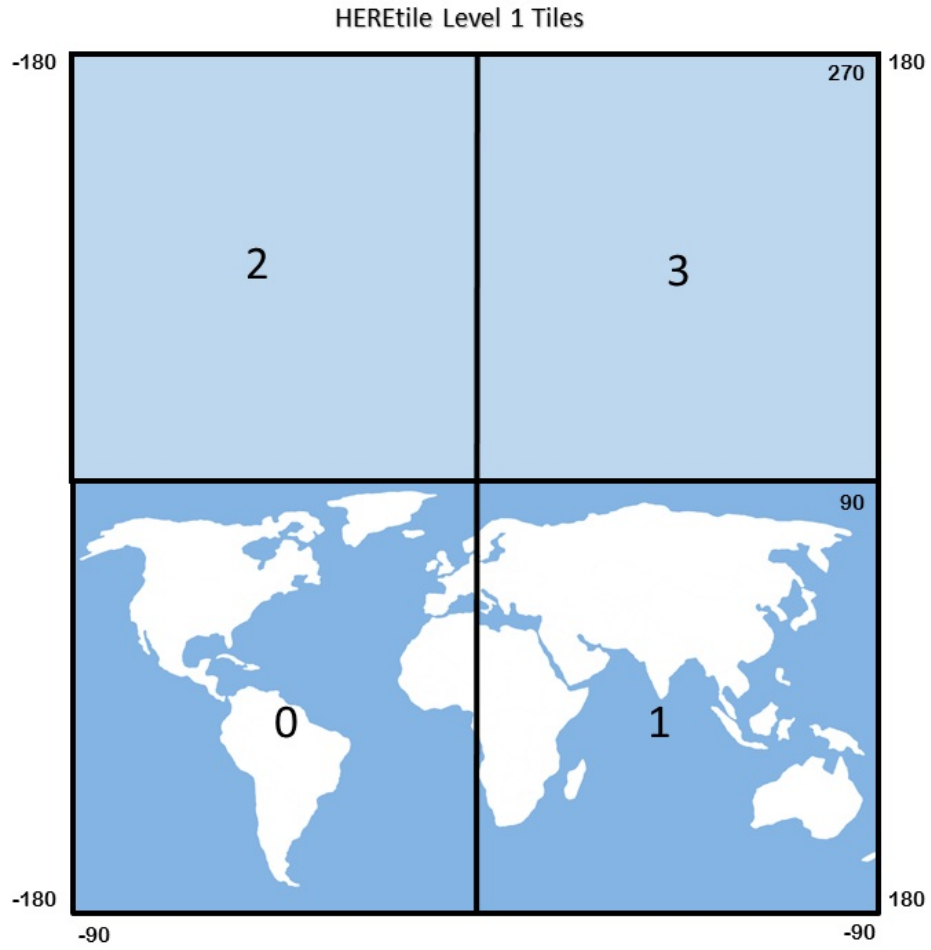
1. Prepend the quadkey with a 1:
   12201203120220 = 112201203120220
2. Convert the quadkey from base 4 to base 10:
   $112201203120220_4 = 377894440_{10}$
3. The resulting base 10 number is the HERE Tile ID:
   377894440

Note

HERE Location Libraries provide a `TileResolver` that you can use to calculate HERE Tile IDs. If you are using Scala or Java, you can use the `mapquad` library for the same purpose.

HERE Tile Coordinate Ranges

The normal coordinate range on a world map is -180° to +180° longitude and -90° to +90° latitude. However, in HERE Tile partitioning, the level 0 root tile representing the entire world is augmented with a virtual counterpart north of the North Pole. This is done to avoid special handling of level 1 tiles. As a result, the base coordinate range in HERE Tile partitioning is -180° to +180° longitude and -90° to +270° latitude, making the level 0 world tile a square with sides of 360°. From here, the root world tile is split in the standard quadtree way into four tiles at level 1, resulting in tiles 0 and 1 covering the world and tiles 2 and 3 generally unused.



Note the following special cases:

- Longitude values of +180° are converted to -180°, so tile references "wrap" over the anti-meridian.
- Latitude values of +90° are owned by their southern tiles.

Calculating Latitude/Longitude Degrees for a Tile Level

The latitude and longitude range for a tile can be calculated as:

$360° / 2^{\text{tile level}}$

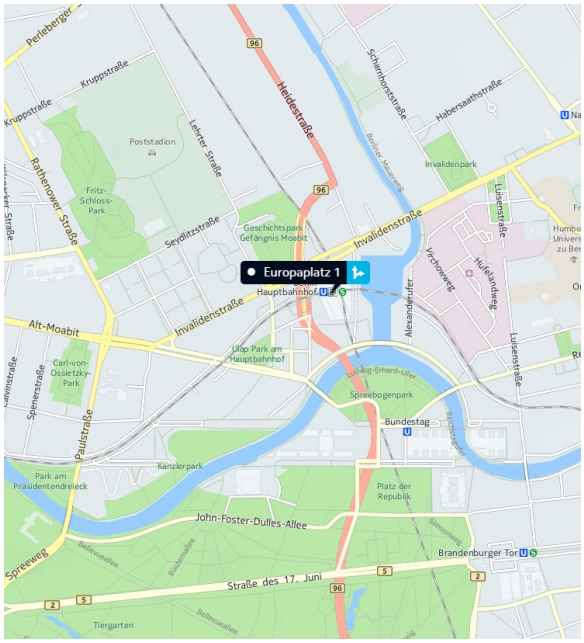
So for level 14 tiles, the latitude/longitude range would be calculated as:

$360° / 2^{14} = 360° / 16384 = 0.02197265625°$ per tile

Tiling schemes always result in a question of which tile contains a latitude/longitude location that lies on a tile boarder. For the HERE Tile scheme, locations lying on the south-west border of a tile belong to that tile.

Identifying the HERE Tile ID for a Latitude/Longitude

The tile HERE Tile ID for any latitude/longitude position at a given tile level can be calculated algorithmically using a version of Morton coding. Take this example for the Berlin Hauptbahnhof (central train station) at latitude/longitude coordinates 52.52507/13.36937.



Let's calculate the level 14 HERE Tile ID for this location. First, we need to calculate the desired tile's X,Y coordinates on the world map. Tile X,Y coordinates are not latitude/longitude values, they are the tile's integral positional coordinates, indexed from (0,0) in the southwest corner of the world map.

1. Find the horizontal (X) tile index from the longitude value by dividing the world map longitude range (-180° to +180°) into tile-sized ranges based on the desired tile level. As described in HERE Tile Coordinate Ranges, each level 14 tile covers 0.02197265625 degrees per tile:
   180° + 13.36937° = 193.36937° absolute longitude offset from south-west corner
   193.36937° / 0.02197265625° = 8,800.45 = **tile X: 8,800** (round down for 0-based indexing)
2. Find the vertical (Y) tile index, making sure to use the latitude range -90° to +270° as described in HERE Tile Coordinate Ranges.
   90° + 52.52507° = 142.52507° absolute latitude from the south-west corner
   142.52507° /0.02197265625° = 6,486.47 = **tile Y: 6,486** (round down for 0-based indexing)
3. Convert the tile X, Y indexes (8800, 6486) and tile level (14) into a Morton code quadkey. To do this, take the simple binary representation of the tile coordinate indexes, zero-padded to the number of bits in the tile level:
   Tile X coordinate: 8800 = 10001001100000_2 (already 14 bits)
   Tile Y coordinate: 6486 = 01100101010110_2 (zero-padded to 14 bits)
4. Interleave the bits of the binary values, starting with the first bit of the Y-coordinate:
   Interleaved Y/X = 01101000011000110110001010002
5. Convert the resulting binary value to a base 4 integer to get the quadkey string:
   $01101000011000110110001010002 = 12201203120220_4$

So, the quadkey for the Berlin Hauptbahnhof (central train station) at latitude/longitude coordinates 52.52507/13.36937 is **12201203120220**. Here is the quadkey on the map:

The final step is to encode the tile's quadkey as a HERE Tile ID:

1. Prepend the quadkey with a 1:
   12201203120220 = 112201203120220
2. Convert the quadkey from base 4 to base 10:
   $112201203120220_4 = 377894440_{10}$
3. The resulting base 10 number is the HERE Tile ID for the latitude/longitude coordinates 52.52507/13.36937:
   377894440

HERE Resource Names

A HERE Resource Name (HRN) is a unique identifier for resources such as catalogs, schemas, and pipelines. Here's an example of an HRN:

```
hrn:here:data::olp-traffic-1
```

The HRN is generated by the HERE Open Location Platform when the resource is created. Its structure can vary, so you should not try to parse HRNs or infer any meaning from them. Once a resource is created you cannot change the resource's HRN.

Data Limits and Cost

To ensure good performance, the HERE Open Location Platform has limits on data storage and throughput. Some limits can be controlled by layer configuration, which may impact your cost since you are charged based on how you have configured the layers and data usage. As a general rule, the more data you send and receive from the HERE Open Location Platform, and the more data you store, the more you will be charged. You will also be charged more if you configure layers for for higher performance.

You are charged for your use of the HERE Open Location Platform based on how you have configured the layers and data usage.

Versioned Layers

Limits

- Maximum partition size: 50 GB
- We recommend that you do not exceed 24 MB/second when publishing metadata

Cost Considerations

- Usage
  - You are charged for the amount of metadata and data stored in the layer
  - You are charged based on the amount of data you read and write using the `blob` and `metadata` APIs
- Configuration
  - There are no layer configuration settings that affect cost

Volatile Layers

Limits

- Maximum partition size: 2 MB
- Maximum throughput: Determined by the package type configured for the layer. The larger the package size, the greater the throughput.

Cost Considerations

- Usage
  - You are charged based on the amount of data you read and write using the `blob` API
- Configuration
  - You are charged based on the package type configured for the layer. For small, medium, and large package types, your cost triples because of redundant data storage. Experimental packages do not store data redundantly so you are charged for a single instance only.
  - You are charged based on the retention setting configured for the layer

Stream Layers

Limits

- We recommend that messages be smaller than 1 MB for best performance.
- Maximum throughput: Determined by the the maximum throughput setting configured for the layer. Catalogs in the OLP Marketplace have a maximum outbound throughput of 2 megabytes per second (MBps).
- The maximum amount of data that can be concurrently stored in a stream layer can be calculated as follows: `(inbound throughput) x (retention time)`.

Cost Considerations

- Usage
  - Usage does not affect cost
- Configuration
  - You are charged based on the maximum throughput configured for the layer
  - You are charged based on the retention setting configured for the layer

Index Layers

Limits

- It is recommended to group messages with the same indexing attributes before indexing instead of storing metadata for many small files in index layer.
- The maximum number of metadata records that can be returned by a single query is 100,000. You may submit multiple queries to query a larger set of results.

Cost Considerations

- Usage
  - You are charged for the amount of metadata and data stored in the layer
  - You are charged based on the amount of data you read and write using the `blob` and `index` APIs
- Configuration
  - You are charged based on the retention setting configured for the layer

Data Security and Durability

The HERE Open Location Platform protects your data through security and durability practices.

Security

The HERE Open Location Platform utilizes industry-standard data security best practices to protect your data:

- Data stored at rest in versioned layers, stream layers and index layers is encrypted using AES-256, a strong, proven, block cipher. This data protection includes data which has been persisted per the Time to Live (TTL) setting. Data stored in volatile layers is not encrypted.
- Data in transit between OLP and your applications is encrypted using the TLS 1.2 cryptographic protocol and the strong AES-256-GCM cipher.
- Within OLP, data in transit is also encrypted using TLS 1.2. Additional or different protection mechanisms are employed as needed.
- HERE secures the OLP website and API endpoints with trusted certificates issued by a well-known Certificate Authority (CA) and signed using a SHA-256 algorithm.

Durability

Your data is protected from loss due to corruption or system failure. The degree of durability depends on the layer type.

Versioned Layers

Versioned layers are designed to provide 99.999999998% durability of data (both blob data and metadata) over a given year. This durability level corresponds to an average annual expected loss of 0.000000002% of data partitions. For example, if you store 10,000,000 partitions in OLP, you can on average expect to incur a loss of two partitions once every 10,000 years. While OLP data is currently located in a single region (EU-West), Versioned data is stored redundantly on multiple devices across a minimum of three independent network and power domains within that region.

Index Layers

Index layers are designed to provide 99.999999998% durability of data over a given year. This durability level corresponds to an average annual expected loss of 0.000000002% of data partitions. For example, if you store 10,000,000 partitions in OLP, you can on average expect to incur a loss of two partitions once every 10,000 years. While OLP data is currently located in a single region (EU-West), Index data is stored redundantly on multiple devices across a minimum of three independent network and power domains within that region.

Volatile Layers

Volatile data is temporal. Existing data is overwritten every time new data is written to a partition. For the limited time that volatile data (both blob data and metadata) is stored, it is stored redundantly on multiple devices across a minimum of three independent network and power domains within a single region (EU-West). Failure of one device would be recovered by another.

Stream Layers

Stream data is replicated across multiple devices and across three independent network and power domains within a single region (EU-West). Failure of one device will be recovered by another. Additionally, stream data is always written to an underlying filesystem. You can set how long this data is retained in the filesystem by using the TTL (Time To Live) setting. A best practice is to configure the stream data TTL long enough to ensure that data is not dropped in the event of a consumer group interruption (e.g. a pipeline restart) and while corrective actions are taken.

In addition to these data redundancy measures inside OLP, the recommended best practice is to ensure regularly tested backups exist. Secure data backups, in the form of one or more duplicate catalogs, can be assigned a narrower set of permissions to further limit who can delete those backups.

Create a Catalog

If you have data you want to bring to the HERE Open Location Platform, you need to create a catalog to organize the data. A catalog is a collection of data that is logically managed as a single set. Catalogs contain layers that represent different types of data and that can be overlaid spatially to construct a complete digital map. For example, a catalog may contain a map that includes layers for road attributes, topology, and signs.

To create a catalog using the Portal:

1. Click **Data**.
2. Click **Add Catalog** then click **New Catalog**.
3. Define the catalog's metadata by filling in the following fields.

   The information you provide in these fields helps you and others find and understand the data in the catalog. We recommend giving careful thought to the metadata you provide so that it is as easy as possible to find and share catalogs effectively. For examples of metadata, see the HERE-provided catalogs in the Portal.

   - Catalog Name: A user-friendly name for the catalog. This is the name that is displayed when browsing catalogs.
   - Catalog ID: The ID to use when referring to this catalog programatically. Catalog IDs must be unique across all catalogs in the HERE Open Location Platform. This ID will be part of the catalog's HERE Resource Name (HRN).

   ### Note: Catalog IDs

   Catalog IDs are publicly visible since they are part of the catalog HRN. Do not include private or company confidential information when you specify a catalog ID. Catalog names are private by default, which means you can add your private or confidential information in this field.

   - Catalog Summary: A brief summary of the catalog.
   - Catalog Description: A detailed description of the catalog and what it contains.
   - Tags: Keywords that help find the catalog when searching in the Portal.
4. Click **Save**.

   It may take a few minutes to create the catalog. You will see a notification banner when the catalog is created.

   Once the catalog is created you can add layers to the catalog. You can either create a new layer in the catalog or add a layer from another catalog. For more information, see:

   - Creating a Layer
   - Adding a Layer from Another Catalog

   By default, the user who creates a catalog has manage, read, and write permissions. For information about giving other users access to the catalog, see Sharing a Catalog.

   ### Note

   It may take several minutes for a new catalog to appear in the list of catalogs returned by the Data Client Library, CLI, and REST API.

Delete a Catalog

You can delete a catalog if you have *manage* permissions for the catalog. This action deletes the catalog and all layers and partitions within that catalog.

Warning

Deleting a catalog is permanent and cannot be undone.

To delete a catalog using the Portal:

1. Click **Data**.
2. Browse to the catalog that you want delete and select the catalog to open it.
3. Click **More > Delete catalog**.

Note

It may take several minutes for a deleted catalog to be removed from the list of catalogs returned by the Data Client Library, CLI, and REST API.

Share a Catalog

You can share catalogs with users, apps, and groups. You must have *manage* permission to the catalog in order to share it.

1. In the HERE Open Location Platform Portal, click **Data**.
2. Select the catalog you want to share.
3. Click the **Sharing** tab. If you do not see this tab, you do not have *manage* permission and cannot share the catalog.
4. Specify the app, user, or group with whom you want to share the catalog.
5. Choose the permissions to grant to the app, user, or group:
    - **read** - Grants the ability to get data and metadata from the catalog.
    - **write** - Grants the ability to publish data and metadata to the catalog.
    - **manage** - Grants the ability to change catalog settings, including sharing. Also grants the ability to change the settings of layers in the catalog.
6. Click **Grant**.

Edit Catalog Metadata

Catalog metadata can be edited by anyone with *manage* permissions for the catalog.

Hint

This topic describes how to edit catalog metadata using the HERE Open Location Platform Portal. You can also edit catalog metadata using the REST API. For more information about editing catalog metadata using REST, see the Data API Developer's Guide.

To edit catalog metadata using the Portal:

1. Click **Data**.
2. Browse to the catalog that you want edit and click the catalog to open it.
3. Click **More > Edit info**.

For information about the metadata fields, see Creating a Catalog.

Work With GeoJSON Data

The HERE Open Location Platform supports the use of GeoJSON data. GeoJSON is a format for encoding geographic data using JSON. The GeoJSON format defines a structure for geometries, features, and feature collections of feature objects with geographic geometries such as Points, LineStrings, and Polygons, and free-form name:value properties. The GeoJSON specification is maintained by the Internet Engineering Task Force (IETF). For complete details, see the GeoJSON specification.

Note

Older versions of the GeoJSON standard supported custom coordinate systems. The HERE Open Location Platform only supports the World Geodetic System 1984 (WGS84) coordinate system.

Configure a Layer for GeoJSON Data

When creating a layer for GeoJSON data, configure the layer as follows:

- The partitioning scheme must be HERE Tile.
- The layer type must be volatile or versioned.
- The content type must be application/vnd.geo+json.

Partition GeoJSON Data

A GeoJSON `FeatureCollection` contains multiple feature objects, each of which has some geometry tying it to the map. A single FeatureCollection might have millions of feature objects spread over the whole world. If the `FeatureCollection` is very large or the feature objects are spread over a large geographic area, you may want to divide the `FeatureCollection` into multiple `FeatureCollection` objects, each stored in its own HERE Tile partition.

The decision of when to divide a large `FeatureCollection` depends on what you plan to do with the data and how complex individual feature objects are. Some tasks can be done with millions of features per tile, while others can be slow even with thousands of features. As a general rule for visualizing data, one partition should have less than 20,000 features, or less than 5,000 if they are markers (Point features without a Radius).

One way to divide a large `FeatureCollection` into multiple `FeatureCollection` objects is based on the center point (centroid) of each feature. To do this, you iterate over the `FeatureCollection` and calculate the centroid of each feature. Then, you map each centroid to the HERE Tile which overlaps that point. In the end, all feature objects are mapped to a HERE Tile. Each resulting HERE Tile contains a single `FeatureCollection` which in turn contains all features whose centroid overlaps that particular HERE Tile. In this way a large `FeatureCollection` becomes multiple `FeatureCollection` objects which can be processed in parallel.

You can also divide a large `FeatureCollection` based on the first coordinate of the feature geometry rather than the centroid.

Custom GeoJSON Style Properties

The HERE Open Location Platform supports several custom properties that you can use to add visual styling to features when rendered in the Portal or using the Visualization Library. These properties are not part of the GeoJSON specification, but you can add them to the `properties` member of any `features` object, which is designed to take custom properties. You can add the following properties:

- `tooltip` renders a pop-up message when you mouse over on the feature. If the `tooltip` property is not set, the pop-up is displayed with the list of all other configured `properties` for the feature.
- You can use `style.color` and `style.fill` to modify the feature color. You can set the color in any format that is supported by THREE.Color(): `0xff0000`, `rgb(255, 0, 0)`, `rgb(100%, 0%, 0%)`, `skyblue`, `hsl(0, 100%, 50%)`.
- `style.width` is a property intended for `LineString` and `MultiLineString`. It defines the width of the line measured in pixels. The default width is 1px.
- `radius` is a property intended for `Point` and `MultiPoint`. It defines the radius for the circle measured in meters.

Note that when using `MultiPoint`, the defined properties will be the same for all points. The same rule applies to `MultiLineString` and `MultiPolygon`.

For more information, see the Visualization Library Developer's Guide.

Note

You must have the HERE Workspace Plan to use the Visualization Library.

Create a Layer

Once you have created a catalog, you can create or add layers to it. A layer is a set of partitions of a specific data type, functional property, and structure. You can use layers to segment data based on semantics. For example, in a catalog you can have one layer for road signs and another layer for road topology. You can also use layers to segment data based on version schema. Layers can be overlaid spatially to construct a complete digital map. For more information, see Layers.

When creating a layer, consider what layers you need based on what data you want to ingest and how you want to logically organize that data.

To create a layer using the Portal:

1. Click **Data**.
2. Browse to the catalog that you want to contain the new layer and select the catalog to open it.
3. Click **Add New Layer**.
4. Define the metadata for the layer by filling in the following fields.
    - Layer Name: A user-friendly name for the layer that is displayed when browsing layers.
    - Layer ID: The ID to use when referring to this layer programmatically. Layer IDs must be unique within the catalog.
    - Layer Summary: A brief description of the data in the layer.
    - Layer Description: A detailed description of the layer and what it contains.
    - Tags: Keywords that help find the catalog when searching in the Portal.
    - Cost Allocation Tags: One or more free-form tags which are used to group billing records together.
5. Select a layer type. For information about the layer types, see Layers.
6. Configure the layer. For more information for configuring each layer type, see:
    - Stream Layer Settings
    - Versioned Layer Settings
    - Volatile Layer Settings
    - Index Layer Settings
7. Click **Save**.

    It may take a few minutes to create the layer. You will see a notification banner when the layer is created, and the layer will appear in the catalog.

Stream Layer Settings

You can configure stream layer settings when creating a layer in a catalog.

Note

Data in stream layers is encrypted and stored for the amount of time specified in the layer's retention setting, also known as Time-To-Live (TTL).

Maximum Throughput

You can specify the maximum throughput for data going into the layer and, separately, the maximum throughput for data going out of the layer. The HERE Open Location Platform starts throttling inbound messages when the inbound rate exceeds the inbound throughput. It starts throttling outbound messages when the total outbound rate to all consumers exceeds the outbound throughput.

The default value for inbound throughput is 4 MBps and the default outbound throughput is 8 MBps. You can specify up to 32 MBps for inbound and 64 MBps for outbound.

Catalogs in the OLP Marketplace have a maximum outbound throughput of 2 MBps.

We recommend that you set the outbound throughput to be at least the expected number of consumers (users and pipelines) times the inbound throughput. The output rate can be higher if some consumers "replay" recent data. The inbound throughput must not be more than the outbound throughput. If it is, the consumer cannot read all the data that the producer provides.

Retention

A stream layer can be configured with a retention time value, also known as Time-To-Live or TTL. The TTL value defines the minimum length of time that a message remains available for consumption.

Generally, TTL allows consumers to receive all published messages in various error case scenarios, such as temporary network connectivity failure, reboot of a client, and so on.

Messages are removed from the layer after the TTL time has elapsed, but not exactly at the retention time specified by the TTL setting. Messages may remain in the system for a period of time after the TTL time has elapsed. You are not charged for data storage beyond the TTL setting.

The TTL value is applied to all messages published to a layer.

Specifying a larger TTL value is especially useful when the data producer expects consumers to replay (re-consume) recent data. For example, if a layer stores vehicle sensor data the consumer wants to compare results of different algorithms to identify a particular road or vehicle condition, and the consumer needs real data for three recent hours to be consumed several times, then a TTL of four or more hours would be appropriate. Another example is consuming all the data that has been received within the last hour, every hour (provided that TTL is set for more than one hour).

The valid range of TTL values is from 10 minutes to 3 days. The default value is 1 hour.

Content Type

The content type specifies the media type to use to identify the kind of data in the layer.

Content Encoding

The content encoding setting determines whether to use compression to reduce the size of data stored in the layer. To enable compression, specify **gzip**.

If you are using the Data Client Library, the zipping and unzipping of data is handled automatically.

If you are using the Data API, you must zip data before writing it to the layer. When reading data, the data you receive is in gzip format, so you are responsible for unzipping it.

When gzip encoding is enabled, the partition metadata field `compressedDataSize` contains the size of the compressed data and `dataSize` contains the size of the uncompressed data.

Schema

Specifying a schema enables you to share data with others by defining for others how to consume the data. For more information, see Schemas.

Coverage

The geographic area that this layer covers. This setting controls which areas of the world are highlighted in the layer's coverage map in the Portal.

Versioned Layer Settings

This topic describes the configuration settings for a versioned layer. You configure these settings when you create a layer in a catalog.

Partitioning

The partitioning scheme determines how partitions in the layer are named. Use HERE Tile partitioning for map data and use generic partitioning for other kinds of data. For more information, see Partitions.

Content Type

The content type specifies the media type to use to identify the kind of data in the layer.

Content Encoding

The content encoding setting determines whether to use compression to reduce the size of data stored in the layer. To enable compression, specify **gzip**.

If you are using the Data Client Library, the zipping and unzipping of data is handled automatically.

If you are using the Data API, you must zip data before writing it to the layer. When reading data, the data you receive is in gzip format, so you are responsible for unzipping it.

When gzip encoding is enabled, the partition metadata field `compressedDataSize` contains the size of the compressed data and `dataSize` contains the size of the uncompressed data.

Schema

Specifying a schema enables you to share data with others by defining for others how to consume the data. For more information, see Schemas.

Checksum Algorithm

The digest property specifies the algorithm used by the data publisher to generate a checksum for each partition in the layer. By specifying a digest algorithm for the layer, you communicate to data consumers the algorithm to use to verify the integrity of the data they retrieve from the layer.

You can specify a digest only when creating a layer. Once the digest is set, you cannot change it. If you specify "undefined" as the digest, you cannot specify a digest after the layer is created.

When choosing a digest algorithm, consider the following:

- SHA-256 is recommended for applications where strong data security is required
- MD5 and SHA-1 is acceptable when the purpose of applying a checksum is to verify data integrity during transit.

Including a checksum is optional, but if you intend to provide checksums for partitions in this layer you should specify the algorithm you will use.

Note

The HERE Open Location Platform does not verify that the algorithm you specify here is the one used to generate the actual checksums, so it is up to the data publisher to ensure that the algorithm specified here is the one used in the publishing process.

For more information about common algorithms, see Secure Hash Algorithms.

Coverage

The geographic area that this layer covers. This setting controls which areas of the world are highlighted in the layer's coverage map in the Portal.

Volatile Layer Settings

You can configure volatile layer settings when you create a layer in a catalog.

Note

Data in volatile layers is not encrypted in this release of the HERE Open Location Platform.

Package Type

The package type controls the amount of data you can store in the layer.

PACKAGE TYPE	CAPACITY (GB)
experimental	0.416
small	2.085
medium	10.11
large	21.315

All package types have very high availability, except for the experimental package, which has moderate availability.

For the small, medium, and large package types, the read load does not impact the the performance of write operations. This means that for these package types, as the number of consumers increases, there is no impact on how fast data can be written into the layer.

For the experimental package type, the read load may impact write performance.

If you do not specify a package type, the experimental type is used by default.

Maximum Memory Policy

When a volatile layer is full, a decision needs to be made on what action should be taken. The following options are available:

- `FailOnWrite` : the write operation will fail and an error code will be returned to the client.
- `ReplaceLessRecentlyUsedPartition` : The volatile layer keeps track of when each partition was written and read. When no space is available in the layer, the partition that has not been accessed for the longest time will be automatically removed to create space for the new partition. Note that if removing one partition does not create enough space, several partitions may be removed.

If you do not specify a maximum-memory-policy, `FailOnWrite` is used by default.

Retention

A volatile layer can be configured with a retention time value, also known as Time-To-Live or TTL. The TTL value defines the length of time partitions in the layer will exist. After the retention time elapses for a partition, the partition is removed. Specifying a TTL value is especially useful when the validity period of partitions in the layer is

known in advance. For example, if a layer stores traffic incidents which are known to expire within 24 hours, then the TTL should be set to 24 hours.

Since data in volatile layers is not encrypted, you should also consider how long you want unencrypted data to remain in the layer when choosing a retention value. The longer the retention time, the longer unencrypted data remains in the layer.

If you are setting the retention time using the Portal, the unit is minutes. The valid range is 1 minute to 10080 minutes (seven days). The default is 60 minutes.

If you are setting the retention time using the Data API, the unit is milliseconds. The valid range is 1 millisecond to 6.048e+8 milliseconds (6.048×10^8 milliseconds).

Partitioning

The partitioning scheme determines how partitions in the layer are named. Use HERE Tile partitioning for map data and use generic partitioning for other kinds of data. For more information, see Partitions.

Content Type

The content type specifies the media type to use to identify the kind of data in the layer.

Content Encoding

The content encoding setting determines whether to use compression to reduce the size of data stored in the layer. To enable compression, specify **gzip**.

If you are using the Data Client Library, the zipping and unzipping of data is handled automatically.

If you are using the Data API, you must zip data before writing it to the layer. When reading data, the data you receive is in gzip format, so you are responsible for unzipping it.

When gzip encoding is enabled, the partition metadata field `compressedDataSize` contains the size of the compressed data and `dataSize` contains the size of the uncompressed data.

Schema

Specifying a schema enables you to share data with others by defining for others how to consume the data. For more information, see Schemas.

Checksum Algorithm

The digest property specifies the algorithm used by the data publisher to generate a checksum for each partition in the layer. By specifying a digest algorithm for the layer, you communicate to data consumers the algorithm to use to verify the integrity of the data they retrieve from the layer.

You can specify a digest only when creating a layer. Once the digest is set, you cannot change it. If you specify "undefined" as the digest, you cannot specify a digest after the layer is created.

When choosing a digest algorithm, consider the following:

- SHA-256 is recommended for applications where strong data security is required
- MD5 and SHA-1 is acceptable when the purpose of applying a checksum is to verify data integrity during transit.

Including a checksum is optional, but if you intend to provide checksums for partitions in this layer you should specify the algorithm you will use.

Note

The HERE Open Location Platform does not verify that the algorithm you specify here is the one used to generate the actual checksums, so it is up to the data publisher to ensure that the algorithm specified here is the one used in the publishing process.

For more information about common algorithms, see Secure Hash Algorithms.

Coverage

The geographic area that this layer covers. This setting controls which areas of the world are highlighted in the layer's coverage map in the Portal.

Index Layer Settings

An index layer has a user-defined structure and it can contain up to four attributes, one of which must be a time attribute. An index layer contains the metadata (values of the index attributes) plus some additional information such as the data handle for the data blob, the size of the data, timestamps, and checksums.

Note

Data in index layers is encrypted and stored for the amount of time specified in the layer's retention setting, also known as Time-To-Live (TTL).

Retention

An index layer can be configured with a retention time value, also known as Time-To-Live or TTL. The TTL value defines the number of days that records are kept in the layer and available for query. After the expiration of the TTL, the record is eligible for removal (actual deletion may take 24 hours after the expiration). This TTL value is applied to all records published to the layer.

Setting a TTL for your index layer enables you to automate your data management practices so that you can more easily maintain your data retention and cost. Selecting limitless retention requires you to manually manage these variables as your data will continue to accumulate as long as it is ingested and processed by your pipeline. Alternatively, a TTL setting of seven days will start deleting data records when they are seven days old. All newer data records are retained until they also reach seven days in age.

The minimum TTL is 7 days. This is also the default TTL setting.

Note

Once the layer is created, you cannot update the retention setting.

Index Attributes (Keys)

Index attributes define the keys by which you can query data in the index layer. For example, you can define an index layer to index sensor data from automobiles using the attributes time, tile ID, and event type. You can then develop a pipeline using the Data Archiving Library to aggregate data based on these attributes. After messages are indexed into the index layer, you can query the data based on the time, tile ID and event type.

The index layer requires time (either ingestion time or event time) as an attribute to facilitate the archival and querying of stream data where time is typically an important factor. While time is required, the remaining three optional attributes can be used to index and query by other aspects of the data such as location.

All index attributes, also known as keys, have two properties: `name` and `type`. The `name` attribute is used primarily in the query API to express the query predicate. The `type` attribute defines the data type stored in the attribute. The supported types are:

- Standard types:
  - `bool` - a Boolean value.
  - `int` - a signed integer, up to 64 bits.
  - `string` - a string of Unicode characters with a maximum length of 40.

- OLP types:
  - `heretile` - represents the tile id in the HERE tile map tiling scheme. The `heretile` type has an attribute zoomLevel which represents the size of the tile. It is not mutable.
  - `timewindow` - represents the finest time granularity at which the data will be indexed and later queried. The `timewindow` is a time slice, not just a point in time. For example, if you specify a time window of one hour, the value of the `timewindow` attribute for all records with an event time in a given 60-minute window will have the same index value for `timewindow`. You can specify the duration for a time window, which represents the time slice length and is not mutable. Both the `timewindow` value and `timewindow` duration are expressed in milliseconds, and the time value is milliseconds since Epoch. Note that the `timewindow` value is represented as the timestamp of the beginning of the window.

Note

Once the layer is created, you cannot update the indexing attributes.

Attribute Validation Rules

Index attributes must conform to the following rules:

1. The maximum attribute name length is 64 characters.
2. Attribute names must begin with a Unicode letter. Subsequent characters can be letters, underscores (_), and digits (0-9).
3. The `zoomLevel` attribute for the `heretile` type must be from 0 to 14.
4. The `duration` attribute of the `timewindow` type must be from 600000 (10 minutes) to 86400000 (24 hours).
5. The `timewindow` attribute is required.
6. An index can have no more than one `timewindow` attribute and one `heretile` attribute.
7. Each index attribute must have a unique name.
8. The index attribute name cannot be any of the following: "id", "size", "checksum", "metadata" or "timestamp". These are reserved for default attributes.

Content Type

The content type specifies the media type to use to identify the kind of data in the layer.

Content Encoding

The content encoding determines whether to use compression to reduce the size of data stored in the layer. To enable compression, specify **gzip**.

If you are using the Data Client Library, the zipping and unzipping of data is handled automatically.

If you are using the Data API, you must zip data before writing it to the layer. When reading data, the data you receive is in gzip format, so you are responsible for unzipping it.

When gzip encoding is enabled, the partition metadata field `size` contains the size of the compressed data. Otherwise, it will contain the size of the uncompressed data.

Schema

Specifying a schema enables you to share data with others by defining for others how to consume the data. For more information, see Schemas.

Coverage

The geographic area that this layer covers. This setting controls which areas of the world are highlighted in the layer's coverage map in the Portal.

Reconfigure a Layer

A layer can be reconfigured by anyone with *manage* permissions for the catalog that contains the layer. However, there are a few layer configurations that cannot be edited, such as:

- Layer type
- Volatile package size
- Streaming throughput

If you want to change any non-editable layer configuration, you need to delete the catalog, which deletes the layer and its data, then create a new catalog and layer.

To reconfigure a layer using the Portal:

1. Click **Data**.
2. Browse to the layer you want edit and select the layer to open it.
3. Click **More** then click **Reconfigure layer**.

For information about the configuration fields, see Creating a Layer.

Update a Schema

To update a schema, use the Schema Archetype tool in the HERE Open Location Platform SDK to create a new version of the schema and push it, and its dependencies and documentation, to the HERE Open Location Platform. For more information, see the SDK Developer Guide.

Note

You must have the HERE Workspace Plan to have access to the HERE Open Location Platform SDK.

Browse Schemas

Schemas define the organization of data in a layer, including the structure of the data and its content. You can browse the schemas available to you using the HERE Open Location Platform Portal. For example, you may want to browse schemas to look for one to use when creating a catalog.

To browse schemas using the Portal:

1. Click **Data**.
2. In the upper right corner, click **Browse schemas**.

   The list of schemas only contains those that you have permission to read.

3. Click on a schema to view its details, including dependency artifacts and documentation.

   To download the schema, click **More** then **Download schema**.

Create a Schema

If the schemas provided in the HERE Open Location Platform do not describe your data or support your use case adequately, you can create your own schema. To create a schema you must use the HERE Open Location Platform SDK. To access the SDK from the Portal, click the *Resources* tab then click *Download the SDK*.

Note

You must have the HERE Workspace Plan to have access to the HERE Open Location Platform SDK.

Use the SDK to create a schema and deploy it to the HERE Open Location Platform artifact repository. For documentation on creating and deploying a schema, go to the SDK Developer's Guide and browse to the topic *Create and Extend Schemas*.

The schema should now be visible in the list of schemas available in the HERE Open Location Platform Portal. The schema also appears in a dropdown list of schemas when configuring a data layer.

Schemas are only visible to the user who created it. To share a schema with other users, see Share a Schema.

Share a Schema

Sharing a schema is a necessary step in sharing a catalog. When you share a catalog, the catalog and its layers are shared, but not the layers' schemas. You should share the schemas used in the catalog so that users of the catalog have access to the definition of the layers' data structure. You must have manage permissions to the schema in order to share it.

To share a schema using the Portal:

1. Click **Data**.
2. In the upper right corner, click **Browse schemas**.
3. Select the schema you want to share.
4. Click the **Sharing** tab.

   ### Note

   If you do not see the **Sharing** tab it is because you do not have manage permissions for the schema.

5. Click **Share**.
6. Select or enter a user, app, or client ID then choose which permissions to grant.
   - **Read** - Grants the ability to view the schema data and metadata.
   - **Modify** - Grants the ability to read, edit, and delete the schema.
   - **Share** - Grants the ability to share the schema with others.
7. Click **Grant**.

Delete a Schema

To delete a schema you must either be the creator of the schema or the schema creator must grant you delete permission.

Caution

Use caution when deleting a schema that has been shared. When a schema is shared, others will likely create dependencies on the schema. For example, another user's business might rely on a processing pipeline which depends on the schema. Deleting the schema will cause the pipeline which depends on it to break and adversely impact the user's business.

To delete a schema using the Portal:

1. Click **Data**.
2. In the upper right corner, click **Browse schemas**.
3. Select the schema you want to delete.
4. Click **More** then click **Delete schema**.