



# External Data Provider

Revision	Author	Date	Reviewer	Approved By	Changelog	JIRA Story
rev 0.2	Zhang, Mingming	2021-11-10	<div><div><input type="checkbox"/></div><div><input checked="" type="checkbox"/></div><div><input type="checkbox"/></div><div><input type="checkbox"/></div></div>	<div><div><input type="checkbox"/></div><div><input type="checkbox"/></div><div><input type="checkbox"/></div><div><input type="checkbox"/></div><div><input type="checkbox"/></div><div><input type="checkbox"/></div><div><input type="checkbox"/></div><div><input type="checkbox"/></div></div>	<ul style="list-style-type: none"><li>Support multiple providers for different types of external data.</li><li>Each provider will return their own geojson the type of geojsongot from <b>styleKey</b>.</li><li>Add the Pause/Resume for shot down quickly.</li><li>Add GetExternalDataGridLevels(gridLevels) to Get the supported grid levels of external data, mapping of the zoom level of map engine, before do querying.</li></ul>	<div><div> <b>ANDROID-74</b> - Add support for external data provider</div><div>CLOSED</div></div>

rev 0.1	Zhang, Mingming	2020- 11-3			Design of external data provider	 <b>ANDROID-74</b> - Add support for external data provider <span>CLOSED</span>
---------	--------------------	---------------	--	--	----------------------------------	--

## Table of Contents

- [Introduction](#)
- [Requirements](#)
- [Limitations and Constraints](#)
  - [Limitations](#)
- [Design](#)
  - [JSON Format Use the GeoJson <https://geojson.org/>](#)
  - [External Data Cache](#)
- [User Interfaces](#)
  - [Public API](#)
  - [Interface of ExternalDataProvider](#)
  - [The Struct of GirdLevel](#)
- [Change Impact](#)
  - [Performance and Memory Impact](#)
- [Others \(FAQ, References, ...\)](#)

# Introduction

To improve usability, the Atlas engine should have the capability of loading POIs on the map automatically, other than depending on the user to search the POIs and adding them as Annotation on the map manually.

## Requirements

The idea is to define an interface of ExternalDataProvider in the Atlas Engine.

The logic of the POI query will be implemented in Java and then be called by Atlas Engine on the C++ side via JNI of ExternalDataProvider.

## Limitations and Constraints

### Limitations

1. Confirmed with [Adrian Tut](#) Now, the annotation is added per view, can't be shared with other views.
2. ~~At high zoom level( e.g., >5), don't send the query request of external data.~~  
Because the POIs(the same as road name) won't display on the map at high zoom level, besides the bbox will be too big to query too many POIs that don't use.

## Design

### JSON Format

Use the GeoJson <https://geojson.org/>

#### POIs example

```
// the json file result of annotation provider
{
  "type": "FeatureCollection",
  "features": [
    {
      "type": "Feature",
      "properties": {
        "styleKey": "poi_annotations.*", // the styling: icon-size, icon-color, icon-opacity, icon-rotation, icon-placement, icon-collisions, icon-layer-type
                                         // text-font, text-size, text-color, text-alignment, text-position-offset

        "icon_name": "xx.png",           // the "xx.png" should have saved in the local resouce. Note: must have one of "icon_name" and "graphic_id", if both, pick
the first one
        "graphicId": "",                // when get raw file of picture, the instance of ExtenalDataProvider need to use AddAnnotationGraphic to create graphicId
        "text": "",                     // if null, no text display on the annotation
        "annotation_layer": -1,          // -1:NoLayer    0:TilePoiLayer    ...
        "annotation_type": 0             // 0:Fixed      1:Billboard    ...
      }
    }
  ]
}
```

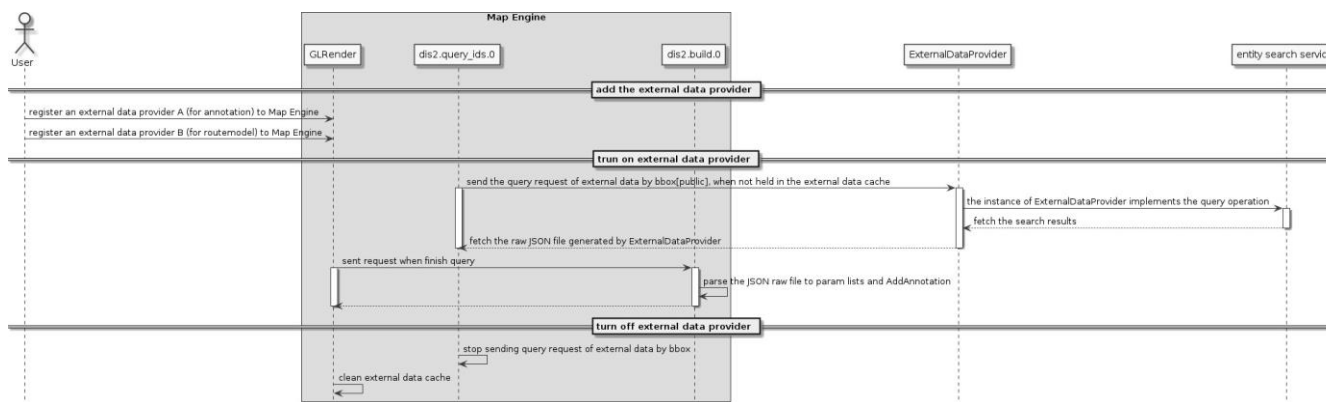
```
                                // comment:we don't support render order, it's done from the tss, we do use the priority to change the render order
however between annotations of the same layer type
```

```
    },
    "geometry": {
      "type": "MultiPoint",
      "coordinates": [
        [lat0,lon0],
        [lat1,lon1]
      ]
    }
  ]
}
```

```
// the json file result of routemodel provider
```

```
{
  "type": "FeatureCollection",
  "features": [
    {
      {
        "type": "Feature",
        "properties": {
          "styleKey": "route.*",          // the styling: simplify-factor, simplify-func, width, opacity, enable-route-eating.
        },
        "geometry": {
          "type": "MultiPoint",
          "coordinates": [
            [lat0,lon0],
            [lat1,lon1],
            [lat2,lon2],
          ]
        }
      }
    ]
  ]
}

....
```



**Note**The external data build request will always refresh to the current frame, meaning the build requests sent by the previous frames will set as stale and clear except for the ones be building.

## External Data Cache

The cache is an LRU contained the key<gridId, zoomlevel>, value is annotationId.

When doing query external data, the engine will verify if the region is already queried by checking the cache. If not in the cache, the engine will sent query request to entity search, otherwise won't send.

After getting the result of the query, the engine records <gridId, zoomLevel>into the cache, which means this region has been queried.

For the multiple providers, using a map to maintain each external data cache, the key is the type of provider, value is LRU <key<gridId, zoomlevel>, value is annotationId/ routemodel id ?>.

maybe no need to add a value in this LRU, just record the **type** of provider(come from the **styleKey**) and the queried <gridId, zoomlevel>.

## User Interfaces

### Public API

### Interface of ExternalDataProvider

#### ExternalDataProvider

```

namespace tn
{
  namespace atlas
  {
  
```

```

struct GridLevel
{
    uint32_t grid_level;           // the grid level fo the external data
    std::set<uint32_t> zoom_Levels; // this grid level mapping the zoom levels of map engine
};

/**
 * @brief External Data Provider
 */
class ExternalDataProvider
{
public:
    /**
     * @brief destructor of ExternalDataProvider
     */
    virtual ~ ExternalDataProvider() {}

    /**
     * @brief The Pause() will stop querying or GetExternalDataZoomLevels(), until Resume() is called.
     */
    virtual void Pause() = 0;

    /**
     * @brief The Resume() will resume querying external data or GetExternalDataZoomLevels().
     */
    virtual void Resume() = 0;

    /**
     * @brief Get the supported grid levels of external data, mapping of the zoom level of map engine.
     * @param zoomLevels [out] the supported grid levels of external data, and the each grid level contains the mapping zoom levels of map engine.
     * @note Only need to be called once,before do querying.
     */
    virtual void GetExternalDataGridLevels(std::vector<GridLevel>& gridLevels) = 0;

    /**
     * @brief Query the data of POIs data
     * @param x [in] the x component of girdId https://www.maptiler.com/google-maps-coordinates-tile-bounds-projection/
     * @param y [in] the y component of girdId
     * @param gridLevel [in] the z component of girdId
     * @return string contained json raw file
     */
    virtual std::string QueryExternalData(uint32_t x, uint32_t y, uint32_t gridLevel) = 0;
};

// tn
// atlas

```

## The Struct of GridLevel

### what is GridLevel

```
struct GridLevel
{
    uint32_t grid_level;           // the grid level fo the external data
    std::set<uint32_t> zoom_Levels; // this grid level mapping the zoom levels of map engine
};
```

We assume that the external data is layered by uniform tile, so grid IDs are distributed by level similar to <https://www.maptiler.com/google-maps-coordinates-tile-bounds-projection/>, here may divide into 18 layers same as the internal engine zoom levels.

**grid\_level** : a level determines the grid size used by the uniform tile solution.

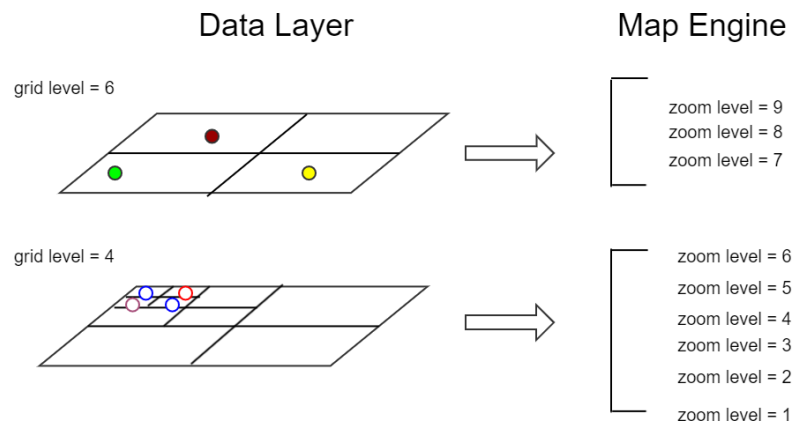
**zoom\_Levels** : a set of zoom level of the map engine.

For example,

Calling `GetExternalDataGridLevels(std::vector<GridLevel>& gridLevels)`, before do querying. One provider only call once.  
and get two elements:

{ grid\_level = 4, zoom\_Levels = { 1,2,3,4,5,6} }

{ grid\_level = 6, zoom\_Levels = { 7,8,9 } }



If the current zoom level of the map engine is 5, then use `grid_level = 4` to query results.

### why use GridLevel

Different results based on the grid level, make sure we can reuse the results between some zoom level ranges(the same grid size).  
When in high zoom level, use the lower grid level, and the numbers of the grid will be smaller, which means do fewer queries.

As [Li, Mingzhou](#) commented "*if we query the data by a fixed grid size on all zoom levels, the data can be reused between zoom levels, but there will be too many queries on high zoom levels.*

*IMO, I would suggest **we use a fixed grid size and limit the query on low zoom levels.**"*

This method (**according to Grid Level/Size, do querying**) is the upgraded version of " use a fixed grid size and limit the query on low zoom levels".

For better extensibility, we reserve an interface to support data layering, if external data do support the layer, we can do querying according to Grid size.  
**if not, `gridLevels.size() = 0`, we just use a fixed grid size(fixed grid level) on low zoom levels.**  
and from we know, the NEO data don't layer.

Discuss:

Maybe we don't need to expose the **zoom\_Levels** to fill by the client, we may specify some mapping mechanisms internally.

Like, If we can ensure the hierarchical level of the data is the same as the map engine, means data is also divided 18 layers, we can just use grid\_level to correspond to the zoom level of engine.

grid\_level = 4.

grid\_level = 6.

zoom level in [0, 4], query use grid\_level = 4,

zoom level in (5, 6], query use grid\_level = 6,

zoom level in (6, 6+min extended num], query use grid\_level = 6,

Interface Signature	Sample Code
---------------------	-------------



```

/**
 * @brief Register an external data provider.
 * @param [in] instance of the external data provider to be
registered.
 * @return true, means register the provider successfully,
otherwise, means failed
 */

virtual bool RegisterExternalDataProvider(tn::
weak_ptr<ExternalDataProvider> const& externalDataProvider) = 0;

```

```

/**
 * @brief Unregister an external data provider
 * @param [in] instance of external data provider to be unregistered.
 * @return true, means unregistered the provider successfully,
otherwise, means failed.
 */

virtual bool UnregisterExternalDataProvider(tn::
weak_ptr<ExternalDataProvider> const& externalDataProvider) = 0;

```

```

...

m_engine = ITnMapEngine::CreateMapEngine(...)

...
// register the providerA to Atlas engine
m_engine->RegisterExternalDataProvider(providerA); //NOTE: providerA is instance of
ExternalDataProvider implemented by the user of map engine.

// register the providerB to Atlas engine
m_engine->RegisterExternalDataProvider(providerB);
...

//query example:
// get the gridMappingZoomLevels
std::vector<GridLevel> gridLevels;
providerA -> GetExternalDataGridLevels(gridLevels);

//the supported grid levels of external data, and the each grid level contains the
mapping zoom levels of map engine.
for (auto girdLevel : gridLevels)
{
    auto it = std::lower_bound(girdLevel.zoom_levels.begin(), girdLevel.zoom_levels.end(),
currentZoomLevel)
    if(it != girdlevel.zoom_levels.end() && *it == currentZoomLevel )
    {
        providerA -> QueryExternalData(girdId, gridLevel);
    }
}

...

// unregister the providerA from Atlas engine
m_engine->UnregisterExternalDataProvider(providerA);

// unregister the providerB from Atlas engine
m_engine->UnregisterExternalDataProvider(providerB );
...

```

```
/*! Set a boolean engine parameter */  
virtual bool SetBool(ViewId viewId, eParameterBool param, bool  
value) = 0;
```

```
...  
// Turn ON, allow all providers to query ExternalDatas  
m_engine->SetBool(m_viewId, ITnMapEngine::eParameterBool_ExternalData, true);  
...  
// Turn OFF, forbid all providers querying the ExternalData and clear all external data  
cache.  
m_engine->SetBool(m_viewId, ITnMapEngine::eParameterBool_ExternalData, false);
```

## Change Impact

### Performance and Memory Impact

performance Impact: The operations of Querying and adding annotations are transformed from HMI thread to build thread of engine layer, so build thread maybe be slower under the same tasks.

memory impact: Adding a cache, which is a hashmap/LRU contained key is <gridId, zoomLevel>, value is annotationId. and its memory size should be configurable.

## Others (FAQ, References, ...)

The generation strategy of gridId is not determined.

- using quadtree
- using uniform tile

2020-11-10

we use the uniform tile, a fixed grid size and limit the query based on the zoom level got from GetExternalDataGridLevels().