Munich RE

# Location Risk Intelligence (LRI) API Specification

Version 2.3.3

Munich RE

# Table of Contents

Munich RE

# Getting Started

## Introduction

The Munich Re Location Risk Intelligence (LRI) platform is a powerful tool for comprehensively assessing physical risks caused by natural hazards and climate change. Integrating the LRI API into your applications and business processes gives you access to a wealth of data and functionalities that can significantly enhance your risk assessment capabilities.

The primary use cases of the API are:

- **Scoring (enrichment) of locations**
  When you submit location data, such as geospatial coordinates or addresses, the API provides enrichment results. These results include scores like natural hazards and climate change data, giving you a comprehensive understanding of the risk profile of the given location.
  See Location Scoring
  (Resource location-info)

- **Scoring (enrichment) of areas and lines (page 30)**
  For scoring areas and lines, you need to send the geospatial geometry of the object. The API then returns a percentage distribution for scores like natural hazards and climate change data. This process allows you to assess the risk distribution across the given area or line.
  See Area & Line Scoring
  (Resource area-line-info)

- **Map services (page 47)**
  You can use our WMS or WMTS services to visualize the LRI map layers in your own applications.
  See Map Services
  (Resource maps)

Additionally, the API provides comprehensive information like value ranges, descriptions, and colors for all the available services (see API Configuration Data).

All API services, such as natural hazards or climate change data, are versioned (see API Data (Content) Versioning).

You can find information about your API consumption in the chapter Usage Statistics.

# Understanding Location Risk Intelligence API resources

The API uses REST principles and can be used with any programming language or framework that supports REST, including test tools such as Curl, Fiddler, or Postman. The following conventions are used:

- Each resource is a named URI that is used with an HTTP method (GET or POST).
- Request and response bodies typically use the JSON format; for details, see Content Types.
- This API has a major version that is a mandatory part of the URL (v2). For details about major and minor versions of the API model, see API Model Versioning.
- The communication is only possible after authentication and is secured via TLS with HTTPS.

The REST resources are accessed through the following base URL:

```
https://api.munichre.com/nathan/v2/
```

For example, an HTTP GET on `https://api.munichre.com/nathan/v2/configuration` retrieves detailed information about all the available API services as a JSON object.

# Onboarding

To access the LRI API, you need a valid contract for the Enterprise edition of the Location Risk Intelligence platform. If you don't have this, please contact your sales representative with a corresponding request.

Our representatives will conduct a kick-off meeting, show how the API works, provide samples, discuss technical implementation, and clarify open questions.

The API onboarding is then performed in the following steps:

- We invite your technical contact to the Munich Re developer portal.
- In the developer portal, you create a subscription for the LRI API and retrieve the subscription key. This subscription key is one of two authentication factors (see also Authentication).
- We provide you with the credentials for your API accounts (see Authentication), data documentation, and a collection of sample requests tailored to your use cases.

# Generating Your API Client Code

You can download the OpenAPI specification (https://swagger.io/specification/) for the LRI API in JSON format from the developer portal here:

https://developers-prod.munichre.com/

You can use this file to generate code for an API client library automatically. Multiple tools support code generation based on OpenAPI specifications for most programming languages (for example, https://github.com/swagger-api/swagger-codegen).

# Authentication

All requests to the LRI API must be authenticated, and all endpoints use the same authentication mechanism. Each request must contain two authentication factors:

- The subscription key from the developer portal
- A JWT bearer token that you retrieve from an OAuth2-endpoint with the credentials for your API account (client-id and client-secret)

## Subscription Key

The first authentication factor is the subscription key. The subscription keys can be retrieved from the developer portal in the profile section (https://developers-prod.munichre.com/profile). It does not matter if you use the primary or the secondary key. You can also regenerate the keys at any time.

Add the primary or the secondary key to the HTTP header `Ocp-Apim-Subscription-Key` of every request to the API.

## Bearer Token

The second authentication factor is a JSON Web Token (JWT) created with an OAuth2 flow using a client-id and a client-secret. This JWT token must be added to every API request's authentication header (bearer schema).

The token can be retrieved with a POST request to this endpoint:

```
POST https://login.microsoftonline.com/munichre.onmicrosoft.com/oauth2/token
-H 'Cache-Control: no-cache'
-F grant_type=client_credentials
-F resource=https://munich.munichre.com/nathan
-F client_id={your_client_id}
-F client_secret={your_client_secret}
```

The values for client_id and client_secret will be provided during the onboarding process.

On success, you will receive a response with a 200 status and the token data in the response body:

```
{
    "token_type": "Bearer",
    "expires_in": "3599",
    "ext_expires_in": "3599",
    "expires_on": "1717439758",
    "not_before": "1717435858",
    "resource": "https://munich.munichre.com/nathan",
    "access_token": "eyJ0eXAiOiJK… "
}
```

Please add the value for the `access_token` property to the HTTP header `Authorization` in the format `bearer {access_token value from the OAuth2 response}`.

The tokens are always valid for one hour (3600 seconds).

### Using the API for Non-Production Environments (Development, Testing, Staging)

The LRI API has no special testing or sandbox environments. Instead, we will provide you with two different API accounts (identified by client-id and client-secret):

- One account for your production system(s)
  Requests made with this account will count towards your contractual quota.

- One account for all your non-production environments
  (Development, testing, staging, etc.)
  We will track requests with your test account, but they won't be invoiced.

By default, test accounts return the same results as the production accounts, but the maximum number of requests is limited. Details will be included in your contract. The data from requests with the test account are also only to be used in non-production systems.

Higher limits for testing purposes are also possible, but the results will be falsified. Contact us if you need higher limits for testing (for example, load tests).

Please note:

We distinguish between testing and production usage only by the client-id of your API accounts. You can use the same subscription key in all environments.

## Making Requests

### HTTP Verbs

As of today, the Location Risk Intelligence API only allows analytical requests. Creating, updating, or modifying resources based on the CRUD principle is not possible or necessary. This is why most requests follow the READ principle (e.g., all GET methods).

The input data for most requests can be quite large. For example, a request to score locations can include up to 1000 coordinates or addresses. This can’t be handled in the query parameters of a GET request and requires a request body. This is why the POST method is used for several analytical requests to retrieve data instead of GET.

The detailed descriptions for the API endpoints provide more details about the request and response model, and you can also use the OpenAPI specification (Swagger file).

### Content Types

POST requests to the API always use the JSON format in the request body (application/json).

Most API responses also use JSON (application/json). A typical example would be the results for scoring locations.

The map services return images (WMS tiles as image/png).

The endpoints to retrieve the data documentation or the map service documentation return PDF documents (application/pdf).

## Responses

Any request will result in a response with the appropriate HTTP status code and a response body, which depends on the type of the request (see Content Types). In case of an error, the HTTP status code will indicate the issue, and an error message will be returned as a JSON object (see Error Handling).

**HTTP status codes**

The status code is used to reflect the status of the request.

- Any successful request will return a status code in the 2xx range.
- Invalid requests will return a status code in the 4xx range.
- Server-side errors will return a status code in the 5xx range.

## Error Handling

The API uses the official "Problem Details" standard for HTTP APIs: `https://tools.ietf.org/html/rfc7807`

In case of an error, you will receive an HTTP response code and a `ProblemDetails` object in the response body.

The `ProblemDetails` object contains these properties:

| | |
|---|---|
| **Title** | String<br>A short message indicating what went wrong<br>Examples:<br>`"Request Validation Error"`<br>`"Unauthorized"`<br>`"Not Found"` |
| **Status** | Int<br>The HTTP status code like 400, 404, 503 |
| **Detail** | String (nullable)<br>A detailed description of the actual issue. This message can help to point you in the right direction to fix the problem (especially for request validation errors). |

Here is an example of the error response to an invalid request to the location-info endpoint

```
{
   "title": "Request Validation Error",
   "status": 400,
   "detail": "Request does not contain items"
}
```

# Health Check

The API provides a `health` endpoint that you can use to check the availability of the service:

```
GET - https://api.munichre.com/nathan/v2/health
```

The HTTP status code of the response determines the availability of the API:

- 200 = API is available
- Any other status code = The API is not available

The response body of the health check is subject to change and, therefore, not documented in this API specification.

Please note that the health check requires the same authentication as any other endpoint (see Authentication).

Munich RE

# Location Scoring (Resource location-info)

## Overview

The resource `location-info` contains the endpoints to retrieve the scores (enrichment), such as natural hazards, risk scores, and climate change data for locations identified by geospatial coordinates or addresses.

This chapter covers the following use cases:

- Requesting scores for locations
- Batch processing of larger quantities
- Getting data documentation for the scores

## Requesting Scores (Synchronous Execution)

### Basic Request & Response Structure

*Request*

To retrieve scores, you send a POST request to the `location-info` endpoint. Such a request contains two main parts:

- An array of **items** = locations (identified by geospatial coordinates or addresses)
- An array of **services** = the required scores such as natural hazards, risk scores, or climate change data

A sample request looks like this:

```
POST - https://api.munichre.com/nathan/v2/location-info
```

```
{
   "items":[
      {
         "id":"1",
         "coordinate":{
            "longitude":10.99438,
            "latitude":45.43898,
            "countryCode":"ITA"
         }
      }
   ],
   "services":[
      {
         "category":"Hazards",
         "service":"NathanEarthquake",
```

```
         "version":"current",
         "requestedServiceId":"Hazards__NathanEarthquake"
      },
      {
         "category":"Hazards",
         "service":"NathanRiverFlood",
         "version":"current",
         "requestedServiceId":"Hazards__NathanRiverFlood"
      }
   ]
}
```

Requests can contain up to 1000 locations and an unlimited number of services.

Each location in the items array needs a unique identifier to connect the requested location with the results in the API response. This identifier can be any string, for example, a primary key from your backend system.

The API scores are organized into categories and services. To request a specific score, you need these four values:

- category
  (for example, `"General"`, `"Hazards"`, `"ClimateChange"`)
- service
  (for example, `"PopulationDensity"`, `"NathanEarthquake"`, `"HeatStressIndexSSP126Y2050"`)
- version
  All data in the API is versioned. You can request a specific version or simply use the constant `"Current"` to always get the latest version (recommended).
- requestedServiceId
  This must be a unique string that connects the requested service (identified by category + service + version) with the corresponding result in the API response (see the example response below).

To learn more about the available categories and services, see Discovering the Available Services and Data Documentation.

For detailed information on location-info requests, see Request Model.

### *Response*

The API response contains an array of `items` that match the input items by the `id` (one element per requested location).

Each item contains an array of `results` that match the requested services by the `requestedServiceId` (one element per requested service).

The API response to the above sample request looks like this:

```
{
   "items":[
      {
         "id":"1",
         "results":[
            {
               "requestedServiceId":"Hazards__NathanEarthquake",
```

```
            "values":{
               "HazardZone":2
            }
         },
         {
            "requestedServiceId":"Hazards__NathanRiverFlood",
            "values":{
               "HazardZone":50
            }
         }
      ],
      "matchedAddress":{
         "id":0,
         "countryCode":"ITA"
      }
   }
],
"requestId":1395626986,
"processingTimeInMilliseconds":0
}
```


The actual scores are contained in the `values` property for each element in the `results` array. This is a dictionary that contains the scores in this format:

- Key = field name
  (for example, `"HazardZone"`, `"RiskScore"` , `"AnnualMaxTemp"`
- Value = enrichment result (score)
  (for example, `0`, `"Medium"`, `32.3`

The field names and types vary between the different API services (see Discovering the Available Services and Data Documentation).

Additionally, the response contains information about the found input location:

- When you send geospatial coordinates, you will get a `matchedAddress` property that contains the found `countryCode` for the given longitude and latitude. You can also specify the `countryCode` in the input `coordinate` object. In this case, the API will simply return the input value.
- When you send addresses, you will get a `matchedAddress` property that contains the matching address data we found during our geocoding process. You will also get a `coordinate` object that contains a `geoqualty` property that indicates the accuracy of the geocoding process.

We recommend always using coordinates instead of addresses to identify locations (see Geospatial Coordinates vs Addresses).

For detailed information on location-info responses, see Response Model.

## Discovering the Available Services and Data Documentation

The available enrichment services (scores) depend on your licensed data edition. You have two possibilities to get detailed information about these services:

### *Structured data*

A GET request to the `configuration` endpoint returns a comprehensive JSON object containing detailed data about the available API content.

Please see the section API Configuration Data for in-depth descriptions of this data.

### *PDF data documentation*

Send a POST request to the `location-info/documentation` endpoint to obtain the human-readable data documentation in PDF format.

Here is an example request for the data documentation:

```
POST - https://api.munichre.com/nathan/v2/location-info/documentation
```

```
{
   "ValueMapping":"Traffic Light"
}
```

This request body is a DocumentationReportOptions object:

| | |
|---|---|
| **ValueMapping** | String (optional)<br>The name of a ValueMapping category like `"Traffic Light"`. If this option is included, the documentation will include information on how the regular API scores (for example, hazard zones) are mapped to the given ValueMapping. |

This data documentation PDF is always generated on the fly using the latest data based on your licensed data edition. The data you get from the `configuration` endpoint is used to build the document.

## Geospatial Coordinates vs Addresses

When you request enrichment results from the API, you have two options to pass the input data for the locations:

- Geospatial coordinates (longitude & latitude)
- Addresses (country, city, zip code, street, etc.)

We **highly recommend using coordinates** for the following reasons:

- Precision: Geo-coordinates provide a much more precise location than an address that may sometimes be ambiguous or inaccurate.
- Consistency: Geo-coordinates don't change, unlike addresses (renumbering, renaming, or other changes).
- Universal: Geo-coordinates can be used anywhere in the world, whereas addresses vary greatly in format and information from one country to another.
- Performance: Sending addresses has a significant performance impact as geocoding is a slow process.
- Further options: You can also use geo-coordinates in your own systems for further analytics or visualizations on digital maps.

Please note:
When you send addresses to the API, we run our own geocoding process. Due to licensing restrictions, we cannot return the resulting coordinates to you. Therefore, it's better to do the geocoding on your side before calling our API with the resulting coordinates.

## Including Value Mappings

Value mappings are different interpretations of the API scoring results. A typical example is mapping hazard zones to traffic light scores.

You can include value mappings for your requested scores by setting the `ValueMappings` property in the request. This property is an array of strings with the names of the value mappings:

```
{
   "valueMappings": ["Traffic Light"],
   "items":[
      {
         "id":"1",
         "coordinate":{
            "longitude":10.99438,
            "latitude":45.43898,
            "countryCode":"ITA"
         }
      }
   ],
   "services":[
      {
         "category":"Hazards",
         "service":"NathanEarthquake",
         "version":"current",
         "requestedServiceId":"Hazards__NathanEarthquake"
      },
      {
         "category":"Hazards",
         "service":"NathanRiverFlood",
         "version":"current",
         "requestedServiceId":"Hazards__NathanRiverFlood"
      }
   ]
}
```

The response will then contain a new property, `MappedResults`, for each requested location. This is a dictionary with the following structure:

- Key = name of the value mapping
- Value = Array of results similar to the normal results for the zone values.

For the request above, the response then looks like this:

```
{
    "items": [
        {
            "id": "1",
            "results": [
                {
                    "requestedServiceId": "Hazards__NathanEarthquake",
                    "values": {
```

```
                        "HazardZone": 2
                    }
                },
                {
                    "requestedServiceId": "Hazards__NathanRiverFlood",
                    "values": {
                        "HazardZone": 50
                    }
                }
            ],
            "mappedResults": {
                "Traffic Light": [
                    {
                        "requestedServiceId": "Hazards__NathanEarthquake",
                        "values": {
                            "HazardZone": 3
                        }
                    },
                    {
                        "requestedServiceId": "Hazards__NathanRiverFlood",
                        "values": {
                            "HazardZone": 5
                        }
                    }
                ]
            },
            "matchedAddress": {
                "id": 0,
                "countryCode": "ITA"
            }
        }
    ],
    "requestId": 1422423934,
    "processingTimeInMilliseconds": 22
}
```


Please note:

Not all zone results support all value mappings. For example, most services in the “General”-category, like “NathanElevation” or “PopulationDensity”, have no Traffic Light mapping. You will not get mapped results for these services.

Requesting value mappings has little impact on processing times but significantly increases the response size. If you have large requests with up to 1000 locations, it is reasonable to use smaller request sizes to avoid HTTP response size restrictions.

If you don’t need the default zone results at all, you can use the property `ExcludeZoneResults` to remove them from the response:

```
{
   "valueMappings": ["Traffic Light"],
   "excludeZoneResults": true,
   "items":[ … ],
   "services":[ … ]
}
```

In this case, the response items will only contain the mapped results. You can also only request services that support all selected value mappings.

The available value mappings and their definitions are also available through the `configuration` endpoint of the API. You can find detailed information in the chapter API Configuration Data (Resource configuration).

## Including a PDF Report

You can include an asset risk report in PDF format in the API responses for your location-info requests. This report will visualize the scoring results for the requested services similar to the PDF exports from the Location Risk Intelligence web application.

To request the report, add a `"ReportParameters"` object to the location-info request like this:

```
{
   "ReportParameters":{
      "GenerateReport":true,
      "AssetLabel":"Sample Asset Name",
      "AssetInfos":{
         "TSI":"1.234.567 EUR",
         "Contract date":"22 March 2023",
         "Building type":"Residential"
      }
   },
   "items":[
      …
   ],
   "services":[
      …
   ],
}
```

The `"AssetLabel"` property is a mandatory string for the title on the cover page.

The `"AssetInfos"` property is an optional dictionary with name/value pairs. If present, these will be added to the cover page.

For details, please refer to the Request Model.

Please note:

- Asset reports can only be added for requests with exactly one location.
- The generated reports include only the requested services and are not necessarily the same as the LRI web application.
- The reports from the LRI web application contain a screenshot of the current map window. This is not available in the reports from the API.
- Requesting a report significantly increases the processing time (by around 1 second).

The API response then contains a new property `"ReportData"` with the binary data for the PDF encoded in base64:

```
{
   …
    "reportData": "JVBERi0xLjQKJdPr6eEKMSAwI…"
   …
}
```

The base64 string has to be converted into binary data (byte array) and then serialized into a file to get the resulting PDF report.

## Error Handling

Enrichment requests to the location-info endpoint can lead to three basic kinds of errors:

- Error on the request level
- Error on the location level
- Error on the service (score) level

Here are some examples of these cases:

### *Error on the request level*

In case of structural errors in the request, you will receive a 400 HTTP response status (bad request) indicating the problem. For example, a typo in a requested service name prevents the execution and leads to an error message like this:

```
{
   "title":"Request Validation Error",
   "status":400,
   "detail":"Requested category Hazards does not contain the requested service
             NathanEarthquark"
}
```

Please refer to the `"title"` and `"detail"` properties to determine the cause of the problem.

### *Error on the location level*

A request can contain between 1 and 1000 locations. It's possible that some locations of one request contain errors and others work. Typical problems are invalid geo-coordinates or address data.

In these cases, the HTTP response code will still indicate success (200), and you can check the individual items in the response.

Here is an example response to a request containing two locations. One location is valid ("1"), and the other one ("2") has invalid coordinates:

```
{
   "items":[
      {
         "id":"1",
         "results":[
            …
         ]
      },
      {
         "id":"2",
         "errorCode":10,
         "error":"Latitude: Latitude is out of range (must be between -90 and 90)"
      }
   ],
   …
}
```

If there is an error on the location level, you will get an `errorCode` and an `error` message indicating the source of the problem instead of the results array with the scores.

### *Error on the service (score) level*

It's possible that sometimes individual scores cannot be retrieved for certain locations. A typical example would be an insufficient geocoding accuracy for some high-resolution scores.

Here is an example response for an input address that could only be geocoded on a city level (geoquality = 50, possibly due to invalid street data):


```
{
   "items":[
      {
         "id":"1",
         "results":[
          …
          {
               "requestedServiceId":"Hazards__NathanLightning__current",
               "values":{
                  "HazardZone":2
               }
            },
            {
               "requestedServiceId":"Hazards__NathanRiverFlood__current",
               "errorCode":21,
               "error":"Geoquality is too low (needs to be at least 90 - Street)"
            },
          …
```


In this case, you will receive scores for most services, but high-resolution hazards like riverflood require higher geocoding accuracy. If there is an error on the result level, you will get an `errorCode` and an `error` message indicating the source of the problem instead of the values dictionary with the scores.

## Limits

### *Limits for location-info requests*

The following limits apply for location-info enrichment requests:

- Up to 1000 locations per request
- No limit for the number of requested services (scores)
- No limit for the request and response sizes. Please bear in mind that responses to requests with a large number of locations and services can get quite large, and you might run into restrictions from your network settings. Split the requests into smaller numbers of locations if you encounter such limitations. Always include all required services in a request.

### *Connection limits and rate limits*

The API has no connection or rate limits. The maximum number of location-info requests that can be executed in parallel before experiencing reduced performance depends on the type and complexity of the individual requests.

For example, when requesting the full natural hazard and climate change scores for single locations, around 60 requests can run parallel before performance decreases.

Please note that this number can vary depending on the complexity of the actual request. We constantly monitor API performance and optimize our systems to scale with increasing demand.

Please note that executing one request with 100 locations is much faster than running 100 requests with one location each.

Moreover, the API is equipped with specialized batch endpoints. These endpoints are designed to handle larger quantities of data, ensuring optimal throughput (see Batch Processing (Asynchronous Execution)).

## Request Model

*LocationInfoRequest*

| | |
|---|---|
| **Items** | Array of **LocationInfoRequestItem** objects (required)<br>This is the list of the locations for the enrichment. |
| **Services** | Array of **LocationInfoRequestService** objects (required)<br>This is the list of the required services (scores) for the enrichment. |
| **ValueMappings** | String array (optional)<br>You can include value mappings like Traffic Light for your requested services in the response. This property is an array of strings with the names of the value mappings.<br>The response will then contain a new property, MappedResults, for each requested item. This is a dictionary with the following structure:<br>Key = name of the value mapping<br>Value = Array of results similar to the normal results for the zone values |
| **ExcludeZoneResults** | Boolean (optional)<br>If you requested one or more ValueMappings and don't need the normal zone results, you can set this property to true to remove them from the response. |
| **ReportParameters** | A **SingleAssetReportParameters** object (optional)<br>Add this object to include a PDF single asset report in the API response. |
| **UserId** | String (optional)<br>You can pass an identifier, which we will add to your API usage statistics for further analytics. This could be useful if you have multiple systems that use our API on your side.<br>Another typical use case is for API resellers to pass a key to identify the client who initiated the request.<br>Please note:<br>This field is not intended to pass sensitive information about actual end users, such as names or mail addresses. |
| **TransactionId** | String (optional)<br>You can pass an identifier for the transaction that initiated the API request on your side. This could be a primary key in your database or a log entry ID. We store this TransactionId in our logs, and it can be used to cross-reference our usage statistics with your data. |

*LocationInfoRequestItem*

| | |
|---|---|
| **Id** | String (required)<br><br>Each location in the request requires a unique identifier to connect the API response results to the requested location. There are no guidelines for this Id. You can use any value that helps you connect the API response data to the asset in your system (for example, a primary key from your backend). |
| **Coordinate** | A **Coordinate** object (optional)<br>This object contains the geo-coordinates of the location. We recommend using this object instead of an address.<br><br>A LocationInfoRequestItem must have either a Coordinate or an Address – not both. |
| **Address** | An **Address** object (optional)<br>This object contains address data such as country, city, zip code, and street. Please only use addresses if you cannot get the geo-coordinates (geocoding) yourself. |

*LocationInfoRequestService*

| | |
|---|---|
| **Category** | String (required)<br><br>The name of the enrichment category<br>(for example, "General", "Hazards", "ClimateChange") |
| **Service** | String (required)<br><br>The name of the enrichment service<br>(for example, "PopulationDensity", "NathanEarthquake", "HeatStressIndexSSP126Y2050") |
| **Version** | String (required)<br><br>The name of the enrichment service version<br><br>All data in the API is versioned. You can request a specific version identified in the format YYYYMMDD or use the constant "Current" to always get the latest version (recommended). |
| **RequestedServiceId** | String (required)<br><br>This must be a unique string that connects the requested service (identified by category + service + version) with the corresponding result in the API response.<br><br>There are no guidelines on how to build this Id. For example, you can use an identifier that corresponds to a primary key in your backend. Or you can simply combine the strings for Category and Service like “Hazards__ NathanEarthquake”.<br><br>This identifier is mainly required to reduce the response size by avoiding repeating the three values for category, service, and version for every location. |

*SingleAssetReportParameters*

| | |
|---|---|
| **GenerateReport** | Bool (required)<br>Indicates if a report should be added to the API response (true) or not (false). |
| **AssetLabel** | String (required)<br>This value will be rendered as the report title. |
| **AssetInfos** | Dictionary with Key = String and Value = String (optional)<br>If this field is set, the values will be added to the cover page of the report in this format:<br>{Key}: {Value}<br>For example:<br>Building type: Residential |
| **ValueMapping** | String (optional)<br>If you requested one or more ValueMappings, you can use this property to define which one should be used for the report. By default, the first value mapping in the ValueMappings property of the request will be used.<br>You can also use the constant “Zones” to get the report with the default zone values. |
| **IncludeDataSource** | Bool (optional, default = false)<br>If set to true, detailed data source information for all selected services will be added at the end of the PDF report. |
| **IncludeDetailedDescription** | Bool (optional, default = false)<br>If set to true, detailed descriptions for all selected services will be added at the end of the PDF report. |

## Response Model

*LocationInfoResponse*

| | |
|---|---|
| **ResultsAreFalsified** | Bool (nullable)<br>This property will be set to true if you use an API test account that delivers falsified results.<br>If the results are not falsified, this property will be null. |
| **Items** | An array of **LocationInfoResponseItem** objects<br>This array contains one object per input location with the enrichment results. |

| | |
|---|---|
| **ReportData** | String (nullable)<br>If a report has been requested, this property will contain the binary data for the PDF encoded in base64.<br>The base64 string has to be converted into binary data (byte array) and then serialized into a file to get the resulting PDF report. |
| **RequestId** | Int64<br>This is our internal identifier for the request. You can store this on your side to cross-reference your logs with ours. |
| **ProcessingTimeInMilliseconds** | Int32<br>This is the time the API spent processing your request (without network overhead). |

*LocationInfoResponseItem*

| | |
|---|---|
| **Id** | String<br>The value of this property matches the corresponding Id for the location in the items array of the LocationInfoRequest. |
| **Results** | An array of **LocationInfoResult** objects (nullable)<br>This array contains one object per requested service with the enrichment results.<br>If this array is null, something went wrong with the input location, and the error properties will be set. |
| **MappedResults** | A Dictionary with<br>Key = String (name of the requested ValueMapping) and<br>Value = Array of **LocationInfoResult** objects<br>(nullable)<br>If you requested one or more ValueMappings, this property will contain a dictionary with the mapped results. |
| **MatchedAddress** | An **Address** object (nullable)<br>When the corresponding input location has geospatial coordinates, this property will contain the found CountryCode for the given longitude and latitude.<br>When the corresponding input location has an address, this property will contain the matching address data we found during our geocoding process. |
| **Coordinate** | A **Coordinate** object (nullable)<br>When the corresponding input location has an address, this property will contain the geoqualty property, which indicates the accuracy of the geocoding process. |

**ErrorCode** Int32 (nullable)

This is an internal error code that you can use for logging purposes.

**Error** String (nullable)

If the input location contains an error (for example, invalid geo-coordinates), this property will contain a message that you can use to identify the source of the problem.

*LocationInfoResult*

**RequestedServiceId** String

The value of this property matches the corresponding Id for the requested service in the in the services array of the LocationInfoRequest.

**Values** A dictionary with key = String and value = object (nullable)

The actual scores are contained in the values property. This is a dictionary that contains the scores in this format:

Key = field name

(for example, "HazardZone", "RiskScore" , "AnnualMaxTemp"

Value = enrichment result (score)

(for example, 0, "Medium", 32.3)

The field names and types vary between the different API services. You can check the data documentation or use the API configuration endpoint to discover the available services and their fields.

If this property is null, then something went wrong while retrieving the enrichment results for this location and service, and the error properties will be set.

**ErrorCode** Int32 (nullable)

This is an internal error code that you can use for logging purposes.

**Error** String (nullable)

If the enrichment results for the input location and the requested service could not be retrieved, this property will contain an error message that you can use to identify the source of the problem.

A typical example for this case is insufficient geocoding accuracy (geoquality) for high-resolution hazards like riverflood.

# Common Model

## *Coordinate*

**CountryCode**

String (optional)

For requests:
You can pass an ISO 3166 country code (alpha-2 or alpha-3) to identify the country of the location. If you don't set this value, we will determine the country based on longitude and latitude.

For responses:
This field will contain the ISO 3166 country code (alpha-3) we found for the input coordinates.

**Longitude**

Double

For requests:
This field must contain the WGS84 longitude value in decimal degrees (for example, -77.0089). The value must be in the range from -180 to 180. Values will be rounded to 6 decimal places as more digits don't provide additional accuracy (already down to max. 10cm for the 6$^{th}$ digit).

For responses:
This field will be null as we cannot provide you with the geocoding results due to the licensing terms of our geocoding provider.

**Latitude**

Double

For requests:
This field must contain the WGS84 latitude value in decimal degrees (for example, 38.8897). The value must be in the range from -90 to 90. Values will be rounded to 6 decimal places as more digits don't provide additional accuracy (already down to max. 10cm for the 6$^{th}$ digit).

For responses:
This field will be null as we cannot provide you with the geocoding results due to the licensing terms of our geocoding provider.

**Geoquality** Integer

This field indicates the quality of the geocoding process with a value between 0 (worst) and 100 (best). The possible values are:
0 = not found
20 = country
40 = administration level
50 = city / Zip code
90 = street
95 = street and house number
100 = exact coordinates

For requests:
You will typically not set this value. When you send coordinates, we automatically assume the highest geoquality of 100.

For responses:
When you send an address, this field contains the geoquality of the geocoding process in our backend.

*Address*

**Id** Int64 (obsolete)
This value is not in use anymore. Please ignore it.

**Street** String (optional)

The street name without the house number
(Use this field if you store street and house number separately in your system)

**HouseNr** String (optional)

The house number
(Use this field if you store street and house number separately in your system)

**StreetHouseNr** String (optional)

The combined street name and house number
(Use this field if you store street name and house number in one combined field in your system)

**ZipCode** String (required)

This zip code or postcode

**District** String (optional)

The name of the district

**City** String (required)

The city name

**Region** String (optional)

The name of the region

| | |
|---|---|
| **CountryCode** | String (optional)<br><br>For requests:<br>You can pass an ISO 3166 country code (alpha-2 or alpha-3) to identify the country of the location. If you don't set this value, we will determine the country based on the CountryName.<br><br>For responses:<br>This field will contain the ISO 3166 country code (alpha-3) we found for the input address. |
| **CountryName** | String (optional)<br><br>For requests:<br>Our geocoding process will recognize multiple spellings for the country name in different languages. The unambiguous CountryCode is, however, the safer option.<br><br>For responses:<br>This field will contain the English name of the country. |

# Batch Processing (Asynchronous Execution)

## Overview

Normal POST requests to the location-info endpoint work synchronously. This means you send a request, and the API response directly contains the enrichment results (scores). This type of processing is meant for smaller amounts and real-time integration into your workflows.

To process larger quantities of locations (typically more than 10k), it is best to use the `location-info/jobs` endpoint. Here, the requests are added to a job queue and processed asynchronously. This allows the API to optimize resource allocations and maximize throughput.

You can then continuously monitor the progress of your requests and retrieve the results as soon as they are done.

## Submitting Batch Jobs

The synchronous `location-info` endpoint and the asynchronous `location-info/jobs` endpoint use the same request and response model. Also, the same request size limit (max. 1000 locations) applies to both types. Batch requests could theoretically be larger, but the responses would get very large and probably hit HTTP size limits.

Therefore, you must split your batch of locations into individual requests with up to 1000 locations each.

You then send your requests via a POST to the `location-info/jobs` endpoint. This is completely similar to the synchronous location scoring (see Location Scoring (Resource location-info)).

Instead of the enrichment results, you will receive a job id (numeric, Int64). You can use this id to check the job's progress and retrieve the results later.

## Checking the Job Status

After you have added your requests to the job queue, you can query the progress by sending a GET request to the `location-info/jobs` endpoint.

```
GET - https://api.munichre.com/nathan/v2/location-info/jobs
```

The result of this request is an array of `QueuedRequestStatus` objects that looks like this:

```
[
   {
      "startTimeUtc":"2024-03-23T14:24:34.84716+00:00",
      "endTimeUtc":"2024-03-23T14:27:34+00:00",
      "jobStatus":"Finished",
      "id":68100052
   },
   {
      "startTimeUtc":"2024-03-23T14:24:34.84716+00:00",
      "jobStatus":"Working",
      "id":68100053
   }
]
```

You will receive an item in this array for each request you submit.

A `QueuedRequestStatus` object contains these fields:

| | |
|---|---|
| **Id** | Int64<br>This is the job id you received when submitting the queued request. |
| **StartTimeUtc** | Timestamp (UTC)<br>This is the time the request was submitted to the job queue. |
| **EndTimeUtc** | Timestamp (UTC, nullable)<br>This is the time the API finished processing your request. |
| **JobStatus** | JobStatus enum<br>This enumeration indicates the current status of your queued request. |
| **ErrorMessage** | If an error occurs during the processing of your job, this field will contain a message indicating the source of the problem. |

The `JobStatus` enum contains these values:

- Pending
  The request is still waiting in the queue for available processing slots.

- Working
  The request is currently being processed.

- Finished
  The request has been processed correctly, and the results can be retrieved.

- Canceled
  The caller has canceled the request processing.

- Error
  Something went wrong while processing the request. See the `ErrorMessage` property in the `QueuedRequestStatus` object.

You can continuously send GET requests to the `location-info/jobs` endpoint (for example, every 10 seconds) until all queued requests have been processed.

## Collecting the Results

As soon as a queued request goes into the `Finished` status, you can collect the results. You do this by sending a GET request to `/location-info/jobs/{jobId}`. For example:

```
GET - https://api.munichre.com/nathan/v2/location-info/jobs/68100052
```

The result is a normal LocationInfoResponse (similar to the synchronous location-info endpoint).

Please note that the results are deleted from the API immediately after you retrieve them. Repeated calls to the same job ID will result in HTTP 404 errors (not found).

## Cancelling Jobs

You can cancel the unfinished jobs (`Pending` or `Working` status) by sending a PUT request to `/location-info/jobs/{jobId}/cancel`. For example:

```
PUT - https://api.munichre.com/nathan/v2/location-info/jobs/68100053/cancel
```

The result of this call should be HTTP status 204 (no content).

Munich RE

# Area & Line Scoring (Resource area-line-info)

## Overview

The resource `area-line-info` contains the endpoints to retrieve the scores (enrichment), such as natural hazards, risk scores, and climate change data for area or line objects identified by geospatial geometries.

This chapter covers the following use cases:

- Requesting scores for areas and lines
- Batch processing of larger quantities
- Getting data documentation for the scores

## Requesting Scores (Synchronous Execution)

### Basic Request & Response Structure

*Request*

To retrieve scores, you send a POST request to the `area-line-info` endpoint. Such a request contains two main parts:

- An array of **items** = area or line objects (identified by geospatial geometries)
- An array of **services** = the required scores such as natural hazards, risk scores, or climate change data

A sample request looks like this:

```
POST - https://api.munichre.com/nathan/v2/area-line-info
```

```
{
   "items":[
      {
         "id":"1",
         "geometry":{
            "type":"Polygon",
            "coordinates":[
               [
                  [
                     -82.738037,
                     28.207709
                  ],
                  [
                     -82.75589,
                     28.212953
```

```
                ],
                [
                   -82.761154,
                   28.201457
                ],
                [
                   -82.778778,
                   28.181284
                ],
                [
                   -82.742844,
                   28.178862
                ],
                [
                   -82.709656,
                   28.186731
                ],
                [
                   -82.695694,
                   28.174424
                ],
                [
                   -82.677841,
                   28.190765
                ],
                [
                   -82.661819,
                   28.199238
                ],
                [
                   -82.683334,
                   28.21255
                ],
                [
                   -82.726135,
                   28.225861
                ],
                [
                   -82.738037,
                   28.207709
                ]
             ]
          ]
       }
    }
 ],
 "services":[
    {
       "category":"Hazards",
       "service":"NathanEarthquake",
       "version":"Current",
       "requestedServiceId":"Hazards__NathanEarthquake"
    },
    {
       "category":"Hazards",
       "service":"NathanRiverFlood",
       "version":"Current",
       "requestedServiceId":"Hazards__NathanRiverFlood"
```

```
      }
   ]
}
```

Requests can contain up to 10 items and an unlimited number of services.

Each element in the items array needs a unique identifier to connect the requested area or line object with the results in the API response. This identifier can be any string, for example, a primary key from your backend system.

The API scores are organized into categories and services. To request a specific score, you need these four values:

- category
  (for example, `"General"`, `"Hazards"`, `"ClimateChange"`)
- service
  (for example, `"PopulationDensity"`, `"NathanEarthquake"`, `"HeatStressIndexSSP126Y2050"`)
- version
  All data in the API is versioned. You can request a specific version or simply use the constant `"current"` to always get the latest version (recommended).
- requestedServiceId
  This must be a unique string that connects the requested service (identified by category + service + version) with the corresponding result in the API response (see the example response below).

To learn more about the available categories and services, see Discovering the Available Services and Data Documentation.

For detailed information on area-line-info requests, see Request Model.

### *Response*

The API response contains an array of `items` that match the input items by the `id` (one element per requested area or line object).

Each item contains an array of `results` that match the requested services by the `requestedServiceId` (one element per requested service).

The API response to the above sample request looks like this:

```
{
   "items":[
      {
         "id":"1",
         "results":[
            {
               "requestedServiceId":"Hazards__NathanEarthquake",
               "field":"HazardZone",
               "resultItems":[
                  {
                     "value":0,
                     "percent":100
                  }
               ]
            },
```

```
            {
               "requestedServiceId":"Hazards__NathanRiverFlood",
               "field":"HazardZone",
               "resultItems":[
                  {
                     "value":0,
                     "percent":37.44
                  },
                  {
                     "value":50,
                     "percent":29.82
                  },
                  {
                     "value":100,
                     "percent":6.13
                  },
                  {
                     "value":500,
                     "percent":26.61
                  }
               ]
            }
         ],
         "areaInSquareKilometers":34.4691
      }
   ],
   "requestId":391504183,
   "processingTimeInMilliseconds":133
}
```


The actual scores are contained in the `resultItems` property for each element in the `results` array. This array shows the percentage distribution of the area or line object for the requested service (for example, a natural hazard like riverflood). Each element of this array contains these fields:

- Value
  This is the zone for the requested service (for example, 100 for the 100-year return period of a riverflood hazard)
- Percent
  This is the percentage of the area or line object that falls into the zone value.

For detailed information on area-line-info responses, see Response Model.

## Discovering the Available Services and Data Documentation

The available enrichment services (scores) depend on your licensed data edition. You have two possibilities to get detailed information about these services:

### *Structured data*

A GET request to the `configuration` endpoint returns a comprehensive JSON object containing detailed data about the available API content.

Please see the section API Configuration Data for in-depth descriptions of this data.

*PDF data documentation*

Send a POST request to the `area-line-info/documentation` endpoint to obtain the human-readable data documentation in PDF format.

Here is an example request for the data documentation:

```
POST - https://api.munichre.com/nathan/v2/area-line-info/documentation
```

```
{
   "ValueMapping":"Traffic Light"
}
```

This request body is a DocumentationReportOptions object:

| | |
|---|---|
| **ValueMapping** | String (optional)<br>The name of a ValueMapping category like `"Traffic Light"`. If this option is included, the documentation will include information on how the regular API scores (for example, hazard zones) are mapped to the given ValueMapping. |

This data documentation PDF is always generated on the fly using the latest data based on your licensed data edition. The data you get from the `configuration` endpoint is used to build the document.

## Including Value Mappings

Value mappings are different interpretations of the API scoring results. A typical example is mapping hazard zones to traffic light scores.

You can include value mappings for your requested scores by setting the `ValueMappings` property in the request. This property is an array of strings with the names of the value mappings:

```
{
   "valueMappings": ["Traffic Light"],
   "items":[
      {
         "id":"1",
         "geometry":{
            "type":"Polygon",
            "coordinates":[
               [
                  [
                     -82.738037,
                     28.207709
                  ],
                  [
                     -82.75589,
                     28.212953
                  ],
                  [
                     -82.761154,
                     28.201457
                  ],
                  [
                     -82.778778,
                     28.181284
```

```
                  ],
                  [
                     -82.742844,
                     28.178862
                  ],
                  [
                     -82.709656,
                     28.186731
                  ],
                  [
                     -82.695694,
                     28.174424
                  ],
                  [
                     -82.677841,
                     28.190765
                  ],
                  [
                     -82.661819,
                     28.199238
                  ],
                  [
                     -82.683334,
                     28.21255
                  ],
                  [
                     -82.726135,
                     28.225861
                  ],
                  [
                     -82.738037,
                     28.207709
                  ]
               ]
            ]
         }
      }
   ],
   "services":[
      {
         "category":"Hazards",
         "service":"NathanEarthquake",
         "version":"Current",
         "requestedServiceId":"Hazards__NathanEarthquake"
      },
      {
         "category":"Hazards",
         "service":"NathanRiverFlood",
         "version":"Current",
         "requestedServiceId":"Hazards__NathanRiverFlood"
      }
   ]
}
```

The response will then contain a new property, `MappedResults`, for each requested area or line object. This is a dictionary with the following structure:

- Key = name of the value mapping
- Value = Array of results similar to the normal results for the zone values.

For the request above, the response then looks like this:


```
{
    "items": [
        {
            "id": "1",
            "results": [
                {
                    "requestedServiceId": "Hazards__NathanEarthquake",
                    "field": "HazardZone",
                    "resultItems": [
                        {
                            "value": 0,
                            "percent": 100
                        }
                    ]
                },
                {
                    "requestedServiceId": "Hazards__NathanRiverFlood",
                    "field": "HazardZone",
                    "resultItems": [
                        {
                            "value": 0,
                            "percent": 37.44
                        },
                        {
                            "value": 50,
                            "percent": 29.82
                        },
                        {
                            "value": 100,
                            "percent": 6.13
                        },
                        {
                            "value": 500,
                            "percent": 26.61
                        }
                    ]
                }
            ],
            "mappedResults": {
                "Traffic Light": [
                    {
                        "requestedServiceId": "Hazards__NathanEarthquake",
                        "field": "HazardZone",
                        "resultItems": [
                            {
                                "value": 1,
                                "percent": 100
                            }
                        ]
                    },
```

```
                    {
                        "requestedServiceId": "Hazards__NathanRiverFlood",
                        "field": "HazardZone",
                        "resultItems": [
                            {
                                "value": 1,
                                "percent": 37.44
                            },
                            {
                                "value": 3,
                                "percent": 26.61
                            },
                            {
                                "value": 4,
                                "percent": 6.13
                            },
                            {
                                "value": 5,
                                "percent": 29.82
                            }
                        ]
                    }
                ]
            },
            "areaInSquareKilometers": 34.4691
        }
    ],
    "requestId": 404304213,
    "processingTimeInMilliseconds": 373
}
```


Please note:

Not all zone results support all value mappings. For example, most services in the “General”-category, like “NathanElevation” or “PopulationDensity”, have no Traffic Light mapping. You will not get mapped results for these services.

Requesting value mappings has little impact on processing times but significantly increases the response size.

If you don’t need the default zone results at all, you can use the property `ExcludeZoneResults` to remove them from the response:

```
{
   "valueMappings": ["Traffic Light"],
   "excludeZoneResults": true,
   "items":[ … ],
   "services":[ … ]
}
```

In this case, the response items will only contain the mapped results. You can also only request services that support all selected value mappings.

The available value mappings and their definitions are also available through the `configuration` endpoint of the API. You can find detailed information in the chapter API Configuration Data (Resource configuration).

## Error Handling

Enrichment requests to the area-line-info endpoint can lead to three basic kinds of errors:

- Error on the request level
- Error on the item (area or line object) level
- Error on the service (score) level

Here are some examples of these cases:

### *Error on the request level*

In case of structural errors in the request, you will receive a 400 HTTP response status (bad request) indicating the problem. For example, a typo in a requested service name prevents the execution and leads to an error message like this:

```
{
   "title":"Request Validation Error",
   "status":400,
   "detail":"Requested category Hazards does not contain the requested service
    NathanEarthquark"
}
```

Please refer to the `"title"` and `"detail"` properties to determine the cause of the problem.

### *Error on the item level*

A request can contain between 1 and 10 area or line objects. It's possible that some items of one request contain errors and others work. A typical problem would be an invalid geometry.

In these cases, the HTTP response code will still indicate success (200), and you can check the individual items in the response.

Here is an example response to a request containing two areas. One polygon is valid ("1"), the other one ("2") has an invalid geometry:

```
{
   "items":[
      {
         "id":"1",
         "results":[
            …
         ]
      },
      {
         "id":"2",
         "errorCode":10,
         "error":"Error reading GeoJSON: points must form a closed linestring"
      }
      …
}
```

If there is an error on the item level, you will get an `errorCode` and an `error` message indicating the source of the problem instead of the results array with the scores.

### *Error on the service (score) level*

It's possible that sometimes individual scores cannot be retrieved for certain area or line objects. A typical example would be a geometry covering an area where a requested service has no data.

Here is an example response for an input geometry over Antarctica:


```
{
   "items":[
      {
         "id":"1",
         "results":[
          …
                {
                    "requestedServiceId": "Hazards_NathanRiverFlood",
                    "field": "HazardZone",
                    "errorCode": 41,
                    "error": "No information at specified location. Service does not
cover these countries: ATA."
                },
          …
```


If there is an error on the result level, you will get an `errorCode` and an `error` message indicating the source of the problem instead of the `resultItems` array with the percentage distribution of the scores.

## Limits

### *Limits for area-line-info requests*

The following limits apply for area-line-info enrichment requests:

- Up to 10 area or line objects per request
- An area or line object can have a maximum area of 50000 square kilometers
- No limit for the number of requested services (scores)
- No limit for the request and response sizes. Please bear in mind that responses to requests with very complex geometries services can get quite large, and you might run into restrictions from your network settings. Split the requests into smaller numbers of items if you encounter such limitations. Always include all required services in a request.

### *Connection limits and rate limits*

The API has no connection or rate limits. The maximum number of area-line-info requests that can be executed in parallel before experiencing reduced performance depends on the type and complexity of the individual requests. Especially large geometries cause increased processing times.

Moreover, the API is equipped with specialized batch endpoints. These endpoints are designed to handle larger quantities of data, ensuring optimal throughput (see Batch Processing (Asynchronous Execution)).

## Request Model

*AreaLineInfoRequest*

| | |
|---|---|
| **Items** | Array of **AreaLineInfoRequestItem** objects (required)<br>This is the list of the area or line objects for the enrichment. |
| **Services** | Array of **AreaLineInfoRequestService** objects (required)<br>This is the list of the required services (scores) for the enrichment. |
| **ValueMappings** | String array (optional)<br>You can include value mappings like Traffic Light for your requested services in the response. This property is an array of strings with the names of the value mappings.<br><br>The response will then contain a new property, MappedResults, for each requested item. This is a dictionary with the following structure:<br><br>Key = name of the value mapping<br>Value = Array of results similar to the normal results for the zone values |
| **ExcludeZoneResults** | Boolean (optional)<br>If you requested one or more ValueMappings and don't need the normal zone results, you can set this property to true to remove them from the response. |
| **UserId** | String (optional)<br>You can pass an identifier, which we will add to your API usage statistics for further analytics. This could be useful if you have multiple systems that use our API on your side.<br><br>Another typical use case is for API resellers to pass a key to identify the client who initiated the request.<br><br>Please note:<br>This field is not intended to pass sensitive information about actual end users, such as names or mail addresses. |
| **TransactionId** | String (optional)<br>You can pass an identifier for the transaction that initiated the API request on your side. This could be a primary key in your database or a log entry ID. We store this TransactionId in our logs, and it can be used to cross-reference our usage statistics with your data. |

*AreaLineInfoRequestItem*

| | |
|---|---|
| **Id** | String (required)<br><br>Each area or line object in the request requires a unique identifier to connect the API response results to the requested item. There are no guidelines for this id. You can use any value that helps you connect the API response data to the asset in your system (for example, a primary key from your backend). |

**Geometry** A geometry object (GeoJSON, optional)

This object contains the geospatial geometry data in GeoJSON format.

An AreaLineInfoRequestItem must have either a Geometry object or a WKT string– not both.

**Wkt** String (optional)

In this property, you can pass the geometry data as a WKT (well-known text) string.

**BufferInMeters** Double (optional)

You can specify a value for a buffer that will be applied to your input geometry.

If you don’t specify a buffer, the following defaults will be applied:

For point or line geometries: 15 meter buffer

For all other geometry types (for example, polygons): no buffering

*AreaInfoRequestService*

**Category** String (required)

The name of the enrichment category
(for example, "General", "Hazards", "ClimateChange")

**Service** String (required)

The name of the enrichment service
(for example, "PopulationDensity", "NathanEarthquake", "HeatStressIndexSSP126Y2050")

**Version** String (required)

The name of the enrichment service version

All data in the API is versioned. You can request a specific version identified in the format YYYYMMDD or simply use the constant "Current" to always get the latest version (recommended).

**RequestedServiceId** String (required)

This must be a unique string that connects the requested service (identified by category + service + version) with the corresponding result in the API response.

There are no guidelines on how to build this id. For example, you can use an identifier that corresponds to a primary key in your backend. Or you can simply combine the strings for Category and Service like “Hazards__NathanEarthquake”.

This identifier is mainly required to reduce the response size by avoiding repeating the three values for category, service, and version for every area or line result.

## Response Model

*AreaLineInfoResponse*

| | |
|---|---|
| **ResultsAreFalsified** | Bool (nullable)<br>This property will be set to true if you use an API test account that delivers falsified results.<br>If the results are not falsified, this property will be null. |
| **Items** | An array of **AreaLineInfoResponseItem** objects<br>This array contains one object per input item with the enrichment results. |
| **RequestId** | Int64<br>This is our internal identifier for the request. You can store this on your side to cross-reference your logs with ours. |
| **ProcessingTimeInMilliseconds** | Int32<br>This is the time the API spent processing your request (without network overhead). |

*AreaLineInfoResponseItem*

| | |
|---|---|
| **Id** | String<br>The value of this property matches the corresponding Id for the area or line object in the items array of the AreaLineInfoRequest. |
| **Results** | An array of **AreaLineInfoResult** objects (nullable)<br>This array contains one object per requested service and field with the enrichment results.<br>If this array is null, something went wrong with the input geometry, and the error properties will be set. |
| **MappedResults** | A Dictionary with<br>Key = String (name of the requested ValueMapping) and<br>Value = Array of **AreaLineInfoResult** objects<br>(nullable)<br>If you requested one or more ValueMappings, this property will contain a dictionary with the mapped results. |
| **AreaInSquareKilometers** | Double (nullable)<br>This is the total area of the input area or line object in square kilometers. |
| **LengthInKilometers** | Double (nullable)<br>If the input item was a line or a multiline, this property will contain the total length of the geometry. |

**ErrorCode** Int32 (nullable)

This is an internal error code that you can use for logging purposes.

**Error** String (nullable)

If the input item contains an error (for example, invalid GeoJSON), this property will contain a message that you can use to identify the source of the problem.

*AreaLineInfoResult*

**RequestedServiceId** String

The value of this property matches the corresponding Id for the requested service in the in the services array of the AreaLineInfoRequest.

**Field** String

This is the field name for the resultItems array with the percentage distribution (for example, “HazardZone”)

The field names and types vary between the different API services. You can check the data documentation or use the API configuration endpoint to discover the available services and their fields.

**ResultItems** Array of **AreaLineInfoResultItem** objects

This array contains the percentage distribution for the current field.

If this property is null, then something went wrong while retrieving the enrichment results for this location and service, and the error properties will be set.

**ErrorCode** Int32 (nullable)

This is an internal error code that you can use for logging purposes.

**Error** String (nullable)

If the enrichment results for the input area or line object and the requested service could not be retrieved, this property will contain an error message that you can use to identify the source of the problem.

*AreaLineInfoResultItem*

**Value** Object

This is the enrichment result (score), for example, a HazardZone

| | |
|---|---|
| **Percent** | Double<br>The percentage of the area or line object that falls into the current value (score) |

# Batch Processing (Asynchronous Execution)

## Overview

Normal POST requests to the area-line-info endpoint work synchronously. This means you send a request, and the API response directly contains the enrichment results (scores). This type of processing is meant for smaller amounts and real-time integration into your workflows.

To process larger quantities of areas or lines (typically more than 100), it is best to use the `area-line-info/jobs` endpoint. Here, the requests are added to a job queue and processed asynchronously. This allows the API to optimize resource allocations and maximize throughput.

You can then continuously monitor the progress of your requests and retrieve the results as soon as they are done.

## Submitting Batch Jobs

The synchronous `area-line-info` endpoint and the asynchronous `area-line-info/jobs` endpoint use the same request and response model. Also, the same request size limit (max. 10 items) applies to both types. Batch requests could theoretically be larger, but the requests or responses would get very large and probably hit HTTP size limits.

Therefore, you must split your batch of locations into individual requests with up to 10 items each.

You then send your requests via a POST to the `area-line-info/jobs` endpoint. This is completely similar to the synchronous area-line scoring (see Area & Line Scoring (Resource area-line-info)).

Instead of the enrichment results, you will receive a job id (numeric, Int64). You can use this id to check the job's progress and retrieve the results later.

## Checking the Job Status

After you have added your requests to the job queue, you can query the progress by sending a GET request to the `area-line-info/jobs` endpoint.

```
GET - https://api.munichre.com/nathan/v2/area-line-info/jobs
```

The result of this request is an array of `QueuedRequestStatus` objects that looks like this:

```
[
   {
      "startTimeUtc":"2024-03-23T14:24:34.84716+00:00",
      "endTimeUtc":"2024-03-23T14:27:34+00:00",
      "jobStatus":"Finished",
      "id":68100052
   },
```

```
   {
      "startTimeUtc":"2024-03-23T14:24:34.84716+00:00",
      "jobStatus":"Working",
      "id":68100053
   }
]
```

You will receive an item in this array for each request you submit.

A `QueuedRequestStatus` object contains these fields:

| | |
|---|---|
| **Id** | Int64<br>This is the job id you received when submitting the queued request. |
| **StartTimeUtc** | Timestamp (UTC)<br>This is the time the request was submitted to the job queue. |
| **EndTimeUtc** | Timestamp (UTC, nullable)<br>This is the time the API finished processing your request. |
| **JobStatus** | JobStatus enum<br>This enumeration indicates the current status of your queued request. |
| **ErrorMessage** | If an error occurs during the processing of your job, this field will contain a message indicating the source of the problem. |

The `JobStatus` enum contains these values:

- Pending
  The request is still waiting in the queue for available processing slots.

- Working
  The request is currently being processed.

- Finished
  The request has been processed correctly, and the results can be retrieved.

- Canceled
  The caller has canceled the request processing.

- Error
  Something went wrong while processing the request. See the `ErrorMessage` property in the `QueuedRequestStatus` object.

You can continuously send GET requests to the `area-line-info/jobs` endpoint (for example, every 10 seconds) until all queued requests have been processed.

## Collecting the Results

As soon as a queued request goes into the `Finished` status, you can collect the results. You do this by sending a GET request to `area-line-info/jobs/{jobId}`. For example:

```
GET - https://api.munichre.com/nathan/v2/area-line-info/jobs/68100052
```

The result is a normal LocationInfoResponse (similar to the synchronous area-line-info endpoint).

Please note that the results are deleted from the API immediately after you retrieve them. Repeated calls to the same job ID will result in HTTP 404 errors (not found).

## Cancelling Jobs

You can cancel the unfinished jobs (`Pending` or `Working` status) by sending a PUT request to `/area-line-info/jobs/{jobId}/cancel`. For example:

```
PUT - https://api.munichre.com/nathan/v2/area-line-info/68100053/jobs/cancel
```

The result of this call should be HTTP status 204 (no content).

Munich RE

# Map Services (Resource maps)

## Overview

You can connect your web mapping applications or GIS platform with the Location Risk Intelligence Hazard and Risk Maps. The `maps` endpoint provides access to our map layers through these protocols:

- **Tile Map Service (TMS)**
  (https://wiki.osgeo.org/wiki/Tile_Map_Service_Specification)

- **Web Map Tile Service (WMTS)**
  (https://www.ogc.org/standard/wmts/)

- **Web Map Service (WMS)**
  (https://www.ogc.org/standard/wms/)

All our map layers have pre-generated caches for optimal performance. The map tiles are available in two projections:

- **EPSG:4326** (WGS 84)

- **EPSG:900913** (Google Maps Global Mercator -- Spherical Mercator)
  (compatible to EPSG:3857, Web Mercator)

The parameters for map requests are usually not created manually. Instead, they are generated dynamically by a client library for mapping applications or a GIS platform. Typical examples of such libraries or platforms are:

- OpenLayers (https://openlayers.org/, open source)
- Leaflet (https://leafletjs.com/, open source)
- Esri ArcGIS Maps SDK for JavaScript (https://developers.arcgis.com/javascript/latest/)
  (commercial license required)
- Esri Portal for ArcGIS
  (commercial license required)

This documentation cannot cover every possible library and use case. Still, it includes some sample snippets for OpenLayers, Leaflet, and the Esri ArcGIS Maps SDK for JavaScript as well as guidelines on integrating the map layers into Portal for ArcGIS.

This chapter covers the following topics:

- Using LRI Map Layers as Tile Map Service (TMS) (Recommended Approach)
- Using LRI Map Layers as Web Map Tile Service (WMTS)
- Using LRI Map Layers as Web Map Service (WMS)
- Using a Proxy to Connect Your GIS Client Applications
- Discovering the Available Services and Documentation

# Using LRI Map Layers as Tile Map Service (TMS) (Recommended Approach)

## TMS Request Structure

### *Requesting Map Tiles*

The URL structure for a TMS request to fetch a map tile is:

```
GET -
https://api.munichre.com/nathan/v2/
maps/services/gwc/service/wmts/rest/{category}:{service}/{projection}/
{projection}:{z}/{y}/{x}?format=image/png
```

You need to replace these placeholders with your values:

- category
  For example, `General`, `Hazards`, `ClimateChange`)
- service
  For example, `PopulationDensity`, `NathanEarthquake`, `HeatStressIndexSSP126Y2050`)
- projection
  `EPSG:4326` or `EPSG:900913`
  (For web-based mapping applications, you typically use `EPSG:900913`)
- z = tile matrix
- x/y = column and row of the map tile

For information on how to get the available categories and services, please refer to Discovering the Available Services and Documentation.

Here is an example to request a tile for the `NathanRiverFlood` service in the `Hazards` category:

```
GET -
https://api.munichre.com/nathan/v2/
maps/services/gwc/service/wmts/rest/Hazards:NathanRiverFlood/EPSG:900913/
EPSG:900913:3/1/0?format=image/png
```

The request parameters are typically not added manually. Below are some examples of how to use the LRI map layers with client libraries and GIS platforms.

### *Service Metadata (GetCapabilities)*

You can also get the service metadata as a WMTS GetCapabilities response with this request:

```
GET -
https://api.munichre.com/nathan/v2/
maps/services/gwc/service/wmts/rest/WMTSCapabilities.xml
```

This response is usable by clients that support WMTS service metadata.

## Using the TMS Map Services in Your Applications

### *OpenLayers*

To add the LRI layers with the TMS format, use the `TileLayer` object with an `XYZ` source in your OpenLayers application.

The following snippet adds an OpenStreetMap base map and the `NathanRiverFlood` layer in the category `Hazards`:

```
import Map from 'ol/Map.js';
import OSM from 'ol/source/OSM.js';
import TileLayer from 'ol/layer/Tile';
import View from 'ol/View.js';
import XYZ from 'ol/source/XYZ';

const layers = [
  new TileLayer({
    source: new OSM(),
  }),
  new TileLayer({
    opacity: 0.6,
    source: new XYZ({
      attributions: 'Risk Management Partners, Munich Re',
      url:
https://api.munichre.com/nathan/v2/maps/services/gwc/service/wmts/rest/Hazards:Natha
nRiverFlood/EPSG:900913/EPSG:900913:{z}/{y}/{x}?format=image/png',
    }),
  }),
];
const map = new Map({
  layers: layers,
  target: 'map',
  view: new View({
    center: [1300000, 3300000],
    zoom: 2,
  }),
});
```

You will need to replace the `https://api.munichre.com/nathan/v2/` part with the URL of your proxy server, as web clients cannot handle the LRI API authentication (see Using a Proxy to Connect Your GIS Client Applications).

### *Leaflet*

To add the LRI layers with the TMS format in your Leaflet application, use the `L.tileLayer` object.

The following snippet adds an OpenStreetMap base map and the `NathanRiverFlood` layer in the category `Hazards`:

```
      const map = L.map('map').setView([0, 0], 2);

      const tiles = L.tileLayer('https://tile.openstreetmap.org/{z}/{x}/{y}.png', {
            maxZoom: 19,
            attribution: '&copy; <a
href="http://www.openstreetmap.org/copyright">OpenStreetMap</a>'
      }).addTo(map);

      var wmtsLayer = L.tileLayer(
'https://api.munichre.com/nathan/v2/maps/services/gwc/service/wmts/rest/Hazards:Nath
anRiverFlood/EPSG:900913/EPSG:900913:{z}/{y}/{x}?format=image/png', {
      attribution: 'Risk Management Partners, Munich Re',
      opacity: 0.6
      }).addTo(map);
```

You will need to replace the `https://api.munichre.com/nathan/v2/` part with the URL of your proxy server, as web clients cannot handle the LRI API authentication (see Using a Proxy to Connect Your GIS Client Applications).

### *ArcGIS Maps SDK for JavaScript*

To add the LRI layers with the TMS format, use the `WebTileLayer`object with the required `urlTemplate` in your application.

The following snippet adds an OpenStreetMap base map and the `NathanEarthquake` layer in the category `Hazards`:

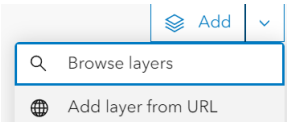
```
    <script>
      require(["esri/Map", "esri/Basemap", "esri/views/MapView",
"esri/layers/WebTileLayer", "esri/layers/OpenStreetMapLayer"], (Map, Basemap,
MapView, WebTileLayer, OpenStreetMapLayer) => {

        const osmLayer = new OpenStreetMapLayer();

        const basemap = new Basemap( {
          baseLayers: [osmLayer],
        });

        const map = new Map({
          basemap: basemap
        });

        const view = new MapView({
        map: map,
        container: "viewDiv"
        });

        // add LRI map layer
        const lriLayer = new WebTileLayer({
```

```
          urlTemplate:
"https://api.munichre.com/nathan/v2/maps/services/gwc/service/wmts/rest/Hazards:Nath
anEarthquake/EPSG:900913/EPSG:900913:{z}/{y}/{x}?format=image/png",
          opacity: 0.6,
          copyright: 'Risk Management Partners, Munich Re'
        });
        map.add(lriLayer);
      });
    </script>
```

You will need to replace the `https://api.munichre.com/nathan/v2/` part with the URL of your proxy server, as web clients cannot handle the LRI API authentication (see Using a Proxy to Connect Your GIS Client Applications).

### *Portal for ArcGIS*

To add the LRI layers with the TMS format in Portal for ArcGIS, select the option “Add layer from URL”:




Enter the URL to retrieve the WMTS capabilities:

```
https://api.munichre.com/nathan/v2/
maps/services/gwc/service/wmts/rest/WMTSCapabilities.xml
```

Portal for ArcGIS will automatically detect the type WMTS (OGC). Click “Next” to retrieve the list of available services.

Select the required layer from the list and the projection (typically EPSG: 900913), then click “Add to map”.

You will need to replace the `https://api.munichre.com/nathan/v2/` part with the URL of your proxy server, as Portal for ArcGIS cannot handle the LRI API authentication (see Using a Proxy to Connect Your GIS Client Applications).

# Using LRI Map Layers as Web Map Tile Service (WMTS)

## WMTS Request Structure

### *Requesting Map Tiles*

The URL structure for a WMTS request to fetch a map tile is:

```
GET -
https://api.munichre.com/nathan/v2/
/maps/services/gwc/service/wmts?Service=WMTS&Request=GetTile&Version=1.0.0&layer={ca
tegory}:{service}&style=default&tilematrixset={projection}&Format=image/png&TileMatr
ix={projection}:{z}&TileCol={x}&TileRow={y}
```

You need to replace these placeholders with your values:

- category
  For example, `General`, `Hazards`, `ClimateChange`)
- service
  For example, `PopulationDensity`, `NathanEarthquake`, `HeatStressIndexSSP126Y2050`)
- projection
  `EPSG:4326` or `EPSG:900913`
  (For web-based mapping applications, you typically use `EPSG:900913`)
- z = tile matrix
- x/y = column and row of the map tile

For information on how to get the available categories and services, please refer to Discovering the Available Services and Documentation.

Here is an example to request a tile for the `NathanRiverFlood` service in the `Hazards` category:

```
GET -
https://api.munichre.com/nathan/v2/
/maps/services/gwc/service/wmts?Service=WMTS&Request=GetTile&Version=1.0.0&layer=Hazards:NathanRiverFlood&style=default&tilematrixset=EPSG:900913&Format=image/png&TileMatrix=EPSG:900913:3&TileCol=3&TileRow=4
```

The request parameters are typically not added manually. Below are some examples of how to use the LRI map layers with client libraries and GIS platforms.

### *Service Metadata (GetCapabilities)*

You can also get the service metadata as a WMTS GetCapabilities response with this request:

```
GET -
https://api.munichre.com/nathan/v2/
/maps/services/gwc/service/wmts?SERVICE=WMTS&request=GetCapabilities&version=1.0.0
```

This response is usable by clients that support WMTS service metadata.

## Using the WMTS Map Services in Your Applications

### *OpenLayers*

To add the LRI layers with the WMTS format, use the `TileLayer` object with a `WMTS` source in your OpenLayers application.

The following snippet adds an OpenStreetMap base map and the `NathanRiverFlood` layer in the category `Hazards`:

```
import Map from 'ol/Map.js';
import OSM from 'ol/source/OSM.js';
import TileLayer from 'ol/layer/Tile.js';
import View from 'ol/View.js';
import WMTS from 'ol/source/WMTS.js';
import WMTSTileGrid from 'ol/tilegrid/WMTS.js';
import {get as getProjection} from 'ol/proj.js';
import {getTopLeft, getWidth} from 'ol/extent.js';

const projection = getProjection('EPSG:900913');
```

```
const projectionExtent = projection.getExtent();
const size = getWidth(projectionExtent) / 256;
const resolutions = new Array(19);
const matrixIds = new Array(19);
for (let z = 0; z < 19; ++z) {
  // generate resolutions and matrixIds arrays for this WMTS
  resolutions[z] = size / Math.pow(2, z);
  matrixIds[z] = "EPSG:900913" + ":" + z;
}

const layers = [
  new TileLayer({
    source: new OSM(),
  }),
  new TileLayer({
    opacity: 0.6,
    source: new WMTS({
      attributions: 'Risk Management Partners, Munich Re',
      url: https://api.munichre.com/nathan/v2/maps/services/gwc/service/wmts',
      layer: 'Hazards:NathanRiverFlood',
      matrixSet: 'EPSG:900913',
      format: 'image/png',
      projection: projection,
      tileGrid: new WMTSTileGrid({
        origin: getTopLeft(projectionExtent),
        resolutions: resolutions,
        matrixIds: matrixIds,
      }),
      style: 'default',
      wrapX: true,
    }),
  }),
];
const map = new Map({
  layers: layers,
  target: 'map',
  view: new View({
    center: [1300000, 3300000],
    zoom: 2,
  }),
});
```

You will need to replace the `https://api.munichre.com/nathan/v2/` part with the URL of your proxy server, as web clients cannot handle the LRI API authentication (see Using a Proxy to Connect Your GIS Client Applications).

### *Leaflet*

To add the LRI layers with the WMTS format in your Leaflet application, use the `L.tileLayer` object.

The following snippet adds an OpenStreetMap base map and the `NathanRiverFlood` layer in the category `Hazards`:

```
const map = L.map('map').setView([0, 0], 2);

const tiles = L.tileLayer('https://tile.openstreetmap.org/{z}/{x}/{y}.png', {
       maxZoom: 19,
```

```
      attribution: '&copy; <a
href="http://www.openstreetmap.org/copyright">OpenStreetMap</a>'
}).addTo(map);

var wmtsLayer = L.tileLayer(
https://api.munichre.com/nathan/v2/maps/services/gwc/service/wmts?Service=WMTS&Request=GetTile&Version=1.0.0&layer=Hazards:NathanRiverFlood&style=default&tilematrixset=EPSG:900913&Format=image/png&TileMatrix=EPSG:900913:{z}&TileCol={x}&TileRow={y}', {
      attribution: 'Risk Management Partners, Munich Re',
      opacity: 0.6
}).addTo(map);
```

You will need to replace the `https://api.munichre.com/nathan/v2/` part with the URL of your proxy server, as web clients cannot handle the LRI API authentication (see Using a Proxy to Connect Your GIS Client Applications).

### *ArcGIS Maps SDK for JavaScript*

To add the LRI layers with the WMTS format, use the `WMTSLayer` object with the required parameters for `activeLayer` in your application.

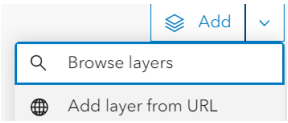
The following snippet adds an OpenStreetMap base map and the `NathanEarthquake` layer in the category `Hazards`:

```
    <script>
      require(["esri/Map", "esri/Basemap", "esri/views/MapView",
"esri/layers/WMTSLayer", "esri/layers/OpenStreetMapLayer"], (Map, Basemap, MapView,
WMTSLayer, OpenStreetMapLayer) => {

        const osmLayer = new OpenStreetMapLayer();

        const basemap = new Basemap( {
          baseLayers: [osmLayer],
        });

        const map = new Map({
          basemap: basemap
        });

        const view = new MapView({
        map: map,
        container: "viewDiv"
        });

        // add LRI map layer
        const lriLayer = new WMTSLayer({
          url: "https://api.munichre.com/nathan/v2/maps/services/gwc/service/wmts",
          activeLayer: {id: 'Hazards:NathanEarthquake', tileMatrixSetId:
"GoogleMapsCompatible", imageFormat: "image/png"},
          opacity: 0.6,
          copyright: 'Risk Management Partners, Munich Re'
        });
        map.add(lriLayer);
      });
    </script>
```

You will need to replace the `https://api.munichre.com/nathan/v2/` part with the URL of your proxy server, as web clients cannot handle the LRI API authentication (see Using a Proxy to Connect Your GIS Client Applications).

### *Portal for ArcGIS*

To add the LRI layers with the WMTS format in Portal for ArcGIS, select the option “Add layer from URL”:




Enter the URL to retrieve the WMTS capabilities:

```
https://api.munichre.com/nathan/v2/
maps/services/gwc/service/wmts?SERVICE=WMTS&request=GetCapabilities&version=1.0.0
```

Portal for ArcGIS will automatically detect the type WMTS (OGC). Click “Next” to retrieve the list of available services.

Select the required layer from the list and the projection (typically EPSG: 900913), then click “Add to map”.

You will need to replace the `https://api.munichre.com/nathan/v2/` part with the URL of your proxy server, as Portal for ArcGIS cannot handle the LRI API authentication (see Using a Proxy to Connect Your GIS Client Applications

# Using LRI Map Layers as Web Map Service (WMS)

## WMS Request Structure

### *Requesting Maps*

The URL structure for a WMS request to fetch a map image:

```
GET -
https://api.munichre.com/nathan/v2/
maps/services/{category}/wms?request=GetMap&service=WMS
&layers={service}&{WMS Parameters}
```

You need to replace these placeholders with your values:

- category
  For example, `General`, `Hazards`, `ClimateChange`)
- service
  For example, `PopulationDensity`, `NathanEarthquake`, `HeatStressIndexSSP126Y2050`)
- WMS parameters
  Please refer to the official WMS documentation for parameters like styles, bounding box, width, height, etc.

For information on how to get the available categories and services, please refer to Discovering the Available Services and Documentation.

Here is an example to request a tile for the `NathanRiverFlood` service in the `Hazards` category:

```
GET -
https://api.munichre.com/nathan/v2/
maps/services/Hazards/wms?REQUEST=GetMap&SERVICE=WMS&VERSION=1.3.0&FORMAT=image%2Fpn
g&STYLES=&TRANSPARENT=true&LAYERS=NathanRiverFlood&TILED=true&FORMAT_OPTIONS=dpi%3A1
35&WIDTH=384&HEIGHT=384&CRS=EPSG%3A3857&BBOX=2504688.542848654%2C6026906.8062295765%
2C2582960.0598126748%2C6105178.323193597
```

The request parameters are typically not added manually. Below are some examples of how to use the LRI map layers with client libraries and GIS platforms.

### *Service Metadata (GetCapabilities)*

You can also get the service metadata as a WMS GetCapabilities response with this request:

```
GET -
https://api.munichre.com/nathan/v2/
maps/services/{category}/wms?REQUEST=GetCapabilities&SERVICE=Wms&VERSION=1.3.0
```

This response is usable by clients that support WMS service metadata.

## Using the WMS Map Services in Your Applications

### *OpenLayers*

To add the LRI layers with the WMS format, use the `TileLayer` object with a `TileWMS` source in your OpenLayers application.

The following snippet adds an OpenStreetMap base map and the `NathanRiverFlood` layer in the category `Hazards`:

```
import Map from 'ol/Map.js';
import OSM from 'ol/source/OSM.js';
import TileLayer from 'ol/layer/Tile.js';
import TileWMS from 'ol/source/TileWMS.js';
import View from 'ol/View.js';

const layers = [
  new TileLayer({
    source: new OSM(),
  }),
  new TileLayer({
    opacity: 0.6,
    source: new TileWMS({
      attributions: 'Risk Management Partners, Munich Re',
      url: 'https://api.munichre.com/nathan/v2/maps/services/Hazards/wms',
      params: {'LAYERS': 'NathanRiverFlood', 'TILED': true},
      serverType: 'geoserver',
      // Countries have transparency, so do not fade tiles:
      transition: 0,
    }),
  }),
];
const map = new Map({
  layers: layers,
  target: 'map',
  view: new View({
```

```
    center: [1300000, 3300000],
    zoom: 2,
  }),
});
    zoom: 2,
  }),
});
```

You will need to replace the `https://api.munichre.com/nathan/v2/` part with the URL of your proxy server, as web clients cannot handle the LRI API authentication (see Using a Proxy to Connect Your GIS Client Applications).

### *Leaflet*

To add the LRI layers with the WMS format in your Leaflet application, use the `L.tileLayer` object.

The following snippet adds an OpenStreetMap base map and the `NathanRiverFlood` layer in the category `Hazards`:

```
const map = L.map('map').setView([0, 0], 2);

const tiles = L.tileLayer('https://tile.openstreetmap.org/{z}/{x}/{y}.png', {
       maxZoom: 19,
       attribution: '&copy; <a
href="http://www.openstreetmap.org/copyright">OpenStreetMap</a>'
}).addTo(map);

var wmsLayer = L.tileLayer.wms(
'https://api.munichre.com/nathan/v2/maps/services/Hazards/wms?', {
    layers: 'NathanRiverFlood',
    attribution: 'Risk Management Partners, Munich Re',
    format: 'image/png',
    opacity: 0.6
}).addTo(map);
```

You will need to replace the `https://api.munichre.com/nathan/v2/` part with the URL of your proxy server, as web clients cannot handle the LRI API authentication (see Using a Proxy to Connect Your GIS Client Applications).

### *ArcGIS Maps SDK for JavaScript*

**Please note:**
ArcGIS Maps SDK for JavaScript does not currently support tiled WMS. For best performance, we recommend using TMS or WMTS.

To add the LRI layers with the WMS format, use the `WMSLayer` object with the required parameters in your application.

The following snippet adds an OpenStreetMap base map and the `NathanEarthquake` layer in the category `Hazards`:

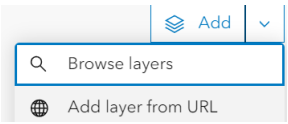
```
    <script>
      require(["esri/Map", "esri/Basemap", "esri/views/MapView",
"esri/layers/WMSLayer", "esri/layers/OpenStreetMapLayer"], (Map, Basemap, MapView,
WMSLayer, OpenStreetMapLayer) => {

        const osmLayer = new OpenStreetMapLayer();
```

```
        const basemap = new Basemap( {
          baseLayers: [osmLayer],
        });

        const map = new Map({
          basemap: basemap
        });

        const view = new MapView({
        map: map,
        container: "viewDiv"
        });

        // add LRI map layer
        const lriLayer = new WMSLayer({
          url: "https://api.munichre.com/nathan/v2/maps/services/Hazards/wms",
          sublayers: [
            {
              name: "NathanEarthquake",
            }
          ],
          imageFormat: "image/png",
          opacity: 0.6,
          copyright: 'Risk Management Partners, Munich Re'
        });
        lriLayer.load()

        map.add(lriLayer);
      });
    </script>
```

You will need to replace the `https://api.munichre.com/nathan/v2/` part with the URL of your proxy server, as web clients cannot handle the LRI API authentication (see Using a Proxy to Connect Your GIS Client Applications).

### *Portal for ArcGIS*

To add the LRI layers with the WMS format in Portal for ArcGIS, select the option “Add layer from URL”:




Enter the URL to retrieve the WMTS capabilities:

```
https://api.munichre.com/nathan/v2/
maps/services/Hazards/wms?REQUEST=GetCapabilities&SERVICE=Wms&VERSION=1.3.0
```

Portal for ArcGIS will automatically detect the type WMS (OGC). Click “Next” to retrieve the list of available services.

Select the required layer from the list, then click “Add to map”.

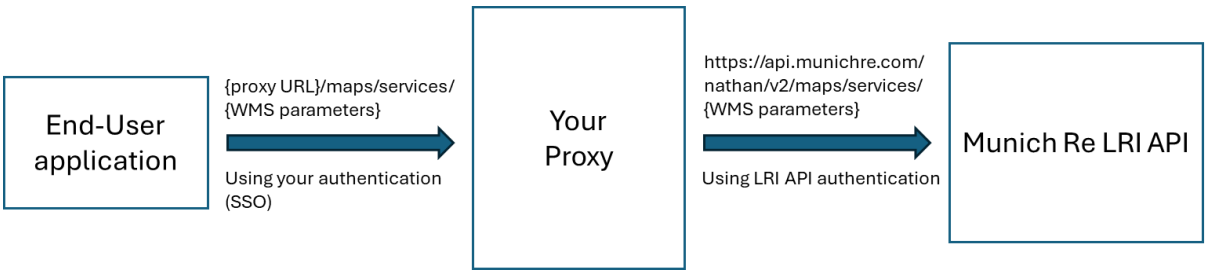
You will need to replace the `https://api.munichre.com/nathan/v2/` part with the URL of your proxy server, as Portal for ArcGIS cannot handle the LRI API authentication (see Using a Proxy to Connect Your GIS Client Applications

# Using a Proxy to Connect Your GIS Client Applications

The map services use the same authentication as all other API endpoints (JWT token + subscription key, see Authentication). This is not suitable for typical end-user applications:

- Client libraries or GIS systems usually can't handle the two authentication factors.
- Transferring the API credentials to end-user clients is considered a security risk.

It's therefore recommended to use a proxy server between your end-users applications and the LRI API:




Your clients connect to the proxy in a way that is supported by your organization (for example, SSO). The proxy then processes client requests in the following way:

1. The proxy creates a new request by replacing the proxy URL with the URL of the Munich Re LRI API. For example:
   From
   `{your proxy URL}/maps/services/Hazards/wms?REQUEST=GetMap…`
   To
   `https://api.munichre.com/nathan/v2/maps/services/Hazards/wms?REQUEST=GetMap…`
2. The proxy adds the authentication for the LRI API (JWT token and subscription key) to the headers of the new request.
3. The proxy sends the new request to the LRI API and then forwards the response to the client.

Example implementations for such reverse proxies are available for most modern development platforms. We can provide you with a sample implementation in ASP.NET Core.

# Discovering the Available Services and Documentation

The available map services depend on your licensed data edition. You have two possibilities to get detailed information about these services:

### *Structured data*

A GET request to the `configuration` endpoint returns a comprehensive JSON object containing detailed data about the available API content.

Please see the section API Configuration Data for in-depth descriptions of this data.

This configuration data is also ideal for rendering map legends for the LRI layers.

### *PDF data documentation*

Send a POST request to the `maps/documentation` endpoint to obtain the human-readable map service documentation in PDF format.

Currently, this POST request requires an empty JSON object as the request body:

```
POST - https://api.munichre.com/nathan/v2/maps/documentation
```

```
{}
```

This request body is a DocumentationReportOptions object, but the map service documentation currently does not support further parameters.

This map service documentation PDF is always generated on the fly using the latest data based on your licensed data edition. The data you get from the `configuration` endpoint is used to build the document.

Please note:
The category and service names (for example, “Hazards” → “NathanEarthquake”) always point to the latest version, so you don’t have to adjust your systems when we release new versions. Older versions will have a version identifier in the service name, separated by double underscores (for example, “NathanEarthquake__20160203”).

Munich RE

# API Versioning

## Overview

The Location Risk Intelligence API differentiates between two types of versioning:

- **Model Versioning**
  This relates to the version of the API itself (endpoint definitions, request and response model; see API Model Versioning)

- **Data (Content) Versioning**
  This relates to the version of the API content (enrichment services and map services; see API Data (Content) Versioning).

These two types of versioning are completely independent of each other.

## API Model Versioning

The current major version of the LRI API specification is v2. The major version is also part of the API URL:

```
https://api.munichre.com/nathan/v2
```

Minor versions are always backward-compatible and are not reflected in the API URL. Typical minor version changes are:

- Adding new functionality (new endpoint, new properties in the request and response models)
- Marking endpoints as obsolete (they will still be available and work as before)

Changes like these will be communicated through our LRI API mail distribution list and reflected in this technical API documentation. If you want to be added to the LRI API mail distribution list, please email us at location-risk-intelligence@munichre.com.

Please note:

- API v1 was decommissioned and is no longer available.
- A new version (v3) would be introduced if breaking changes were introduced to the API (for example, a mandatory new field, renamed endpoints with changed functionality, changes in the authentication, etc.).
- There are currently no plans for any breaking changes and a v3. Our goal is to provide a stable platform without the requirement to adjust to constant changes.
- If a new major version is introduced at some point in the future, the current v2 will continue to work as before for at least 12 months, giving you ample time to adapt your systems.

# API Data (Content) Versioning

The LRI API contains comprehensive data on natural hazards, climate change, financial risks, etc. The chapter on API Configuration Data describes how this data is organized in detail.

All this API content is versioned. New content is released multiple times yearly as it becomes available (there is no fixed release schedule). These content updates will be communicated through our LRI API mail distribution list and reflected in the API data documentation. If you want to be added to the LRI API mail distribution list, please email us at location-risk-intelligence@munichre.com.

The version number for the content is determined by the release date and has this format:

```
YYYYMMDD
```

Additionally, each version has a `VersionType` property with one of these values:

- Obsolete
  (This content is outdated)
- Previous
  (This content was the predecessor of the current version)
- Current
  (This is the latest version of the content)
- Preview
  (This is new content currently available as a preview)

For example, the service “NathanStormSurge” in the category “Hazards” has multiple versions:

- 20171220 = Obsolete
- 20220215 = Previous
- 20240401 = Current

When making enrichment requests to the API (location-info or area-line-info endpoints), you must also specify the version you want. We recommend always using the constant “Current” to get the latest data. Querying previous or obsolete data is not recommended.

Please note:

- New content will have the “Preview” type for a few weeks. You can use this phase to query the services that are currently in “Preview” to update your existing data to the new version.
- Querying services that are in preview does not count towards your contractual quota. All other version types (obsolete, previous, current) do, however, count towards your contractual quota, and calls will be invoiced.
- It is possible to mix different version types (e.g., „Current“ and “Preview”) in one request. Requests with mixed version types will always be counted towards your contractual quota.

Munich RE

# API Configuration Data (Resource configuration)

## General

When calling the API `configuration` endpoint, you receive a complex ApiConfiguration JSON object that describes all the API content that is available to you. This configuration data can be used to automate all kinds of processing tasks for the API results. A typical example would be visualizing raw API scoring results with proper hazard zone descriptions and legend colors.

At the highest level, the ApiConfiguration object has these properties:

- **LocationInfos**
  Describes all the API content available for location-based scoring (`location-info` endpoint).

- **AreaLineInfos**
  Describes all the API content available for scoring complex geometries like polygons or lines (`area-line-info` endpoint).

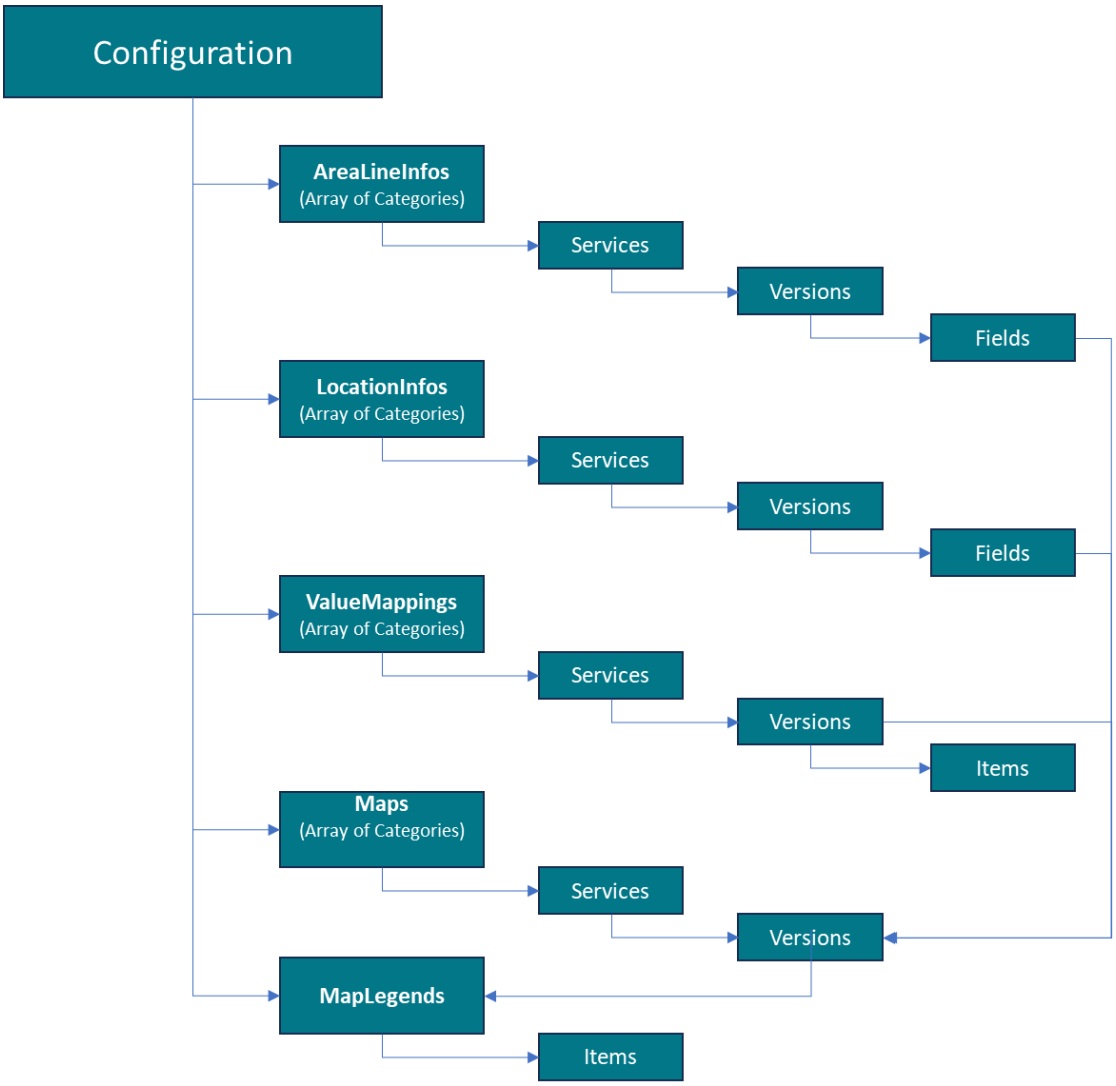
- **ValueMappings**
  Describes how API scoring results can be mapped to different outputs like traffic lights.

- **Maps**
  Describes the available map services in the API (`maps` endpoint).

- **MapLegends**
  This section describes the possible values for the scoring results and how to visualize them with the proper names, descriptions, and colors.

The actual available content depends on your licensed data edition.

Here is an overview of the most important API configuration data:




Most of the API content follows the same basic organizational structure:

- **Categories**
  The content is organized in categories (can be seen as folders) like Hazards, RiskScores, ClimateChange, General, etc...

- **Services**
  Inside the categories are individual services like NathanEarthquake, NathanTropicalCyclone, NathanRiverFlood

- **Versions**
  All API content is versioned as we continuously improve and update the data over time. Therefore, each service contains one or more versions.

- **Fields**
  → Only for enrichment services that return scores (LocationInfo and AreaLineInfo)
  The field configuration describes the individual result returned by the API (for example, an Earthquake HazardZone)

These data types are described in more detail below.

# Enrichment Configuration Data (LocationInfos and AreaLineInfos)

In the ApiConfiguration object, the available content for scoring (=enrichment) is defined in the properties LocationInfos and AreaLinesInfos:

- The **LocationInfos** property describes the content available for location enrichment (geospatial coordinates or an address identify locations)
- The **AreaLineInfos** property describes the content available to enrich complex geometries like polygons or lines.

Both properties (LocationInfos and AreaLinesInfos) use the same data structures to define the content. Therefore, we use the prefix "Enrichment" for these types.

## EnrichmentCategory

The properties LocationInfos and AreaLineInfos in the ApiConfiguration object are arrays of enrichment categories. These categories organize the API content into a folder-like structure.

Examples of categories: General, Hazards, RiskScores, ClimateChange

Properties:

| | |
|---|---|
| **Id** | Int64<br>A unique identifier for the category |
| **Name** | String<br>A unique name for the category |
| **Title** | String (nullable)<br>A display-friendly title for the category. If this value is null, use the Name property as display text. |
| **Services** | An array of EnrichmentService objects |

## EnrichmentService

A category contains an array of EnrichmentService objects. Each service defines one specific type of content that can be used to retrieve scores in the API.

Examples of services:

- General: PopulationDensity, NathanElevation
- Hazards: NathanEarthquake, NathanTropicalCyclone, NathanRiverFlood
- ClimateChange: TropicalCycloneCurrent, TropicalCycloneRCP45Y2030, TropicalCycloneRCP45Y2050

Properties:

| | |
|---|---|
| **Id** | Int64<br>A unique identifier for the service |
| **CategoryId** | Int64<br>A reference to the category of this service |
| **Name** | String<br>A unique name for the service inside its category |
| **Title** | String (nullable)<br>A display-friendly title for the service. If this value is null, use the Name property as display text. |
| **Versions** | An array of EnrichmentServiceVersion objects |

## EnrichmentServiceVersion

All API content is versioned. Therefore, a service contains an array of EnrichmentServiceVersion objects. Each version defines one specific version of the content released at some point.

Examples of versions: 20180101, 20220101, 20211205

The version naming follows a date-style naming convention: YYYYMMDD (Year, Month, Day)

Properties:

| | |
|---|---|
| **Id** | Int64<br>A unique identifier for the version |
| **ServiceId** | Int64<br>A reference to the service of this version |
| **Name** | String<br>A unique name for the version is included in its service. |
| **Title** | String (nullable)<br>A display-friendly title for the version. If this value is null, use the Name property as display text. |
| **VersionType** | Enum of ServiceVersionType<br>Defines the type of the content version in its lifecycle<br>Possible values:<br>Obsolete, Previous, Current, Preview |
| **Fields** | An array of EnrichmentServiceVersionField objects |

## EnrichmentServiceVersionField

An EnrichmentServiceVersion contains an array of EnrichmentServiceVersionField objects.
These fields define the actual scores returned by the API.

Examples of fields:

- General → PopulationDensity → **ClassificationPeoplePerSqm**

- Hazards → NathanEarthquake → **HazardZone**
- ClimateChange → TropicalCycloneRCP45Y2030 → **HazardZone**

Properties:

| | |
|---|---|
| **Id** | Int64<br>A unique identifier for the field |
| **ServiceVersionId** | Int64<br>A reference to the version of this field |
| **Name** | String<br>A unique name for the field inside its version. |
| **Title** | String (nullable)<br>A display-friendly title for the field. If this value is null, use the Name property as display text. |
| **DataType** | Enum of PropertyType<br>Defines data type of scoring results returned for this field<br>Possible values:<br>Long, Int, Short, Byte, Decimal, Double, Char, String, Date, DateAndTime, Boolean |
| **AttributeValue** | String (nullable)<br>This property can contain a common identifier for different fields of the same peril types.<br>Example: The value “TropicalCyclone” is used for all climate change services connected to the tropical cyclone hazard (current, all scenarios, all years) |
| **AttributeValueType** | String (nullable)<br>This property can contain a common identifier for different types of attribute values.<br>Examples: Acute, Chronic, Overall |
| **Scenario** | String (nullable)<br>This property contains the scenario name for climate change scores.<br>Examples: SSP2-/ RCP4.5, SSP5-/ RCP8.5 |
| **Year** | Int32 (nullable)<br>This property contains the year for climate change scores.<br>Examples: 2030, 2050, 2100 |
| **MitigatedFieldId** | Int64 (nullable)<br>If available, this property references the mitigated version of this score. For example, a flood zone that includes defense measures in its calculation. |
| **UnmitigatedFieldId** | Int64 (nullable)<br>If available, this property references the unmitigated version of this score. For example, a flood zone that does not include defense measures in its calculation. |

**MapServiceVersionId**
Int64 (nullable)
This property points to the map service definition for this field, if available. This reference can be used to show the map layer for this score in a custom mapping application. You can also retrieve the map legend for this field from the referenced map service definition. The map legend data is helpful for visualizing the raw scoring results in a user-friendly way with proper zone names and colors.

**ValueMappings**
Dictionary<String, Int64) (nullable)
This property defines which value mapping types are available for this field and references the ruleset that describes how the values must be mapped.

The property uses a Dictionary with this structure:
Key = the Id of the ValueMappingCategory object
Value = the Id of the ValueMappingServiceVersion object

Example for a traffic light mapping:

"valueMappings": { "1000000": 1100100 }

The key "1000000" references the ValueMappingCategory object for Traffic Light.
The value 1100100 references the ValueMappingServiceVersion object inside the Traffic Light mapping category. This ValueMappingServiceVersion object then contains an Items-array that defines how the API scoring results must be mapped to the Traffic Light scores.

**NumberOfDecimalPlaces**
Int32 (nullable)
If available, this property defines how many decimal places you can expect for the scoring results.

# Value Mappings

The value mapping configuration data defines how API scoring results can be interpreted differently. A typical example is mapping hazard zones to traffic light scores.

The ValueMappings property of the ApiConfiguration object contains all information on the available value mappings. The value mapping configuration follows the same basic organizational structure as other API content definitions, with categories, services, and versions.

## ValueMappingCategory

Each ValueMappingCategory defines one specific value mapping type, like traffic lights.
Properties:

**Id**
Int64
A unique identifier for the category

**Name**
String
A unique name for the category

| | |
|---|---|
| **Title** | String (nullable)<br>A display-friendly title for the category. If this value is null, use the Name property as display text. |
| **Services** | An array of ValueMappingService objects |

## ValueMappingService

A category contains an array of ValueMappingService objects. The services define the rulesets for mapping API scores to a value of the value mapping type (=ValueMappingCategory):

Example: NathanEarthquake = mapping an earthquake hazard zone value to a traffic light value.

Properties:

| | |
|---|---|
| **Id** | Int64<br>A unique identifier for the service |
| **CategoryId** | Int64<br>A reference to the category of this service |
| **Name** | String<br>A unique name for the service inside its category |
| **Title** | String (nullable)<br>A display-friendly title for the service. If this value is null, use the Name property as display text. |
| **Versions** | An array of ValueMappingServiceVersion objects |

## ValueMappingServiceVersion

All API content is versioned. Therefore, a service contains an array of ValueMappingServiceVersion objects. Each version defines one specific ruleset to map the scores returned by the API to the values of the value mapping type (=ValueMappingCategory).

Examples of versions: 20180101, 20220101, 20211205

The version naming follows a date-style naming convention: YYYYMMDD (Year, Month, Day)

Properties:

| | |
|---|---|
| **Id** | Int64<br>A unique identifier for the version<br>The EnrichmentServiceVersionField objects reference this identifier to define how the values have to be mapped for the ValueMappingCategory. |
| **ServiceId** | Int64<br>A reference to the service of this version |
| **Name** | String<br>A unique name for the version is included in its service. |

| | |
|---|---|
| **Title** | String (nullable)<br>A display-friendly title for the version. If this value is null, use the Name property as display text. |
| **VersionType** | Enum of ServiceVersionType<br>Defines the type of the content version in its lifecycle<br>Possible values:<br>Obsolete, Previous, Current, Preview |
| **MapServiceVersionId** | Int64<br>This property points to the map service definition for this value mapping version. The referenced map service definition is just a dummy in this case. But this map service then points to the required map legend for the current value mapping. The map legend data is helpful for visualizing the mapped values in a user-friendly way with proper score names and colors. |
| **Items** | An array of ValueMappingItem objects<br>These items define the actual mapping rules for the API scoring results. |

## ValueMappingItem

A ValueMappingServiceVersion contains an array of ValueMappingItem objects. These objects define how API scoring results are mapped to the target values of the current ValueMappingCategory (for example, traffic lights).

A score returned by the API always matches exactly one item in the array of ValueMappingItem objects.

There are three different types of mappings:

- Single value to mapped value
  → Property “Value”

- Value range to the mapped value
  → Property “ValueRange”

- Value array to the mapped value
  → Property “Values”

Properties:

| | |
|---|---|
| **Value** | Object (nullable)<br>This property is set when there is a 1:1 mapping of an API scoring result to the target value.<br><br>The type of the object depends on the EnrichmentServiceVersionField.DataType property.<br><br>The API scoring result must equal the Value property for a match. |

| | |
|---|---|
| **ValueRange** | ValueRange (nullable)<br>This property is set when a range of API scoring results (from-to) are mapped to the target value (see ValueRange object description for details). |
| **Values** | Array of objects (nullable)<br>This property is set when multiple API scoring results are mapped to the target value.<br>The type of the objects in the array depends on the EnrichmentServiceVersionField.DataType property.<br>The API scoring result must equal one of the Values array items for a match. |
| **MappedValue** | Object<br>The mapped value for the matching API score. |

### ValueRange

Properties:

| | |
|---|---|
| **From** | Double (nullable)<br>The lower boundary of the value range (included for the first item of the ValueMappingItem- Array, excluded for the other items).<br>If the value for the From property is null, then there is no lower boundary; only the upper limit has to match. |
| **To** | Double (nullable)<br>The upper boundary of the value range (always included)<br>If the value for the To-property is null, then there is no upper boundary, and only the lower limit has to match. |

Matching rule:

- If it is the first item in the ValueMappingItem array:
  From <= API score <= To

- For all other items in the ValueMappingItem array:
  From < API score <= To

## Map Service Configuration Data

The map configuration data defines the available map services in the API and links to the corresponding map legends. These map services can be used to visualize the Munich Re Location Risk Intelligence map layers in your custom mapping application. They are also partially relevant when you want to get the map legend data to visualize your API scoring results in a user-friendly way with proper zone names and colors.

The Maps property of the ApiConfiguration object contains all information on the available map services. The map service configuration follows the same basic organizational structure as other API content definitions, with categories, services, and versions.

## MapCategory

In the ApiConfiguration object, the property Maps is an array of map categories. These categories are used to organize the API content into a folder-like structure.

Examples of categories: Hazards, RiskScores, ClimateChange

Properties:

| | |
|---|---|
| **Id** | Int64<br>A unique identifier for the category |
| **Name** | String<br>A unique name for the category |
| **Title** | String (nullable)<br>A display-friendly title for the category. If this value is null, use the Name property as display text. |
| **Services** | An array of MapService objects |

## MapService

A category contains an array of MapService objects. Each object defines one specific type of map service that can be used to show Munich Re Location Risk Intelligence map layers in your custom mapping application.

Examples of services:

- Hazards: NathanEarthquake, NathanTropicalCyclone, NathanRiverFlood
- ClimateChange: TropicalCycloneCurrent, TropicalCycloneRCP45Y2030, TropicalCycloneRCP45Y2050

Properties:

| | |
|---|---|
| **Id** | Int64<br>A unique identifier for the service |
| **CategoryId** | Int64<br>A reference to the category of this service |
| **Name** | String<br>A unique name for the service inside its category |
| **Title** | String (nullable)<br>A display-friendly title for the service. If this value is null, use the Name property as display text. |
| **Versions** | An array of MapServiceVersion objects |

| | |
|---|---|
| **AttributeValue** | String (nullable)<br>This property can contain a common identifier for different services of the same peril types.<br>Example: The value “TropicalCyclone” is used for all climate change services connected to the tropical cyclone hazard (current, all scenarios, all years) |
| **Scenario** | String (nullable)<br>This property contains the scenario name for climate change scores.<br>Examples: SSP2-/ RCP4.5, SSP5-/ RCP8.5 |
| **Year** | Int32 (nullable)<br>This property contains the year for climate change scores.<br>Examples: 2030, 2050, 2100 |

## MapServiceVersion

All API content is versioned. Therefore, a service contains an array of MapServiceVersion objects. Each version defines one specific version of the content released at some point.

Examples of versions: 20180101, 20220101, 20211205

The version naming follows a date-style naming convention: YYYYMMDD (Year, Month, Day)

Properties:

| | |
|---|---|
| **Id** | Int64<br>A unique identifier for the version |
| **ServiceId** | Int64<br>A reference to the service of this version |
| **Name** | String<br>A unique name for the version is included in its service. |
| **Title** | String (nullable)<br>A display-friendly title for the version. If this value is null, use the Name property as display text. |
| **VersionType** | Enum of ServiceVersionType<br>Defines the type of the content version in its lifecycle<br>Possible values:<br>Obsolete, Previous, Current, Preview |
| **MapLegendId** | Int64<br>A reference to the map legend for this map service version. |
| **ServiceType** | Enum of MapServiceType<br>Defines the type of the map service endpoint in the API<br>Possible values:<br>Wms (OGC Web Map Service), Mvt (Mapbox Vector Tiles) |

| | |
|---|---|
| **ServiceUrl** | Obsolete, will be removed in November 2024<br>String (nullable)<br>Defines the endpoint of the map service in the API.<br><br>This value can be null because some map services are just dummy entries required to link to map legends. |
| **LayerIndexes** | Obsolete, will be removed in November 2024<br>Array of Int32 (nullable)<br>If the map service endpoint contains multiple layers, this property includes the layer indexes for the current map service version. |
| **Extent** | MapExtent object (nullable)<br>This property is set for map services that don't have worldwide coverage. You can use this value to automatically zoom to the extent of the map service when it is activated. |
| **Opacity** | Double<br>A default opacity for the map layer. |
| **ZIndex** | Int32<br>A default Z-index for the map layer. |
| **Attribution** | String (nullable)<br>If this property is set, the attribution must be visible when the map layer is displayed in the mapping application. |

### MapExtent

Properties:

| | |
|---|---|
| **XMin** | Double<br>The min coordinate along the X-axis (lower left corner) |
| **YMin** | Double<br>The min coordinate along the Y-axis (lower left corner). |
| **XMax** | Double<br>The max coordinate along the X-axis (upper right corner). |
| **YMax** | Double<br>The max coordinate along the Y-axis (upper right corner). |
| **SpatialReference** | Int32<br>The spatial reference identifier (SRID) for the specific coordinate system |

## Map Legends

Map legends define how the API content is visualized. They contain descriptions, zone names, and colors to display the raw API scoring results in a user-friendly way.

Example:

API scoring result for a location for the Tropical Cyclone hazard zone: **3**

Visualization with map legend data:


**How to find the corresponding map legend for API scores:**

The map legends are always referenced from MapServiceVersion objects. To get the map legend for a specific API score, you always take this path through the configuration objects:

EnrichmentServiceVersionField.MapServiceVersionId → References a MapServiceVersion object

MapServiceVersion.MapLegendId → References the MapLegend object.

(See also the overview picture at the beginning of the chapter.)

Example:

EnrichmentCategory: Hazards
EnrichmentService: NathanTropicalCyclone
EnrichmentServiceVersion: 20210415 (Current)
EnrichmentServiceVersionField: **HazardZone**

The MapServiceVersionId property of the HazardZone field points to the MapServiceVersion **1040200**.

Therefore, you get the MapServiceVersion object with the Id = **1040200** from the Maps configuration.

The MapLegendId property of the MapServiceVersion object points to the MapLegend **1040001**.

Therefore, you get the MapLegend object with the Id = 1040001. This gives you the map legend for the tropical cyclone hazard.

## MapLegend

In the ApiConfiguration object, the property MapLegends is an array of map legend objects.

Properties:

| | |
|---|---|
| **Id** | Int64<br>A unique identifier for the map legend |
| **Name** | String<br>A unique name for the map legend |
| **Title** | String (nullable)<br>A display-friendly title for the map legend. If this value is null, use the Name property as display text. |
| **Description** | String (nullable)<br>A short description of the score |

| | |
|---|---|
| **DetailedDescription** | String (nullable)<br>A longer and more detailed description of the score |
| **Unit** | String (nullable)<br>The unit of the score |
| **UnitDescription** | String (nullable)<br>A description of the unit of the score |
| **Source** | String (nullable)<br>The source of the underlying data of the score |
| **Items** | An array of MapLegendItem objects |

## MapLegendItem

A map legend item defines how specific scores can be visualized.

A score returned by the API always matches exactly one item in the array of MapLegendItem objects.

There are three different types of matching API scores to legend items:

- Single value to map legend item
  → Property “Value”

- Value range to map legend item
  → Property “ValueRange”

- Value array to map legend item
  → Property “Values”

Properties:

| | |
|---|---|
| **Value** | Object (nullable)<br>This property is set when exactly one API scoring result, like a hazard zone, matches the legend item.<br>The type of the object depends on the EnrichmentServiceVersionField.DataType property.<br>The API scoring result must equal the Value property for a match. |
| **ValueRange** | ValueRange (nullable)<br>This property is set when a range of API scoring results (from-to) are mapped to the target value (see ValueRange object description for details). |

| | |
|---|---|
| **Values** | Array of objects (nullable)<br>This property is set when multiple API scoring results are mapped to the target value.<br>The type of the objects in the array depends on the EnrichmentServiceVersionField.DataType property.<br>The API scoring result must equal one of the Values array items for a match. |
| **ShortText** | String<br>A short name for the score, for example, "Zone 3" |
| **LongText** | String<br>A long name for the score, for example, "Zone 3: 213 - 251 km/h" |
| **Symbol** | MapLegendSymbol<br>Defines the color and rendering for the map legend item |

## ValueRange

Properties:

| | |
|---|---|
| **From** | Double (nullable)<br>The lower boundary of the value range (included for the first item of the ValueMappingItem- Array, excluded for the other items).<br>If the value for the From property is null, then there is no lower boundary; only the upper limit has to match. |
| **To** | Double (nullable)<br>The upper boundary of the value range (always included)<br>If the value for the To-property is null, then there is no upper boundary, and only the lower limit has to match. |

Matching rule:

- If it is the first item in the ValueMappingItem array:
  From <= API score <= To

- For all other items in the ValueMappingItem array:
  From < API score <= To

## MapLegendSymbol

Properties:

| | |
|---|---|
| **SymbolType** | Enum of MapLegendSymbolType<br>Possible values:<br>SolidColor, Pattern<br>Currently, only SolidColor is used. |

| | |
|---|---|
| **Color** | String<br>The color as an HTML color code, for example, "#27A16E |

Munich RE

# Usage Statistics

## Overview

Retrieving enrichment results (scores) counts towards your contractual quota:

Each successfully processed location in a request increases the usage counter by 1 (location-info endpoint).

Each successfully processed area or line object in a request increases the usage counter by 10 (area-line-info endpoint).

Please note that the number of services in a request does not matter, only the number of locations. So, always add all required services to a request.

You then have multiple ways of getting usage statistics:

- Using the Interactive Dashboards in the LRI Web Application
- Using the Contract Information in the API
- Using Service Statistics in the API

## Using the Interactive Dashboards in the LRI Web Application

The simplest and most comfortable way to get detailed usage statistics is to access the interactive dashboards in the Location Risk Intelligence web application:

- Login to `https://risksuite.munichre.com/`.
- Click on the “Analytics Dashboard"

If you don’t have access to the LRI web application or you don’t see the analytics dashboard on the page, please contact your administrator.

Please note that the data in these dashboards is updated nightly, so today's usage will not be shown.

## Using the Contract Information in the API

If you want to get information about your current contract and the usage counter, you can send a GET request to the `/usage` endpoint.

```
POST - https://api.munichre.com/nathan/v2/usage
```

You will receive a response that looks like this:

```
{
   "contract":{
```

```
      "id":10002,
      "clientId":10000,
      "number":"CON123456",
      "startDate":"2024-01-01",
      "endDate":"2024-12-31",
      "renewalType":"Automatic",
      "quota":100000,
      "quotaTest":1000,
      "quotaLoadTest":0,
      "isDeleted":false
   },
   "usageProd":{
      "usageCounter":26301,
      "usagePercent":26.30,
      "limits":{
         "QuotaLimitInfo":90,
         "QuotaLimitWarning":100
      },
      "usageLeftTo100PercentQuota":73699
   },
   "usageTest":{
      "usageCounter":712,
      "usagePercent":71.20,
      "limits":{
         "QuotaLimitInfo":80,
         "QuotaLimitWarning":90,
         "QuotaLimitExceeded":100
      },
      "usageLeftTo100PercentQuota":288,
      "usageLeftUntilScoringIsBlocked":288
   }
}
```


The `/usage` endpoint has no additional request parameters and simply returns a `ContractUsageResponse` object for your current contract:

*ContractUsageResponse*

| | |
|---|---|
| **Contract** | A **Contract** object<br>This object contains the data for your current contract. |
| **UsageProd** | A ContractUsageResponseItem object<br>This object contains the usage data for requests made with your API account(s) for production use. |
| **UsageTest** | A ContractUsageResponseItem object<br>This object contains the usage data for requests made with your API account(s) for test use. |
| **UsageLoadTest** | A ContractUsageResponseItem object<br>This object contains the usage data for requests made with your API account(s) for load test use (only relevant when you have special load test accounts). |

*Contract*

| | |
|---|---|
| **Id** | Int64<br>This is an internal identifier for the contract. |
| **ClientId** | Int64<br>This is an internal identifier for your organization in the LRI platform. |
| **Number** | String<br>This is the number of the contract that Munich Re and your organization have signed. |
| **StartDate** | Date<br>This is the start date of your current contract. |
| **EndDate** | Date<br>This is the end date of your current contract. |
| **RenewalType** | ContractRenewalType enum<br>This enum can have the following values:<br>Automatic:<br>The contract will automatically be renewed for a year. The new contract period will start one day after the EndDate of your current contract.<br>Manual:<br>The contract will be renewed manually after an agreement between Munich Re and your organization has been reached. |
| **Quota** | Int32<br>This is the agreed contractual quota for production use. |
| **QuotaTest** | Int32<br>This is the agreed contractual quota for non-production use (development, testing, staging systems, etc.) |
| **QuotaLoadTest** | Int32<br>This is the agreed contractual quota for load tests (only relevant when you have special load test accounts). |
| **IsDeleted** | Bool<br>True, if the contract has been canceled. |

*ContractUsageResponseItem*

| | |
|---|---|
| **UsageCounter** | Int32<br>This is the current usage counter. |

| | |
|---|---|
| **UsagePercent** | Double<br><br>This is the current usage in percent based on your contractual quota (it can be more than 100%). |
| **Limits** | Dictionary with key = NotificationType and value = double (nullable)<br><br>The keys for this dictionary are NotificationType values:<br><br>QuotaLimitInfo<br>When this limit is hit, an info mail will be sent to your LRI administrators.<br><br>QuotaLimitWarning<br><br>When this limit is hit, a warning mail will be sent to your LRI administrators.<br><br>QuotaLimitExceeded<br><br>When this limit is hit, scoring will be blocked for the API account.<br><br>The corresponding values in the dictionary are the percentages at which the limits are triggered.<br><br>Blocking limits are usually only applied for API test accounts. Accounts for production use typically have no fixed upper limit, but info and warning mails will be triggered at 90% and 100%.<br><br>Contact us if you want to change these limits. |
| **UsageLeftTo100PercentQuota** | Int32<br><br>This is the number of locations left until you reach 100% percent of your contractual quota. This can be a negative number if you are above your quota. |
| **UsageLeftUntilScoringIsBlocked** | Int32 (nullable)<br><br>If a limit is set for the NotificationType QuotaLimitExceeded, this property will indicate how many usages are left until the API account gets blocked from scoring. |

## Using Service Statistics in the API

*Request*

You can get usage statistics for the requested enrichment services by sending a POST request to one of these endpoints:

For location scoring:

```
POST - https://api.munichre.com/nathan/v2/location-info/usage
```

For area and line scoring:

```
POST - https://api.munichre.com/nathan/v2/area-line-info/usage
```

In both cases, the request looks like this:

```
{
   "StartDate":"2024-01-01T00:00:00",
   "EndDate":"2024-06-30T00:00:00",
   "AggregatedBy":"Month"
}
```

The request body is a `LocationInfoUsageRequest` (location-info) or an `AreaLineInfoUsageRequest` (area-line-info) object with these properties:

| | |
|---|---|
| **StartDate** | DateTime (UTC)<br>This is a timestamp (UTC) that defines the start of the requested statistics. |
| **EndDate** | DateTime (UTC)<br>This is a timestamp (UTC) that defines the end of the requested statistics. |
| **AggregatedBy** | DateAggregation enum<br>This property defines how the statistics should be aggregated:<br>Day<br>Week<br>Month<br>Year |

*Response*

The API response then looks like this:

```
{
   "usedServices":[
      {
         "category":"Hazards",
         "service":"NathanEarthquake",
         "version":"20210415",
         "stats":[
            {
               "date":"2024-01-01T00:00:00Z",
               "locationCount":5907
            },
            {
               "date":"2024-02-01T00:00:00Z",
               "locationCount":1
            },
            {
               "date":"2024-03-01T00:00:00Z",
               "locationCount":204036
            },
            …
         ]
      },
      …
   ]
```

```
}
```

### *LocationInfoUsageResponse / AreaLineInfoUsageResponse*

At the top level, the request body is a `LocationInfoUsageResponse` (location-info) or an `AreaLineInfoUsageResponse` (area-line-info). This object contains only one property:

| | |
|---|---|
| **UsedServices** | Array of **LocationInfoUsageService or AreaLineInfoUsageService** objects<br>You will receive one item for each requested enrichment service in the selected timeframe. |

### *LocationInfoUsageService / AreaLineInfoUsageService*

| | |
|---|---|
| **Category** | String<br>The name of the enrichment category<br>(for example, "General", "Hazards", "ClimateChange") |
| **Service** | String<br>The name of the enrichment service<br>(for example, "PopulationDensity", "NathanEarthquake", "HeatStressIndexSSP126Y2050") |
| **Version** | String<br>The name of the enrichment version in the format YYYYMMDD |
| **Stats** | Array of **LocationInfoUsageEntry** or AreaLineInfoUsageEntry objects<br>You will get one item for each aggregated timeframe (day, week, month, year) |

### *LocationInfoUsageEntry*

| | |
|---|---|
| **Date** | Timestamp (UTC)<br>The start of the aggregated timeframe<br>(for example, the first of the month if the requested DateAggregation was set to Month) |
| **LocationCount** | Int32<br>The number of locations that have received enrichment results for the current service |

### *AreaLineInfoUsageEntry*

| | |
|---|---|
| **Date** | Timestamp (UTC)<br>The start of the aggregated timeframe<br>(for example, the first of the month if the requested DateAggregation was set to Month) |

| | |
|---|---|
| **GeometryCount** | Int32<br>The number of area or line objects that have received enrichment results for the current service |

# History

| Version | Date | Description |
|---|---|---|
| **2.0.1** | 30.04.2020 | Initial release of LRI API v2 |
| **2.0.2** | 01.07.2020 | Added service usage statistics endpoints |
| **2.1.0** | 09.03.2021 | Added ArcGIS map service endpoints |
| **2.2.1** | 23.03.2021 | Added AreaLineInfo endpoints and queued job processing |
| **2.2.2** | 22.04.2022 | Added documentation endpoints |
| **2.3.1** | 04.07.2024 | Major rework of the documentation, including:<br>- Complete rewrite of all chapters<br>- Added health check endpoint<br>- Added PDF reports for LocationInfo endpoints<br>- Added new TMS, WMTS, and WMS map services<br>- Removed descriptions for obsolete ArcGIS endpoints<br>- Added API configuration data endpoint (content description)<br>- Reworked usage statistics chapter |
| **2.3.2** | 30.07.2024 | Added optional TransactionId to LocationInfo and AreaLineInfo requests<br>Added map samples for ArcGIS Maps SDK for JavaScript<br>Added possibility to query multiple version types (for example, “Current” and “Previous”) in one enrichment request |
| **2.3.3** | 06.08.2024 | Added possibility to include mapped values (for example, traffic light mapping) in enrichment requests |