precisely

Global Routing SDK

User Guide

Version 2.0.12

Table of Contents

1 - Getting Started

This chapter lists all the prerequisites to get the GR SDK up and running.

In this section

What is Global Routing SDK?

The Global Routing SDK (GR SDK) is a routing toolkit that allows access to core routing capabilities through an in-process and REST API. This allows certain core functions to be integrated into any solution and enables you to develop and deploy routing desktop, mobile, or web applications. These applications can provide the ability to obtain driving or walking directions, calculate drive time and drive distance, and identify locations within a certain time or distance from a starting point.

GR SDK includes the following:

- The API, a core set of jar files
- Samples to use as reference
- WAR file for REST API
- JavaDoc providing descriptions of the Java API
- User guide, describing core features and examples (Java, REST, and SOAP API). To access the HTML-based user guide, open index.html page in your web browser located in the `dev\docs\User Guide\` folder.

System Requirements

To install GR SDK, your computer must meet the following system requirements.

Operating Systems

Windows

- Windows Server 2008 R2
- Windows Server 2012 R1
- Windows Server 2012 R2

Linux/Unix

- AIX 6.1
- AIX 7.1
- Solaris 10 (SPARC only)
- Solaris 11 (SPARC only)
- CentOS 5
- CentOS 6
- CentOS 7

- Oracle Linux 6
- Oracle Linux 7
- Red Hat Enterprise Linux 5
- Red Hat Enterprise Linux 6
- Red Hat Enterprise Linux 7
- SUSE 10
- SUSE 11
- Ubuntu 12.04 LTS
- Ubuntu 14.04 LTS
- Ubuntu 16.04 LTS

Web Server

Apache Tomcat version 7 and above

Disk Space

300MB of disk space for a complete installation (without data)

Java

JDK7 or higher

Note: You need to install Java 7 or above and set the environment variable `JAVA_HOME`.

Installing Data

GR SDK does not include routing data (only sample data is included with the sample applications). Applications built using GR SDK can use any routing data from Precisely. All data can be purchased separately.

Follow the install instructions that are included with your routing data to install the data on your system.

See the section **Configuring Routing Properties and Data Resources** on page 6 for instructions on how to configure your data with the GR SDK.

Installing the API

GR SDK is provided as a zip file (gra-*version*.zip) for Windows and as a TAR file (gra-*version*.tar) for non-Windows.

Once installed, you will have the following folder structure:

- **lib:** Contains the required jars and dependencies
- **javadoc:** Contains Javadoc files
- **docs:** Contains user documentation
- **legal:** Contains clauses for 3rd party library distribution.
- **service:** Contains REST WAR file
- **samples:** Contains sample applications illustrating use cases for the GR SDK, Global Geocoding API (GGA), and Spectrum Spatial SDK (LI-SDK).

Java API

To install the Java API:

1. Choose the relevant file for your system, extract it to any location on the file system.
2. Add the lib directory to the classpath.

REST API

To install the REST API:

1. Choose the relevant file for your system, extract it to any location on the file system.
2. Locate the GRA WAR file in the `install_dir\service` directory.
3. Deploy the WAR file in Apache Tomcat webapps directory: `install_dir\webapps`.
4. Configure the database. For Database configuration refer to **Configuring Routing Properties and Data Resources** on page 6.
5. Restart the Tomcat server.

Configuring Routing Properties and Data Resources

There are two configuration files involved in GRA:

- `routing.properties` - The routing properties file contains configuration parameters for the GR SDK instance.
- `dbList.json` - The data resource file contains the datasets and database configurations used by the GR SDK.

Note: For the Java API, the two configuration files can have any filename as long as they are referenced appropriately. For the instructions in this document, we are going to refer to these configuration files as `routing.properties`, and dbList.json.

Note: For the REST API, only the dbList.json configuration file can have any filename as long as it is referenced appropriately.

Routing Properties File

The routing properties file controls the routing instance default values and configuration. There are example `routing.properties` files (Java and REST) included with the GR SDK installation. These can be found in the following locations:

- Java - `install_dir\samples\resources`
- REST - after deploying the WAR file, `tomcat_install_dir>\webapps\webApp-context\WEB-INF\classes`

These files can be located anywhere on the file system. When creating routing applications, these property files a defined in the `GRAInstanceImpl` class to pass the routing parameters. For more information see, **Using Java API** on page 11.

The `routing.properties` file has the following parameters:

Parameter Name	Description	Required	Example
routeDefaultCoordSys	Specifies the default coordinate system.	No	EPSG:4326
routeTimeout	Specifies the amount of processing time (in milliseconds) before stopping a point to point routing request if processing is not complete. Default is 500000.	No	750000
multiPointTimeout	Specifies the amount of processing time (in milliseconds) before stopping a multipoint (point to point with intermediate points) routing request if processing is not complete. Default is 500000.	No	600000

Parameter Name	Description	Required	Example
matrixRouteTimeout	Specifies the amount of processing time (in milliseconds) before stopping a matrix routing request if processing is not complete. Default is 500000.	No	750000
allowFallback	Specifies whether to use major roads when other roads are unavailable. This is a Boolean parameter; if set as true, the engine uses major roads when other roads cannot be used.	No	False
shortProcessThreads	Specifies the number of threads to be used for point to point route requests and routing data requests.	No	16
longProcessThreads	Specifies the number of threads to be used for isochrone, matrix route, and multi-point route requests.	No	16
dynamicLoading	Specifies if the configuration changes are dynamically loaded. This is a Boolean type and if set to true, any changes in the properties and JSON configuration files will be loaded without restarting the application. Default is false.	No	False
dbConfigJSONFile	Specifies the file object with the path of the data resource file (dbList.json). This can be the absolute path to the directory on the file system or the relative path from the properties file to the data resource file.	Yes	E:\\GRA\\resources\\dbList.json

Note: All properties names in data routing.properties file are case sensitive.

Data Resource File

The data resource file contains all of the data resources that are accessible to the GR SDK, and defines the names of the databases used in the API. The file can be configured with multiple datasets and you can combine these datasets into the databases used by the API. There are example `dbList.json` files (Java and REST) included with the GR SDK installation. These can be found in the following locations:

- Java - See `install_dir\samples\resources`

- REST - After deploying the WAR file, see `tomcat_install_dir>\webapps\webApp-context\WEB-INF\classes`

  Note: To execute the REST API sample, refer to the `readme.txt` file in the `Install_dir\samples` directory. The `dbList.json` file inside the extracted WAR file must have a database with name `DC` configured with the path of the data in the `install_dir\resources\datasets` directory.

There are three main sections to the file.

Parameter Name	Description	Values
defaultDatabase	Specifies the name of the default database to be used. The database must be a valid database name defined in this file. For the Java API, this database is used if the DBResource method is not set in the API. For the REST API, this database is used if the *dbsource*.json is set to *default*.json. Although this parameter is optional, it is recommended to define the default database in case one is not specified in the code.	Define the name of the database to be used as the default. This must be one of the *name* values defined in the *databases* parameter.
datasets	Specifies the list of datasets to be made accessible to the GR SDK as databases.	Define the id element which is the unique identifier for the dataset. This is the value used to define the datasets element in the databases parameter. Define the paths element which is the location of the data on your file system. The path can be the absolute path to the directory on the file system or the relative path from the data resource file to the data. Path can also be a folder containing multiple routing datasets. In this case, each sub folder will be scanned for routing data and the routing data loaded for processing. **Note:** When specifying a folder, it should only contain routing data. During data loading, the Spatial server looks for the `metadata.json` file to locate the routing data. This file is also present for GeoCoding data. To avoid confusion (a failure during data loading) we recommend keeping only routing data under a folder.

Parameter Name	Description	Values
databases	Specifies the list of databases used by the API. You can combine datasets into a single database.	Define the name element which is the unique identifier for the database. This is also the name used to define the default database, or set the database being used in the API. Define the datasets element, which is the list of datasets that will make up the database. You can define one or more comma separated id(s) of the datasets.

Note: If the DBResource method is not set in the API, the defaultDatabase parameter in the data resource file is used for calculations.

Note: All properties in data resource file are case sensitive.

Example data resource file, with one database (US) consisting of two datasets:

```
{
 "defaultDatabase": "US",
 "datasets": [{
   "id": "US NE dataset",
   "paths":
["E:\\db\\ERM_Quarterly_MAR2015\\ERM-US\\2014.09\\driving\\northeast"]
  }, {
   "id": "US Central dataset",
   "paths":
["E:\\db\\ERM_Quarterly_MAR2015\\ERM-US\\2014.09\\driving\\central"]
  }
 ],
 "databases": [{
   "name": "US",
   "datasets": ["US NE dataset", "US Central dataset"]
  }
 ]
}
```

2 - Using Java API

The main entry point for the Global Routing API is described by the `IGRAInstance`, which is implemented by the `GRAInstanceImpl` class.

`GRAInstanceImpl` class can be instantiated by providing an object of either of the following two classes:

1. **File:** The **File** object created with the path to the `routing.properties` file.
2. **Options:** The **Options** class where you can specify all of the same parameters as the `File` object, however define them inline.

The `Options` class has the following mandatory parameter:

Field Name	Type	Description
dbConfigJSON	File	File object with the path of the database configuration JSON file

The `Options` class has the ability to specify the same optional parameters as defined in the `routing.properties` file. Descriptions for these properties can be found in **Configuring Routing Properties and Data Resources** on page 6.

You can instantiate the `GRAInstanceImpl` using one of the following ways:

```
IGRAInstance gra =
IGRAInstanceFactory.getInstance(PROPERTY_FILE_OBJECT);
```

Where `PROPERTY_FILE_OBJECT` is the `File` object containing all the necessary configuration and location of `dbList.json` file containing database resources information.

or

```
IGRAInstance gra =
IGRAInstanceFactory.getInstance(dbConfigJSON, options);
```

Where `dbConfigJSON` is the JSON file containing database resources information and options is the instance of `Options` class containing values for optional parameters.

```
IGRAInstance gra = IGRAInstanceFactory.getInstance(resourcesConfiguration,
options);
```

Where `resourcesConfiguration` is the instance of `DBResourcesConfiguration` class containing database resources information and options is the instance of `Options` class containing values for optional parameters.

> **Note:** The `IGRAInstance` is a heavy weight object and you should not create more than one instance in a single application. It is important to call the destroy method for the `IGRAInstance` once it is no longer required.

Once you get an instance of `IGRAInstance`, the routing capabilities methods can then be called.

In this section

GetTravelBoundary

Description

`GetTravelBoundary` determines a drive or walk time or distance boundary from a location. This feature obtains polygons corresponding to an isochrone or isodistance calculation. An isochrone is a polygon or set of points representing an area that can be traversed in a network from a starting point in a given amount of time. An isodistance is a polygon or set of points representing the area that is a certain distance from the starting point. The `GetTravelBoundary` operation (also known as an iso definition) takes a starting point, a unit (linear or time), one or more costs as input and returns the resulting travel boundary. Costs refer to the amount of time or distance to be used in calculating an iso. Multiple costs can also be given as input. In case of multiple costs, costs can also be provided as a comma delimited string.

Required Parameters

The `getTravelBoundary` method under the `IGRAInstance` interface accepts an object of `GetTravelBoundaryRequest` class. The `GetTravelBoundaryRequest` constructor must have the following as parameters:

Parameter	Type	Description
point	IPoint	Specifies the start location from where the travel boundary has to be calculated.
costs	Double[]	Specifies the cost distance or time, in the cost units specified (can be a decimal value). For example, if the unit specified is miles and you specify 10 in this parameter, the travel boundary will be calculated for how far you can travel in 10 miles. You can also specify multiple costs by specifying the values as a comma delimited string. It will return a separate travel boundary for every cost specified. If you specify multiple costs, every response will have cost and cost units associated with that response.

Parameter	Type	Description
costUnit	LinearUnit/TimeUnit	Specifies the type of metric used to calculate the travel boundary. Available distance values are: • LinearUnit.METER • LinearUnit.KILOMETER • LinearUnit.YARD • LinearUnit.FOOT/LinearUnit.FT • LinearUnit.MILE Available time values are: • TimeUnit.MINUTE • TimeUnit.MILLISECOND • TimeUnit.SECOND • TimeUnit.HOUR.

Optional Parameters

Optional parameters can be get or set in `GetTravelBoundaryRequest` object:

Parameter	Type	Description
dbResource	String	Specifies the name of the database resource used to run the request.
maxOffroadDistance	Double	Specifies the maximum distance to allow travel off the road network using the `maxOffroadDistanceUnit`. Examples of off-network roads include driveways and access roads. For example, if you specify a maximum off-road distance of 1 mile, the travel boundary will extend no further than one mile from the network road. If you specify a value of 0, the travel boundary will not extend beyond the road itself. Use the ambient speed options to specify the speed of travel along non-network roads.

Parameter	Type	Description
maxOffroadDistanceUnit	LinearUnit	Specifies the distance unit defining the `maxOffroadDistance`. You must also define `maxOffroadDistance` if you define this parameter. Available distance values are: • LinearUnit.METER • LinearUnit.KILOMETER • LinearUnit.YARD • LinearUnit.FOOT/LinearUnit.FT • LinearUnit.MILE.
destinationSrs	CoordSys	Specifies the coordinate system to return the travel boundary geometries. The default is the coordinate system of the data used (for example, epsg:4326).
majorRoads	Boolean	Specifies whether to include all roads in the calculation or just major roads. If you choose to include only major roads, performance will improve but accuracy may decrease. The default is true.
returnHoles	Boolean	Specifies whether you want to return holes, which are areas within the larger boundary that cannot be reached within the desired time or distance, based on the road network. The default is false.
returnIslands	Boolean	Specifies whether you want to return islands, which are small areas outside the main boundary that can be reached within the desired time or distance. The default is false.
simplificationFactor	Double	Specifies the percentage of the original points that should be returned or upon which the resulting complexity of geometries should be based. A number between 0.0 and 1.0 is accepted, exclusive of 0.0 but inclusive of 1.0. Complexity increases as the value increases, therefore 1.0 means the most complex. The default is 0.5.

Parameter	Type	Description
bandingStyle	BandingStyle	Specifies the style of banding to be used in the result. Banding styles are the types of multiple distance bands that can be displayed based on multiple costs. Banding styles can be returned in the following formats: • **BandingStyle.DONUT:** Each boundary is determined by subtracting out the next smallest boundary. This is the default method. • **BandingStyle.ENCOMPASSING:** Each boundary is determined independent of all others.
historicSpeedBucket	HistoricSpeedBucket	Specifies whether the routing calculation uses the historic traffic speeds. These speeds are based on different time-of-day buckets. The data must have historic traffic speeds included in order to use this feature. The data for each country/region has the same bucket definitions, where the speeds for these bucket values may vary. The options are: • **HistoricSpeedBucket.NONE:** This is the default value. It does not include Historic traffic data in calculation, instead an averaged speed value is used. • **HistoricSpeedBucket.AMPEAK:** Calculate routes with the peak AM speeds. The AMPeak time bucket is from 07:00 to 10:00 hrs. time of day. • **HistoricSpeedBucket.PMPEAK:** Calculate routes with the peak PM speeds. The PMPeak time bucket is from 16:00 to 19:00 hrs. time of day. • **HistoricSpeedBucket.OFFPEAK:** Calculate routes with the off peak (daytime) speeds. The OffPeak time bucket is from 10:00 to 16:00 hrs. time of day. • **HistoricSpeedBucket.NIGHT:** Calculate routes with the nighttime speeds. The Night time bucket is from 22:00 to 04:00 hrs. time of day.

Parameter	Type	Description
defaultAmbientSpeed	Double	Specifies the speed to travel when going off in a network road to find the travel boundary (for all road types). To control how off-network travel is used in the travel boundary calculation, you need to specify the speed of travel off the road network (the ambient speed). Ambient speed can affect the size and shape of the travel boundary polygon. In general, the faster the ambient speed, the larger the polygon. For example, if you were at a point with 5 minutes left, and if the ambient speed was 15 miles per hour, boundary points would be put at a distance of 1.25 miles. If the ambient speed was reduced to 10 miles per hour, boundary points would be put at a distance of 0.83 miles.
ambientSpeedUnit	VelocityUnit	Specifies the unit of measure to calculate the ambient speed. Available speed units are: • VelocityUnit.MPH (miles per hour) • VelocityUnit.KPH (kilometers per hour) • VelocityUnit.MTPS (meters per second) • VelocityUnit.MTPM (meters per minute)
ambientSpeeds	Map[RoadType, Double]	Specifies the ambient speed for off-network travel based on the road type. You must specify both the road type and the new speed for that road type. The speed is defined in the defined *ambientSpeedUnit*. Road types can be returned in all supported types. For a list of road type enumerations, see **Java API Road Type Enumeration** on page 132.
propagationFactor	Double	Specifies the percentage of the cost used to calculate the distance between the starting point and the isodistance. Propagation factor serves the same purpose for isodistances as ambient speed does for isochrones, that is, it controls how off-network travel is used in the travel boundary calculation. Propagation factor can affect the size and shape of the travel boundary polygon. In general, more the propagation factor value, the larger the polygon. This is similar to ambient speed, except that it applies to isodistances. The default value for this property is 0.16. If this property is not specified, then the calculation uses the value from the server setting. Valid values are between 0.0 and 1.0, both inclusive.

Parameter	Type	Description
propagationFactorOverride	Map[RoadType,Double]	Specifies the propagation factor to be used for an off-network travel based on the road type. You must specify both the road type and the propagation factor for that road type. This will override the default propagation factor value for the particular roadType. This option is applicable for isodistance only.
avoid	String	Specifies a comma-separated list of road types to be avoided during travel boundary calculation. This is a String parameter. When a road type is provided as the value of the parameter, the boundary excludes that type of roads in the calculation. For example, if tollRoad is provided as the parameter value, the calculated boundary will have no toll roads.

Code Example

Travel boundary with multiple costs

```
IGRAInstance gra = IGRAInstanceFactory.getInstance(PROPERTY_FILE_OBJECT);
IPoint point = new
Point(SpatialInfo.create(CoordSysConstants.longLatWGS84), new
DirectPosition(-73, 40));
GetTravelBoundaryRequest.Builder boundaryRequestBuilder = new
GetTravelBoundaryRequest.Builder(point, new double[]{5,10} ,
TimeUnit.MINUTE);
GetTravelBoundaryRequest boundaryRequest = boundaryRequestBuilder .
DBResource("US") .build();
GetTravelBoundaryResponse boundaryResponse =
gra.getTravelBoundary(boundaryRequest);
```

Note: DBResource("US") is the name of the database resource to use in the request. This parameter is not required if defaultDatabase is configured in JSON configuration file.

GetRoute

Description

`GetRoute` returns routing information for a set of two distinct points or multiple points. It takes a starting location and an ending location with optional intermediate points as input, and returns the route that is either the fastest or the shortest.

Required Parameters

The `getRoute` method under `IGRAInstance` interface, accepts an object of `GetRouteRequest` class. The `GetRouteRequest` constructor must have the following as parameters:

Parameter	Type	Description
startPoint	IPoint	Specifies the start location of the route.
endPoint	IPoint	Specifies the end location of the route.

Optional Parameters

Following optional parameters can be get or set in `GetRouteRequest` object:

Parameter	Type	Description
dbResource	String	Specifies the name of the database resource used to run the request.
intermediatePoints	List[IPoint]	Specifies the list of intermediate points to include along the route.
optimizeIntermediatePoints	Boolean	Specifies a processing parameter that indicates if the intermediate points should be optimized. The default is false. By default the intermediate points will be used in the calculation in the order specified. If set to true, the specified points will be re-ordered in an optimal manner during route computation.

Parameter	Type	Description
destinationSrs	CoordSys	Specifies the coordinate system to return the route and resulting geometries. The default is the coordinate system of the data used.
optimizeBy	OptimizeBy	Specifies the type of optimizing to use for the route. Valid values are `OptimizeBy.TIME` or `OptimizeBy.DISTANCE`. The default is `OptimizeBy.TIME` .
distanceUnit	LinearUnit	Specifies the units to return distance. The default is LinearUnit.METER. Available values are: • LinearUnit.METER • LinearUnit.KILOMETER • LinearUnit.YARD • LinearUnit.FOOT/LinearUnit.FT • LinearUnit.MILE
returnTime	Boolean	Specifies the route directions including the time it takes to follow a direction. The default is true.
timeUnit	TimeUnit	Specifies the units to return the time. The default is TimeUnit.MINUTE. Available values are: • TimeUnit.MINUTE • TimeUnit.MILLISECOND • TimeUnit.SECOND • TimeUnit.HOUR.
language	Language	Specifies the language the travel directions should be returned. The default is Language.ENGLISH. Directions can be returned in all supported languages. For a list of language enumerations, see **Java API Language Enumeration** on page 134.
majorRoads	Boolean	Specifies whether to include all roads in the calculation or just major roads. If you choose to include only major roads, performance will improve but accuracy may decrease. The default is false.
returnDirections	Boolean	Specifies whether to return turn-by-turn directions in route response. The default is false.

Parameter	Type	Description
returnSegmentGeometry	Boolean	Specifies whether to return the route geometry in route response. The default is false.
returnDirectionGeometry	Boolean	Specifies whether to return a separate geometry associated with the route instruction in route response. The default is false.
roadTypePreferences	Map[RoadType, RoadTypePriority]	Specifies the priority to give to different types of roads when determining the route. The road type is the format as described in the `ambientSpeeds` option in `GetTravelBoundary.road` type priority can be of the following format: • **RoadTypePriority.HIGH:** Prefer the road type over other road types. • **RoadTypePriority.MEDIUM:** Give this road type equal preference with other road types. If no preference is specified for a road type, the default is Medium. • **RoadTypePriority.LOW:** Prefer other road types over this road type. • **RoadTypePriority.AVOID:** Exclude the road type from routes if possible. It is not always possible to exclude a road type from the travel directions. Depending on the situation, the alternative to an avoided road type may be so poor that the software will choose a route that uses an avoided road type. Also, if the starting or ending point lies along a segment whose road type has been avoided, the software will still use that segment.

Parameter	Type	Description
historicSpeedBucket	HistoricSpeedBucket	Specifies whether the routing calculation uses the historic traffic speeds. These speeds are based on different time-of-day buckets. The data must have historic traffic speeds included in order to use this feature. The data for each country/region has the same bucket definitions, where the speeds for these bucket values may vary. The options are: • **HistoricSpeedBucket.NONE:** This is the default value. It does not include Historic traffic data in calculation, instead an averaged speed value is used. • **HistoricSpeedBucket.AMPEAK:** Calculate routes with the peak AM speeds. The AMPeak time bucket is from 07:00 to 10:00 hrs. time of day. • **HistoricSpeedBucket.PMPEAK:** Calculate routes with the peak PM speeds. The PMPeak time bucket is from 16:00 to 19:00 hrs. time of day. • **HistoricSpeedBucket.OFFPEAK:** Calculate routes with the off peak (daytime) speeds. The OffPeak time bucket is from 10:00 to 16:00 hrs. time of day. • **HistoricSpeedBucket.NIGHT:** Calculate routes with the nighttime speeds. The Night time bucket is from 22:00 to 04:00 hrs. time of day.
transientUpdates	List<Update>	See **Transient Updates** on page 39 for more details.
avoidTollRoads	Boolean	Specifies whether to calculate a route with or without a toll road. This is a Boolean type parameter. The default value is False. If you set the value of avoidTollRoad to True, the response contains a route without any toll roads. If the value of avoidTollRoad is set to False, the route includes toll roads.
localRoadsLoadFactor	String	Specifies the number of local roads that can be loaded into memory during route or matrix calculation. The number of roads that can load is directly proportional to the value selected in the parameter where 1 is the minimum and 3 is the maximum value. Valid values can be 1,2, or 3. The default is 1. See **Local Roads Load Factor** on page 126 for a detailed description and impact of the parameter on routing or matrix calculation. **Note:** The parameter does not accept decimal values.

Code Example

Simple Route with start and end points.

```
IGRAInstance gra = IGRAInstanceFactory.getInstance(PROPERTY_FILE_OBJECT);
IPoint startPoint = new
Point(SpatialInfo.create(CoordSysConstants.longLatWGS84), new
DirectPosition(-73, 40));
IPoint endPoint = new
Point(SpatialInfo.create(CoordSysConstants.longLatWGS84), new
DirectPosition(-73, 41));
GetRouteRequest.Builder routeRequestBuilder = new
GetRouteRequest.Builder(startPoint, endPoint);
GetRouteRequest routeRequest = routeRequestBuilder . DBResource("US")
.build();
GetRouteResponse routeResponse = gra.getRoute(routeRequest);
```

Note: DBResource("US") is the name of the database resource to use in the request. This parameter is not required if defaultDatabase is configured in JSON configuration file.

Commercial Vehicle Restrictions

Description

Commercial Vehicle Restrictions (CVR) are composed of directives to the routing engine that guides the behavior and attributes of commercial vehicles making trips along the route. Depending upon vehicle attributes provided (such as height, width, length, weight) and the commercial vehicle restriction attributes present in the road network, the decision is made whether to allow for routing a particular vehicle over a segment or not. If there is no commercial vehicle restriction attribute present in the road network, input restriction parameters will not affect the resultant route.

Optional Parameters

Following optional parameters can be get/set in `GetRouteRequest` object:

Parameter	Type	Description
looseningBarrierRestrictions	Boolean	Specifies whether the barriers will be ignored when determining the route. These restrictions are applicable when a commercial vehicle is prohibited from traversing a segment due to local ordinance or a commercial vehicle is allowed on the segment but only when it must (for example, last mile access, local delivery, and so on). Routes where a barrier has been removed will still have a higher route cost even if the route it shorter/faster than a route with no barrier.

Vehicle Attributes

These attributes specify the details of the vehicle on which the restrictions are applied. The attributes can be vehicle type, height, weight, length, or width when determining the route. Commercial vehicles are divided into different types ranging from short trailers to long triples. The Commercial Vehicle Restrictions attribution is organized on a per-vehicle type basis. This means it is entirely possible for a segment to be preferred for one vehicle type and the same segment have a restriction for another type of vehicle. Use the following optional parameters to determine the vehicle's properties:

Parameter	Type	Description
vehicleType	String	Choose either ALL or one of the types of vehicles: • STRAIGHT • SEMI_TRAILOR • STANDARD_DOUBLE • INTERMEDIATE_DOUBLE • LONG_DOUBLE • TRIPLE • OTHER_LONG_COMBINATION_VEHICLE
weight	Double	Specifies the maximum weight of a vehicle. Any vehicles over this value will be restricted when determining the route. The units of weight are: • kg • lb • mt • t

Parameter	Type	Description
height	Double	Specifies the maximum height of a vehicle. Any vehicles over this value will be restricted when determining the route. The units of height are: • ft • yd • mi • m • km
length	Double	Specifies the maximum length of a vehicle. Any vehicles over this value will be restricted when determining the route. The units of length are: • ft • yd • mi • m • km
width	Double	Specifies the maximum width of a vehicle. Any vehicles over this value will be restricted when determining the route. The units of width are: • ft • yd • mi • m • km

Note: You need to specify either weight/height or length/width along with its corresponding unit. Based on the attributes value you can see variation in time and distance value and in route cost also.

Code Example

Simple Route with start and end points.

```
IPoint startPoint = new
Point(SpatialInfo.create(CoordSysConstants.longLatWGS84), new
DirectPosition(-77.088217, 38.937072));
IPoint endPoint = new
Point(SpatialInfo.create(CoordSysConstants.longLatWGS84), new
DirectPosition(-76.971986, 38.889853));
GetRouteRequest.Builder routeRequestBuilder = new
GetRouteRequest.Builder(startPoint, endPoint);
VehicleAttributes attributes = new VehicleAttributes(VehicleType.ALL);
attributes.setLooseningBarrierRestrictions(false);
Length height = new Length(50.0, LinearUnit.METER);
attributes.setHeight(height);

Length length = new Length(50.0, LinearUnit.METER);
attributes.setLength(length);

Length width = new Length(50.0, LinearUnit.METER);
attributes.setWidth(width);

Weight wgt = new Weight(20000, WeightUnit.POUND);
attributes.setWeight(wgt);

routeRequestBuilder.vehicleAttr(attributes);

GetRouteRequest routeRequest = routeRequestBuilder
 .returnSegmentGeometry(true)
 .DBResource("DC")
 .build();
GetRouteResponse routeResponse = m_gra.getRoute(routeRequest);
```

GetRouteCostMatrix

Description

`GetRouteCostMatrix` calculates the travel time and distances between an array of start and end locations and returns the route that is either the fastest or the shortest. The result determines the total time and distance of the individual routes (the route costs). For example if you input four start points and four end points, a total of 16 routes will be calculated.

Required Parameters

The `getRouteCostMatrix` method under `IGRAInstance` interface, accepts an object of `GetRouteCostMatrixRequest` class. The `GetRouteCostMatrixRequest` constructor must have the following as parameters:

Parameter	Type	Description
startPoint	IPoint	Specifies the start location of the route.
endPoint	IPoint	Specifies the end location of the route.

Optional Parameters

Following optional parameters can be get or set in `GetRouteRequest` object:

Parameter	Type	Description
dbResource	String	Specifies the name of the database resource used to run the request.
destinationSrs	CoordSys	Specifies the coordinate system to return the route and resulting geometries. The default is the coordinate system of the data used.
optimizeBy	OptimizeBy	Specifies the type of optimizing to be used to calculate the the route. The default value is OptimizeBy.TIME. Available values are: • OptimizeBy.TIME • OptimizeBy.DISTANCE.
distanceUnit	LinearUnit	Specifies the units to return distance. The default is LinearUnit.METER. Available values are: • LinearUnit.METER • LinearUnit.KILOMETER • LinearUnit.YARD • LinearUnit.FOOT/LinearUnit.FT • LinearUnit.MILE
timeUnit	TimeUnit	Specifies the units to return time. The default is TimeUnit.MINUTE. Available values are: • TimeUnit.MINUTE • TimeUnit.MILLISECOND • TimeUnit.SECOND • TimeUnit.HOUR

Parameter	Type	Description
majorRoads	Boolean	Specifies whether to include all roads in the calculation or just major roads. If you choose to include only major roads, performance will improve but accuracy may decrease. The default is false.
returnOptimalRoutesOnly	Boolean	Specifies whether to return only the optimized route for each start point/end point combination. The default is true. The optimized route is either the fastest route or the shortest distance, depending on the `optimizeBy` parameter.
roadTypePreferences	Map[RoadType, RoadTypePriority]	Specifies the priority to give to different types of roads when determining the route. The road type is the format as described in the `ambientSpeeds` option in `GetTravelBoundary`. road type priority can be of the following format: • **RoadTypePriority.HIGH:** Prefer the road type over other road types • **RoadTypePriority.MEDIUM:** Give this road type equal preference with other road types. If no preference is specified for a road type, the default is Medium. • **RoadTypePriority.LOW:** Prefer other road types over this road type. • **RoadTypePriority.AVOID:** Exclude the road type from routes if possible. It is not always possible to exclude a road type from the travel directions. Depending on the situation, the alternative to an avoided road type may be so poor that the software will choose a route that uses an avoided road type. Also, if the starting or ending point lies along a segment whose road type has been avoided, the software will still use that segment.

Parameter	Type	Description
historicSpeedBucket	HistoricSpeedBucket	Specifies whether the routing calculation uses the historic traffic speeds. These speeds are based on different time-of-day buckets. The data must have historic traffic speeds included in order to use this feature. The data for each country/region has the same bucket definitions, where the speeds for these bucket values may vary. The options are: • **HistoricSpeedBucket.NONE:** The default value. Historic traffic data is not used in the calculation. Instead an averaged speed value is used. • **HistoricSpeedBucket.AMPEAK:** Calculate routes with the peak AM speeds. The AMPeak time bucket is from 07:00 to 10:00 hrs. time of day. • **HistoricSpeedBucket.PMPEAK:** Calculate routes with the peak PM speeds. The PMPeak time bucket is from 16:00 to 19:00 hrs. time of day. • **HistoricSpeedBucket.OFFPEAK:** Calculate routes with the off peak (daytime) speeds. The OffPeak time bucket is from 10:00 to 16:00 hrs. time of day. • **HistoricSpeedBucket.NIGHT:** Calculate routes with the nighttime speeds. The Night time bucket is from 22:00 to 04:00 hrs. time of day.
transientUpdates	List<Update>	See **Transient Updates** on page 39 for more details.
avoidTollRoads	Boolean	Specifies whether to calculate a route with or without a toll road. This is a Boolean type parameter. The default value is False. If you set the value of avoidTollRoad to True, the response contains a route without any toll roads. If the value of avoidTollRoad is set to False, the route includes toll roads.
localRoadsLoadFactor	String	Specifies the number of local roads that can be loaded into memory during route or matrix calculation. The number of roads that can load is directly proportional to the value selected in the parameter where 1 is the minimum and 3 is the maximum value. Valid values can be 1,2, or 3. The default is 1. See **Local Roads Load Factor** on page 126 for a detailed description and impact of the parameter on routing or matrix calculation. **Note:** The parameter does not accept decimal values.

Code Example

Simple Route with start and end points.

```
IGRAInstance gra = IGRAInstanceFactory.getInstance(PROPERTY_FILE_OBJECT);
IPoint startPoint1 = new
Point(SpatialInfo.create(CoordSysConstants.longLatWGS84), new
DirectPosition(-73.994062, 40.76312));
IPoint startPoint2 = new
Point(SpatialInfo.create(CoordSysConstants.longLatWGS84), new
DirectPosition(-73.985124, 40.765067));
IPoint endPoint1 = new
Point(SpatialInfo.create(CoordSysConstants.longLatWGS84), new
DirectPosition(-74.005972, 40.714269));
IPoint endPoint2 = new
Point(SpatialInfo.create(CoordSysConstants.longLatWGS84), new
DirectPosition(-73.9978, 40.7509));
List<IPoint> startPoints = new ArrayList<>();
startPoints.add(startPoint1);
startPoints.add(startPoint2);
List<IPoint> endPoints = new ArrayList<>();
endPoints.add(endPoint1);
endPoints.add(endPoint2);
GetRouteCostMatrixRequest.Builder costMatrixRequestBuilder = new
GetRouteCostMatrixRequest.Builder(startPoints, endPoints);
GetRouteCostMatrixRequest matrixRequest =
costMatrixRequestBuilder.DBResource("US").build();
GetRouteCostMatrixResponse matrixResponse =
gra.getRouteCostMatrix(matrixRequest);
```

Note: DBResource("US") is the name of the database resource to use in the request. This parameter is not required if defaultDatabase is configured in the JSON configuration file.

Commercial Vehicle Restrictions

Description

Commercial Vehicle Restrictions (CVR) are composed of directives to the routing engine that guides the behavior and attributes of commercial vehicles making trips along the route. Depending upon vehicle attributes provided (such as height, width, length, weight) and the commercial vehicle restriction attributes present in the road network, decision is made whether to allow to route a particular vehicle over a segment or not. If there is no commercial vehicle restriction attribute present in road network, input restriction parameters will have no effect in the resultant route.

Optional Parameters

Following optional parameters can be get or set in `GetRouteRequest` object:

Parameter	Type	Description
looseningBarrierRestrictions	Boolean	Specifies that barriers will be ignored when determining the route. These restrictions are most often when a commercial vehicle is prohibited from traversing a segment due to local ordinance or a commercial vehicle is allowed on the segment but only when it must (for example, last mile access, local delivery, and so on). Routes where a barrier has been removed will still have a higher route cost even if the route it shorter/faster than a route with no barrier

Vehicle Attributes

These attributes specify the details of the vehicle on which the restrictions are applied. The attributes can be vehicle type, height, weight, length, or width when determining the route. Commercial vehicles are divided into different types ranging from short trailers to long triples. The Commercial Vehicle Restrictions attribution is organized on a per-vehicle type basis. This means it is entirely possible for a segment to be preferred for one vehicle type and the same segment have a restriction for another type of vehicle. Use the following types of vehicle information. Use the following optional parameters to determine the vehicle's properties:

Parameter	Type	Description
vehicleType	String	Choose either ALL or one of the types of vehicles: • STRAIGHT • SEMI_TRAILOR • STANDARD_DOUBLE • INTERMEDIATE_DOUBLE • LONG_DOUBLE • TRIPLE • OTHER_LONG_COMBINATION_VEHICLE.

Parameter	Type	Description
weight	double	Specifies the maximum weight of a vehicle. Any vehicles over this value will be restricted when determining the route. The units of weight are: • kg • lb • mt • t
height	double	Specifies the maximum height of a vehicle. Any vehicles over this value will be restricted when determining the route. The units of height are: • ft • yd • mi • m • km
length	double	Specifies the maximum length of a vehicle. Any vehicles over this value will be restricted when determining the route. The units of length are: • ft • yd • mi • m • km
width	double	Specifies the maximum width of a vehicle. Any vehicles over this value will be restricted when determining the route. The units of width are: • ft • yd • mi • m • km

Note: You need to specify either weight/height or length/width along with its corresponding unit. Based on the attributes value you can see variation in time and distance value and in route cost also.

Code Example

Simple Route with start and end points.

```
IPoint startPoint1 = new
Point(SpatialInfo.create(CoordSysConstants.longLatWGS84), new
DirectPosition(-73.994062, 40.76312));
IPoint startPoint2 = new
Point(SpatialInfo.create(CoordSysConstants.longLatWGS84), new
DirectPosition(-73.985124, 40.765067));
IPoint endPoint1 = new
Point(SpatialInfo.create(CoordSysConstants.longLatWGS84), new
DirectPosition(-74.005972, 40.714269));
IPoint endPoint2 = new
Point(SpatialInfo.create(CoordSysConstants.longLatWGS84), new
DirectPosition(-73.9978, 40.7509));
List < IPoint > startPoints = new ArrayList <  > ();
startPoints.add(startPoint1);
startPoints.add(startPoint2);
List < IPoint > endPoints = new ArrayList <  > ();
endPoints.add(endPoint1);
endPoints.add(endPoint2);
boolean returnOptimalRoutes = true;
GetRouteCostMatrixRequest.Builder costMatrixRequestBuilder = new
GetRouteCostMatrixRequest.Builder(startPoints,
endPoints).returnOptimalRoutesOnly(returnOptimalRoutes).DBResource("US");
VehicleAttributes attributes = new VehicleAttributes(VehicleType.ALL);
attributes.setLooseningBarrierRestrictions(false);
Length height = new Length(50.0, LinearUnit.METER);
attributes.setHeight(height);
Length length = new Length(50.0, LinearUnit.METER);
attributes.setLength(length);
Length width = new Length(50.0, LinearUnit.METER);
attributes.setWidth(width);
Weight wgt = new Weight(20000, WeightUnit.POUND);
attributes.setWeight(wgt);
costMatrixRequestBuilder.vehicleAttr(attributes);
GetRouteCostMatrixRequest matrixRequest =
costMatrixRequestBuilder.build();
matrixResponse = m_gra.getRouteCostMatrix(matrixRequest);
```

Persistent Updates

Description

The Persistent Update service allows a user to override aspects of the network. The overrides can be done on a per-road type, at a specific point, or at a specific segment. The persistent update is valid only for a specific data source and may not be valid after a data update.

Using persistent updates to make these types of modifications, you have the ability to:

- Exclude a point
- Exclude a segment
- Set the speed of a point, segment, or a road type
- Change (increase or decrease) the speed of a point, segment, or road type by a value
- Change (increase or decrease) the speed of a point, segment, or road type by a percentage

Types of Persistent Updates

The following is a description of the persistent update types:

PointUpdate

Point updates are changes applied to a corresponding point (X, Y). For a particular point, you can exclude the point, set the speed of the point, or change (increase or decrease) the speed of the point by a value or percentage. Use one of the following types of updates:

PointUpdate Type	Description
percentage	Specifies a speed update where you can replace the speed of the point by specifying a percentage to increase (positive value) or decrease (negative value) the speed.
speed	Specifies a speed update where you can define the new speed of the point by specifying the new velocity. For speed updates, the velocity can have one of the following values: • kph (kilometers per hour) • mph (miles per hour) • mtps (meters per second) • mtpm (meters per minute)

PointUpdate Type	Description
speedAdjustment	Specifies a speed update where you can define a change in the speed of the point by specifying the change in the value. The speed velocity can be increased (positive value) or decreased (negative value). For speed updates, the velocity can have one of the following values: • kph (kilometers per hour) • mph (miles per hour) • mtps (meters per second) • mtpm (meters per minute)
exclude	Specifies a value to exclude the specified point from the route calculation. To exclude a point you need to specify the point and exclude parameter defined as True. This is a Boolean update type and valid values are true and false.

SegmentUpdate

Segment updates are changes applied to a corresponding segment ID. For a particular segment, you can exclude the segment, set the speed of the segment, or change (increase or decrease) the speed of the segment by a value or percentage. Use one of the following types of updates:

SegmentUpdate Type	Description
percentage	Specifies a speed update where you can define an increase in the speed of the segmentID by specifying a percentage to increase (positive value) or decrease (negative value) the speed.
speed	Specifies a speed update where you define a new speed of the segmentID by specifying the new velocity. For speed updates, the velocity can have one of the following values: • kph (kilometers per hour) • mph (miles per hour) • mtps meters per second • mtpm (meters per minute)
speedAdjustment	This is a speed update where you can define a change in the speed of the segmentID by specifying the change in velocity. Speed values can be increased (positive value) or decreased (negative value). For speed updates, the velocity can have one of the following values: • kph (kilometers per hour) • mph (miles per hour) • mtps (meters per second) • mtpm (meters per minute)

SegmentUpdate Type	Description
exclude	Specifies a value to exclude the specified segmentID from the route calculation. To exclude a segmentID you need to specify the segmentID and exclude parameter defined as True. This is a Boolean update type and valid values are true and false.
roadType	Specifies a value to change the value of the road type for the segment for the route calculation. This is a String update type. See the roadType options listed below.

RoadTypeUpdate

Road type updates are changes applied to a corresponding road type. For a particular road type, you can set the speed of the roadtype or change (increase or decrease) the speed of the road type by a value or percentage. Use one of the following types of updates:

RoadTypeUpdate Type	Description
percentage	Specifies a speed update where you can define a change in the speed of the road type by specifying a percentage to increase (positive value) or decrease (negative value) the speed.
speed	Specifies a speed update where you can define a new speed of the road type by specifying the new velocity. For speed updates, the velocity unit can have one of the following values: • kph (kilometers per hour) • mph (miles per hour) • mtps (meters per second) • mtpm (meters per minute)
speedAdjustment	Specifies a speed update where you can define a change in the speed of the road type by specifying the change in velocity. Speed values can be increased (positive value) or decreased (negative value). For speed updates, the velocity can have one of the following values: • kph (kilometers per hour) • mph (miles per hour) • mtps (meters per second) • mtpm (meters per minute)

The roadType can be one of the following:

- access way
- back road
- connector
- ferry

- footpath
- limited access dense urban
- limited access rural
- limited access suburban
- limited access urban
- local road dense urban
- local road rural
- local road suburban
- local road urban
- major local road dense urban
- major local road rural
- major local road suburban
- major local road urban
- major road dense urban
- major road rural
- major road suburban
- major road urban
- minor local road dense Urban
- minor local road rural
- minor local road suburban
- minor local road urban
- normal road dense urban
- normal road rural
- normal road rural
- normal road urban
- primary highway dense urban
- primary highway rural
- primary highway suburban
- primary highway urban
- ramp dense urban
- ramp limited access
- ramp major road
- ramp primary highway
- ramp rural
- ramp secondary highway
- ramp urban
- ramp suburban
- secondary highway dense urban
- secondary highway rural
- secondary highway suburban
- secondary highway urban

Code Examples

Point Update

```
IPoint point = new
Point(SpatialInfo.create(CoordSysConstants.longLatWGS84), new
DirectPosition(-73, 40));
Update pointUpdate = PointUpdate.buildPointSpeedUpdate(point, new
Velocity(20, VelocityUnit.MPH));
List < Update > updates = new ArrayList <  > ();
updates.add(pointUpdate);
IGRAInstance m_gra =
IGRAInstanceFactory.getInstance(PROPERTY_FILE_OBJECT);
PersistentUpdateRequest updateReq = new PersistentUpdateRequest(updates);
PersistentUpdateResponse response =
m_gra.createPersistentUpdates(updateReq);
```

Segment Update

```
String SEGMENT_ID = "76ec2634:d0bfa";
Update segmentUpdate = SegmentUpdate.buildSegmentExclude(SEGMENT_ID,
true);
List < Update > updates = new ArrayList <  > ();
updates.add(segmentUpdate);
IGRAInstance m_gra =
IGRAInstanceFactory.getInstance(PROPERTY_FILE_OBJECT);
PersistentUpdateRequest updateReq = new PersistentUpdateRequest(updates);
```

RoadType Update

```
Update roadTypeUpdate =
RoadTypeUpdate.buildRoadTypeSpeedUpdate(RoadType.ACCESS_WAY, new
Velocity(50, VelocityUnit.MPH));
List < Update > updates = new ArrayList <  > ();
updates.add(roadTypeUpdate);
IGRAInstance m_gra =
IGRAInstanceFactory.getInstance(PROPERTY_FILE_OBJECT);
PersistentUpdateRequest updateReq = new PersistentUpdateRequest(updates);
PersistentUpdateResponse response =
m_gra.createPersistentUpdates(updateReq);
```

Note: If you need to save persistent update at any location other than temp folder (which is the default location), then you can add the persistent update file path against `routingUpdateFilePath` property in `resources/routing.properties` file. To do this, you need to declare following property in the `resources/routing.properties` file:

- routingUpdateFilePath=<ROUTING_UPDATE_FILE_PATH>

Transient Updates

Description

It allows you to set transient updates (point, segment, road type updates) for each request. For instance, you can request a server attempt to avoid all of the major road types. Each request can contain one or more updates. For speed updates, positive speed value is a speed increase and negative speed value is a speed decrease.

The Transient Updates service allows a user to override aspects of the network. The overrides can be done on a per-road type, at a specific point or at a specific segment. The Transient Update is valid only for a specific data source and may not be valid after a data update.

Using the Transient Updates, you can perform the following:

- Exclude a point
- Exclude a segment
- Set the speed of a point, segment, or road type
- Change (increase or decrease) the speed of a point, segment, or road type by a value
- Change (increase or decrease) the speed of a point, segment, or road type by a percentage

The following is a description of the transient update types:

PointUpdate

Point updates are changes applied to a corresponding IPoint. For a particular point, you can: exclude the point, set the speed of the point or change (increase or decrease) the speed of the point by a value or percentage. Use one of the following types of updates:

PointUpdate Type	Description
percentage	Specifies a speed update where you can define a change in the speed of the point by specifying a percentage to increase (positive value) or decrease (negative value) the speed.
speed	Specifies a speed update where you can define the new speed of the point by specifying the velocity unit and new velocity (positive value). For speed updates, the velocity unit can have one of the following values: • kph (kilometers per hour) • mph (miles per hour) • mtps (meters per second) • mtpm (meters per minute)

PointUpdate Type	Description
speedAdjustment	Specifies a speed update where you can define a change in the speed of the point by specifying the change in velocity (unit and value). Speed values can be increased (positive value) or decreased (negative value). For speed updates, the velocity unit can have one of the following values: • kph (kilometers per hour) • mph (miles per hour) • mtps (meters per second) • mtpm (meters per minute)
exclude	Specifies a value to exclude the specified point from the route calculation. To exclude a point you need to specify the point and exclude parameter defined as True. This is a Boolean update type and valid values are true and false.

SegmentUpdate

Segment updates are changes applied to a corresponding segment ID. For a particular segment, you can: exclude the segment, set the speed of the segment, change (increase or decrease) the speed of the segment by a value or percentage, or change the road type of the segment. Use one of the following types of updates:

SegmentUpdate Type	Description
percentage	Specifies a speed update where you can define a change in the speed of the segmentID by specifying a percentage to increase (positive value) or decrease (negative value) the speed.
speed	Specifies a speed update where you can define the new speed of the segmentID by specifying the velocity unit and new velocity (positive value). For speed updates, the velocity unit can have one of the following values: • kph (kilometers per hour) • mph (miles per hour) • mtps (meters per second) • mtpm (meters per minute)

SegmentUpdate Type	Description
speedAdjustment	Specifies a speed update where you can define a change in the speed of the segmentID by specifying the change in velocity (unit and value). Speed values can be increased (positive value) or decreased (negative value). For speed updates, the velocity unit can have one of the following values: • kph (kilometers per hour) • mph (miles per hour) • mtps (meters per second) • mtpm (meters per minute)
exclude	Specifies a value to exclude the specified segmentID from the route calculation. To exclude a segmentID you need to specify the segmentID and exclude parameter defined as True. This is a Boolean update type and valid values are true and false.
roadType	Specifies a value to change the value of the road type for the segment for the route calculation. This is a String update type. See list of available `roadType` below.

The `roadType` can be one of the following:

- LIMITED_ACCESS_DENSE_URBAN
- LIMITED_ACCESS_URBAN
- LIMITED_ACCESS_SUBURBAN
- LIMITED_ACCESS_RURAL
- PRIMARY_HIGHWAY_DENSE_URBAN
- PRIMARY_HIGHWAY_URBAN
- PRIMARY_HIGHWAY_SUBURBAN
- PRIMARY_HIGHWAY_RURAL
- SECONDARY_HIGHWAY_DENSE_URBAN
- SECONDARY_HIGHWAY_URBAN
- SECONDARY_HIGHWAY_SUBURBAN
- SECONDARY_HIGHWAY_RURAL
- MAJOR_ROAD_DENSE_URBAN
- MAJOR_ROAD_URBAN
- MAJOR_ROAD_SUBURBAN
- MAJOR_ROAD_RURAL
- NORMAL_ROAD_DENSE_URBAN
- NORMAL_ROAD_URBAN
- NORMAL_ROAD_SUBURBAN
- NORMAL_ROAD_RURAL
- MAJOR_LOCAL_ROAD_DENSE_URBAN
- MAJOR_LOCAL_ROAD_URBAN

- MAJOR_LOCAL_ROAD_SUBURBAN
- MAJOR_LOCAL_ROAD_RURAL
- LOCAL_ROAD_DENSE_URBAN
- LOCAL_ROAD_URBAN
- LOCAL_ROAD_SUBURBAN
- LOCAL_ROAD_RURAL
- MINOR_LOCAL_ROAD_DENSE_URBAN
- MINOR_LOCAL_ROAD_URBAN
- MINOR_LOCAL_ROAD_SUBURBAN
- MINOR_LOCAL_ROAD_RURAL
- RAMP_DENSE_URBAN
- RAMP_URBAN
- RAMP_SUBURBAN
- RAMP_RURAL
- RAMP_LIMITED_ACCESS
- RAMP_PRIMARY_HIGHWAY
- RAMP_SECONDARY_HIGHWAY
- RAMP_MAJOR_ROAD
- FOOTPATH
- FERRY
- BACK_ROAD
- ACCESS_WAY
- CONNECTOR

RoadTypeUpdate

Road type updates are changes applied to a corresponding road type. For a particular road type, you can: set the speed of the roadtype, or change (increase or decrease) the speed of the road type by a value or percentage. Use one of the following types of updates:

RoadTypeUpdate Type	Description
percentage	Specifies a speed update where you can define a change in the speed of the road type by specifying a percentage to increase (positive value) or decrease (negative value) the speed.

RoadTypeUpdate Type	Description
speed	Specifies a speed update where you can define the new speed of the road type by specifying the velocity unit and new velocity. For speed updates, the velocity unit can have one of the following values: • kph (kilometers per hour) • mph (miles per hour) • mtps (meters per second) • mtpm (meters per minute)
speedAdjustment	Specifies a speed update where you can define a change in the speed of the road type by specifying the change in velocity (unit and value). Speed values can be increased (positive value) or decreased (negative value). For speed updates, the velocity unit can have one of the following values: • kph (kilometers per hour) • mph (miles per hour) • mtps (meters per second) • mtpm (meters per minute)

GetRoute with Point Update

```
PointUpdate update = PointUpdate.buildPointExclude(new
Point(m_spatialInfo, m_tustartPosition), true);
List<Update> updates = new ArrayList<>();
updates.add(update);
GetRouteRequest.Builder routeRequestBuilder = new
GetRouteRequest.Builder(new Point(m_spatialInfo, m_tustartPosition),
new Point(m_spatialInfo, m_tuendPosition)).updates(updates);
GetRouteResponse routeResponse =
m_gra.getRoute(routeRequestBuilder.build());
```

GetRouteCostMatrix request with Segment Updates

```
 SegmentUpdate update =
SegmentUpdate.buildSegmentSpeedAdjustment("76ec2634:14f2b3", new
Velocity(-11.0, VelocityUnit.MPH));
 List<Update> updates = new ArrayList<>();
 updates.add(update);
 GetRouteCostMatrixRequest.Builder costMatrixRequestBuilder = new
GetRouteCostMatrixRequest.Builder(startPoints,
endPoints).updates(updates);
 GetRouteCostMatrixResponse routeCostMatrixResponse =
m_gra.getRouteCostMatrix(costMatrixRequestBuilder.build());
```

GetRouteCostMatrix request with RoadType Updates

```
RoadTypeUpdate update = RoadTypeUpdate.buildRoadTypeSpeedPercentageUpdate
(RoadType.NORMAL_ROAD_DENSE_URBAN, 5);
List<Update> updates = new ArrayList<>();
updates.add(update);
GetRouteCostMatrixRequest.Builder costMatrixRequestBuilder = new
GetRouteCostMatrixRequest.Builder(startPoints,
endPoints).updates(updates);
GetRouteCostMatrixResponse routeCostMatrixTUResponse =
m_gra.getRouteCostMatrix(costMatrixRequestBuilder.build());
```

GetSegmentData

The `GetSegmentData` service returns segment information for a point or segment ID. When a point is specified, the closest route segments are returned. When a segment ID is specified, the route data for that specified route segment is returned.

Required Parameters

The `getSegmentData` method under the `IGRAInstance` interface, accepts an object of `GetSegmentDataRequest` class. The builder class in `GetSegmentDataRequest` class has two constructors: one accepts a IPoint and the other accepts a segmentID.

Parameter	Type	Description
point	IPoint	Specifies the location for which the segment data is required.
segmentID	String	Specifies the segment ID for which the segment data is required.

Optional Parameters

Following optional parameters can be get or set in the GetSegmentDataRequest builder object:

Parameter	Type	Description
dbResource	String	Specifies the name of the database resource used to run the request.
destinationSrs	CoordSys	Specifies the coordinate system to return the segment data and resulting geometry. The default value is the coordinate system of the data used.
distanceUnit	LinearUnit	Specifies the distance unit to be returned. The default value is LinearUnit.METER. The available values are: • LinearUnit.METER • LinearUnit.KILOMETER • LinearUnit.YARD • LinearUnit.FOOT/LinearUnit.FT • LinearUnit.MILE
timeUnit	TimeUnit	Specifies the time-unit to be returned The default value is TimeUnit.MINUTE. The available values are: • TimeUnit.MINUTE • TimeUnit.MILLISECOND • TimeUnit.SECOND • TimeUnit.HOUR
velocityUnit	VelocityUnit	Specifies the velocity to be returned. The default is VelocityUnit.MPH. The available values are: • VelocityUnit.MPH • VelocityUnit.KPH • VelocityUnit.MTPS • VelocityUnit. MTPM

Parameter	Type	Description
angularUnit	AngularUnit	Specifies the time-units to be returned. The default is AngularUnit.DEGREE. The available values are: • AngularUnit.DEGREE • AngularUnit. MINUTE • AngularUnit. SECOND • AngularUnit. RADIAN • AngularUnit. GRADIAN
segmentGeometryStyle	SegmentGeometryStyle	Specifies the style of geometry to be returned. The default value is SegmentGeometryStyle.ALL. The available values are: • SegmentGeometryStyle.ALL • SegmentGeometryStyle.END • SegmentGeometryStyle.NONE

Example Code

```
IGRAInstance gra = IGRAInstanceFactory.getInstance(PROPERTY_FILE_OBJECT);
IPoint point = new Point(SpatialInfo.create(CoordSysConstants.longLatWGS84),
 new DirectPosition(-73, 40));
GetSegmentDataRequest.Builder builder = new
GetSegmentDataRequest.Builder(point);
GetSegmentDataRequest request = builder.dbResource("US").build();
GetSegmentDataResponse response = gra.getSegmentData(request);
```

Note: DBResource("US") is the name of the database resource to use in the request. This parameter is not required if the defaultDatabase is configured in the JSON configuration file.

3 - Using REST API

The GRA REST API consists of HTTP GET or POST requests for `GetTravelBoundary`, `GetRoute`, and `GetRouteCostMatrix` operations.

In this section

GetTravelBoundary

Description

`GetTravelBoundary` determines a drive or walk time or distance of a boundary from a location. This feature obtains polygons corresponding to an isochrone or isodistance calculation. An isochrone is a polygon or set of points representing an area that can be traversed in a network from a starting point in a given amount of time. An isodistance is a polygon or set of points representing the area that is at a certain distance from the starting point. The `GetTravelBoundary` operation (also known as an iso definition) takes a starting point, a unit (linear or time), one or more costs as input and returns the resulting travel boundary. Costs refer to the amount of time or distance to use in calculating an iso. Multiple costs can also be given as input. In case of multiple costs, costs can also be provided as a comma delimited string.

Note: Response from REST service is not in JSON format.

HTTP GET URL Format

The following format is used for HTTP GET requests. HTTP GET is used for all travel boundaries where no additional JSON payload is required (ambient speed changes).

```
HTTP GET
/webApp-context/services/databases/dbsource.json?q=travelBoundary&query_parameters
```

Where: *dbsource* is the name of the database that contains the data to use for the route. Use the database name specified in the database resource file (dbList.json file). *webApp-context* is the endpoint to your web application or service. If you are using the default database specified in the database resource file, use `default.json` in the REST URL for *dbsource*.

HTTP POST URL Format

The following format is used for HTTP POST requests:

```
HTTP POST:
/webApp-context/services/databases/dbsource.json?q=travelBoundary&query_parameters
POST BODY: Content-Type:application/json {Route Data}
```

Route Data is the POST JSON body (`Content-Type: application/json`) for the additional route information to be used in the calculation containing ambient speeds for road types. For information and examples on these options, see **GetTravelBoundary HTTP POST Options** on page 55.

Note: The response from REST request is in JSON format and the geometries will be of GEOJSON format.

Query Parameters

This operation takes these query parameters.

Parameter	Type	Required	Description
point	String	Yes	Specifies the start location from where to calculate the travel boundary in the format: x,y,coordSys. For example, -74.2,40.8,epsg:4326
costs	Double	Yes	Specifies the cost distance or time, in the cost units specified (can be a decimal value). For example, if the unit specified is miles and you specify 10 in this parameter, the travel boundary will be calculated for how far you can travel 10 miles. You can also specify multiple costs by specifying the values as a comma delimited string. It will return a separate travel boundary for every cost specified. If you specify multiple costs, every response will have cost and cost units associated with that response.
costUnit	String	Yes	specifies the type of metric used to calculate the travel boundary. Available distance values are: m(meter), km(kilometer), yd(yard), ft(foot), mi(mile). Available time values are: min(minute), msec(millisecond), s(second), h(hour).
maxOffroadDistance	Double	No	Specifies the maximum distance to allow travel off the road network using the `maxOffroadDistanceUnit`. Examples of off-network roads include driveways and access roads. For example, if you specify a maximum off road distance of 1 mile the travel boundary will extend no further than one mile from the network road. If you specify a value of 0 the travel boundary will not extend beyond the road itself. Use the ambient speed options to specify the speed of travel along non-network roads.

Parameter	Type	Required	Description
maxOffroadDistanceUnit	String	No	Specifies the distance unit defining the `maxOffroadDistance`. You must also define `maxOffroadDistance` if you define this parameter. Available distance values are: • m (meter) • km (kilometer) • yd (yard) • ft (foot) • mi (mile)
destinationSrs	String	No	Specifies the coordinate system to return the travel boundary geometries. The default is the coordinate system of the data used (for example, epsg:4326).
majorRoads	Boolean	No	Specifies whether to include all roads in the calculation or just major roads. If you choose to include only major roads, performance will improve but accuracy may decrease. The default is true.
returnHoles	Boolean	No	Specifies whether you want to return holes, which are areas within the larger boundary that cannot be reached within the desired time or distance, based on the road network. The default is false.
returnIslands	Boolean	No	Specifies whether you want to return islands, which are small areas outside the main boundary that can be reached within the desired time or distance. The default is false.
simplificationFactor	Integer	No	Specifies what percentage of the original points should be returned or upon which the resulting complexity of geometries should be based. A number between 0.0 and 1.0 is accepted, exclusive of 0.0 but inclusive of 1.0. Complexity increases as the value increases, therefore 1.0 means the most complex. The default is 0.5.

Parameter	Type	Required	Description
bandingStyle	String	No	Specifies the style of banding to be used in the result. Banding styles are the types of multiple distance bands that can be displayed based on multiple costs. Banding styles can be returned in the following formats: • **Donut:** Each boundary is determined by subtracting out the next smallest boundary. This is the default method. • **Encompassing:** Each boundary is determined independent of all others.
historicTrafficTimeBucket	String	No	Specifies whether the routing calculation uses the historic traffic speeds. These speeds are based on different time-of-day buckets. The data must have historic traffic speeds included in order to use this feature. The data for each country/region has the same bucket definitions, where the speeds for these bucket values may vary. The options are: • **None:** The default value. Historic traffic data is not used in the calculation. Instead an averaged speed value is used. • **AMPeak:** Calculate routes with the peak AM speeds. The AMPeak time bucket is from 07:00 to 10:00 hrs. time of day. • **PMPeak:** Calculate routes with the peak PM speeds. The PMPeak time bucket is from 16:00 to 19:00 hrs. time of day. • **OffPeak:** Calculate routes with the off peak (daytime) speeds. The OffPeak time bucket is from 10:00 to 16:00 hrs. time of day. • **Night:** Calculate routes with the nighttime speeds. The Night time bucket is from 22:00 to 04:00 hrs. time of day.

Parameter	Type	Required	Description
defaultAmbientSpeed	String	No	Specifies the speed to travel when going off a network road to find the travel boundary (for all road types). To control how off-network travel is used in the travel boundary calculation, you need to specify the speed of travel off the road network (the ambient speed). Ambient speed can affect the size and shape of the travel boundary polygon. In general, the faster the ambient speed, the larger the polygon. For example, if you were at a point with 5 minutes left, and if the ambient speed were 15 miles per hour, boundary points would be put at a distance of 1.25 miles. If the ambient speed were reduced to 10 miles per hour, boundary points would be put at a distance of 0.83 miles. **Note:** The default `defaultAmbientSpeed` is **15**. This parameter can also be specified in the POST body. If the same parameter is set in both GET and POST, then the value in GET is considered.
ambientSpeedUnit	String	No	Specifies the unit of measure to use to calculate the ambient speed. Available speed units are: • MPH (miles per hour) • KPH (kilometers per hour) • MTPS (meters per second) • MTPM (meters per minute) **Note:** The default `ambientSpeedUnit` is **MPH**. This parameter can also be specified in the POST body.If the same parameter is set in both GET and POST, thenthe value in GET is considered.

Parameter	Type	Required	Description
propagationFactor	String	No	Specifies the percentage of the cost used to calculate the distance between the starting point and the isodistance (for all road types). Propagation factor serves the same purpose for isodistances as ambient speed does for isochrones, that is, it controls how off-network travel is used in the travel boundary calculation. Propagation factor can affect the size and shape of the travel boundary polygon. In general, more the propagation factor value, the larger the polygon. It applies to isodistances. If this property is not specified, then the calculation uses the server setting. Valid values are between 0.0 and 1.0, both inclusive. **Note:** The default propagationFactor is **0.16**. This parameter can also be specified in the POST body.If the same parameter is set in both GET and POST, then the value in POST is considered.
version	String	No	Specifies the version of the `GetTravelBoundary` REST service. Valid values are *1* and *2*. The default value for version is *1*.

Examples

Travel boundary with a single cost.

```
http://server:port/webApp-context/services/databases/usroutedatabase.json?
q = travelBoundary & point = -77.092609, 38.871256, epsg: 4326 &
costs = 5 & costUnit = m
```

Response

```
{
 "travelBoundary": {
  “costs”: [{
    "cost": 5,
    "costUnit": "m",
    "geometry": {
     "type": "MultiPolygon",
     "crs": {
      "type": "name",
      "properties": {
       "name": "epsg: 4326"
      }
     },
     "coordinates": [
      [
       [
        ...
       ]
      ]
     ]
    }
   }
  ]
 }
}
```

Travel boundary with multiple costs.

```
http:
//server:port/webApp-context/services/databases/usroutedatabase.json?
q = travelBoundary & point = -77.092609, 38.871256, epsg: 4326 &
costs = 2, 5 & costUnit = m
```

Response

```
{
 "travelBoundary": {
  "costs": [{
    "cost": 2,
    "costUnit": "m",
   "geometry": "{" type ":" MultiPolygon "," crs ":{" type ":" name ","
 properties ":{" name ":" epsg: 4326 "}}," coordinates ":[[[...]]]}"
   }, {
    "cost": 5,
    "costUnit": "m",
   "geometry": "{" type ":" MultiPolygon "," crs ":{" type ":" name ","
 properties ":{" name ":" epsg: 4326 "}}," coordinates ":[[[...]]]}"
   }
  ]
 }
}
```

Version specific error response

When you enter an invalid parameter value (for example, points falling outside of the boundaries) in a request, the error response you get depends on the version entered by you. When the version is 1, you get value and error whereas when the version is 2, the response only contains the error.

- Request when version is *1*:

```
http://server:port/webApp-context/services/databases/usroutedatabase.json?q=travelBoundary
&costs=5&costUnit=min&point=-14.321600,60.662859,epsg:4326&version=1
```

- Response:

```
{
  "value": "Point outside boundaries: (-14.3216,60.662859,0)",
  "errors": [
    {
      "errorCode": 5008,
     "userMessage": "Point outside boundaries: (-14.3216,60.662859,0)"

    }
  ]
}
```

- Request when version is *2*:

```
http://server:port/webApp-context/services/databases/usroutedatabase.json?q=travelBoundary
&costs=5&costUnit=min&point=-14.321600,60.662859,epsg:4326&version=2
```

- Response:

```
{
  "errors": [
    {
      "errorCode": 5008,
     "userMessage": "Point outside boundaries: (-14.3216,60.662859,0)"

    }
  ]
}
```

GetTravelBoundary HTTP POST Options

HTTP POST URL Format

In addition to the regular HTTP GET parameters, you can add a HTTP POST payload option to your request that specifies ambient speed changes for road types. The content type must be set to application/json. The following format is used for HTTP POST requests:

```
HTTP POST:
/webApp-context/services/databases/dbsource.json?q=travelBoundary&query_parameters
POST BODY: Content-Type:application/json {Route Data}
```

Route Data is the POST json body (`Content-Type: application/json`) for the additional route information to be used in the calculation containing ambient speeds for road types.

Ambient Speeds

It allows you to set ambient speed updates for each request. An ambient speed is a change to the speed in the normal data to travel off a network road when finding the travel boundary. Examples of off-network travel include driveways and access roads. The following is a description of the ambient speed parameters:

Parameter	Description
DefaultAmbientSpeed	Specifies the speed to travel when going off a network road to find the travel boundary (for all road types). To control how off-network travel is used in the travel boundary calculation, you need to specify the speed of travel off the road network (the ambient speed). Ambient speed can affect the size and shape of the travel boundary polygon. In general, the faster the ambient speed, the larger the polygon. For example, if you were at a point with 5 minutes left, and if the ambient speed were 15 miles per hour, boundary points would be put at a distance of 1.25 miles. If the ambient speed were reduced to 10 miles per hour, boundary points would be put at a distance of 0.83 miles. **Note:** Default `DefaultAmbientSpeed` is **15**
AmbientSpeedUnit	Specifies the unit of measure to use to calculate the ambient speed. Available speed units are: • MPH (miles per hour) • KPH (kilometers per hour) • MTPS (meters per second) • MTPM (meters per minute) **Note:** Default `AmbientSpeedUnit` is **MPH**
AmbientSpeed.RoadType	Specifies the ambient speed to use for off-network travel based on the road type. You must specify both the road type and the new speed for that road type. The speed is defined in the *AmbientSpeedUnit*. Road types can be returned in all supported types. For a list of road type enumerations, see **REST API Road Type Enumeration** on page 133.
propagationFactor	See **Query Parameters** on page 49 for description of propagationFactor. **Note:** This parameter is only supported in `getTravelBoundary` version 2 is specified.

Parameter	Description
RoadType	Specifies the propagationFactor to use for off-network travel based on the road type. You must specify both the road type and the new value of propagationFactor for that road type. See **propagationFactor** to know more. Road types can be returned in all supported types. For a list of road type enumerations, see **REST API Road Type Enumeration** on page 133. **Note:** This parameter is only supported in `getTravelBoundary` version 2 is specified.
avoid	Specifies a comma-separated list of road types to be avoided during travel boundary calculation. This is a String parameter. When a road type is provided as the value of the parameter, the boundary excludes that type of roads in the calculation. For example, if tollRoad is provided as the parameter value, the calculated boundary will have no toll roads.

Example with ambient speed parameters in HTTP POST payload in gettravelboundary version 1.

```
{
 "DefaultAmbientSpeed": 45,
 "AmbientSpeedUnit": "MPH"

 "AmbientSpeed.RoadType.PrimaryHighwayUrban": 15,
 "AmbientSpeed.RoadType.SecondaryHighwayUrban": 10
}
```

Example with ambient speed and propagationFactor parameters in HTTP POST payload in gettravelboundary version 2.

```
{
 "ambientSpeeds": {
  "defaultAmbientSpeed": 24,
  "ambientSpeedUnit": "MPH",
  "ambientSpeedOverrides": {
   "Primary Highway Urban": ".51",
   "Secondary Highway Urban": ".1"
  }
 },
 "propagationFactors": {

  "propagationFactor": "1",
  "propagationFactorOverrides": {
   "Primary Highway Urban": ".51",
   "Secondary Highway Urban": ".1"
  }
 }
}
```

```
{
 "ambientSpeeds": {
  "ambientSpeedOverrides": {
   "Primary Highway Urban": 25,
   "Secondary Highway Urban": 10
  }
 },
 "propagationFactors": {

  "propagationFactor": "0.2",
  "propagationFactorOverrides": {
   "Primary Highway Urban": "0.51",
   "Secondary Highway Urban": "0.1"
  }
 }
}
```

Note: The response from REST request will be in JSON format and the geometries will be of GEOJSON format.

GetRoute

Description

The `GetRoute` service returns routing information for a set of two distinct points or multiple points. It takes a starting location and an ending location with optional intermediate points as input, and returns the route that is either the fastest or the shortest.

Note: Response from REST service will be in JSON format and the geometry returned will be in GeoJSON format

HTTP GET URL Format

The following format is used for HTTP GET requests. HTTP GET is used for simple routes where no additional JSON payload is required. Intermediate points can also be added to the HTTP GET request.

```
HTTP GET
/webApp-context/services/databases/dbsource.json?q=route&query_parameters
```

Where: *dbsource* is the name of the database that contains the data to use for the route. Use the database name specified in the database resource file (dbList.json file). *webApp-context* is the endpoint to your web application or service. If you are using the default database specified in the database resource file, use `default.json` in the REST URL for *dbsource*.

HTTP POST URL Format

The following format is used for HTTP POST requests:

```
HTTP POST:
/webApp-context/services/databases/dbsource.json?q=route&query_parameters
POST BODY: Content-Type:application/json {Route Data}
```

Route Data is the POST json body (Content-Type: application/json) for the additional route information to be used in the calculation containing intermediate points or priority for road types. For information and examples on these options, see **GetRoute HTTP POST Options** on page 66.

Query Parameters

This operation takes the following query parameters.

Parameter	Type	Required	Description
startPoint	String	Yes	Specifies the start location of the route in the format: x,y,coordSys. For example: -74.2,40.8,epsg:4326
endPoint	String	Yes	Specifies the end location of the route in the format: x,y,coordSys. For example: -74.2,40.8,epsg:4326
intermediatePoints	String	No	Specifies the list of intermediate points to include along the route. To include in the HTTP GET request, use the format: Long,Lat,Long,Lat,…,coordsys. For example: -74.2,40.8,-73,42,epsg:4326. To include a set of intermediate points in a HTTP POST request, add the MultiPoint JSON payload indicating the points that the route will include. If intermediate points are specified both in the URL and in the json payload, the json payload is given preference, and the intermediate points in URL are ignored.
oip	Boolean	No	Specifies a processing parameter that indicates if the intermediate points should be optimized. The default is false. By default the intermediate points will be used in the calculation in the order specified. If set to true, the specified points will be re-ordered in an optimal manner during route computation.

Parameter	Type	Required	Description
returnIntermediatePoints	Boolean	No	Specifies whether to return the intermediate points with the route response. The default is false. For any value other that true or false, it will default to false. This option will return the intermediate points in order as specified in the POST body. If the value of option *oip* is set to true, then this option will return intermediate points in optimized order.
destinationSrs	String	No	Specifies the coordinate system to return the route and resulting geometries. The default is the coordinate system of the data used.
optimizeBy	String	No	Specifies the type of optimizing to use for the route. Valid values are *time* or *distance*. The default is time.
returnDistance	Boolean	No	Specifies the route directions include the distance traveled. The default is true.
distanceUnit	String	No	Specifies the units to return distance. The default is m (meter). Available values are: • m (meter) • km (kilometer) • yd (yard) • ft (foot) • mi (mile)
returnTime	Boolean	No	Specifies the route directions including the time it takes to follow a direction. The default is true.
timeUnit	String	No	Specifies the units to return time. The default is min (minute). Available values are: • min (minute) • msec (millisecond) • s (second) • h (hour)

Parameter	Type	Required	Description
language	String	No	Specifies the language the travel directions should be returned, only if route directions are returned (if directionsStyle is defined as Normal or Terse). The default being English (en). Directions can be returned in all supported languages. For a list of language enumerations, see **REST API Language Enumeration** on page 135.
returnDirectionGeometry	Boolean	No	Specifies wheher to include the geometry associated with each route instruction in route response. The default is false.
directionsStyle	String	No	Specifies the type of route directions to be returned. Default value is None. Specify this parameter if you required route directions to be returned. The options when specifying route directions are: • **None:** No directions returned. Default, if not specified. • **Normal:** Directions are returned in a full format, appropriate for web-based applications. • **Terse:** Directions are returned in a short format, appropriate for mobile applications.
segmentGeometryStyle	String	No	Specifies the format of the geometry that represents a segment of the route. Default value is None. Specify this parameter if you required segment geometries to be returned. The options when specifying route directions are: • **None:** No geometric representation of a segment will be returned. Default, if not specified. • **End:** Each segment of the route will be returned with just its endpoints in a LineString. • **All:** Each segment will be returned with all its shape points as a LineString. The LineString can be used as an overlay on a map.
primaryNameOnly	Boolean	No	Specifies whether to return all names for a given street in the directions or to return just the primary name for a street. Only used when route directions are returned. The default being false.

Parameter	Type	Required	Description
majorRoads	Boolean	No	Specifies whether to include all roads in the calculation or just major roads. If you choose to include only major roads, performance will improve but accuracy may decrease. The default is false.
historicTrafficTimeBucket	String	No	Specifies whether the routing calculation uses the historic traffic speeds. These speeds are based on different time-of-day buckets. The data must have historic traffic speeds included in order to use this feature. The data for each country/region has the same bucket definitions, where the speeds for these bucket values may vary. The options are: • **None:** The default value. Historic traffic data is not used in the calculation. Instead an averaged speed value is used. • **AMPeak:** Calculate routes with the peak AM speeds. The AMPeak time bucket is from 07:00 to 10:00 hrs. time of day. • **PMPeak:** Calculate routes with the peak PM speeds. The PMPeak time bucket is from 16:00 to 19:00 hrs. time of day. • **OffPeak:** Calculate routes with the off peak (daytime) speeds. The OffPeak time bucket is from 10:00 to 16:00 hrs. time of day. • **Night:** Calculate routes with the nighttime speeds. The Night time bucket is from 22:00 to 04:00hr time of day.
avoid	String	No	Specifies a comma-separated list of road types to be avoided during route calculation. This is a String parameter. When a road type is provided as the value of the parameter, the route excludes that type of roads in route calculation. For example, if tollRoad is provided as the parameter value, the calculated route contains a route without any toll roads.
version	String	No	Specifies the version of the `GetRoute` REST service. Valid values are *1* and *2*. The default value for version is *1*.

Parameter	Type	Required	Description
localRoadsLoadFactor	String	No	Specifies the number of local roads that can be loaded into memory during route or matrix calculation. The number of roads that can load is directly proportional to the value selected in the parameter where 1 is the minimum and 3 is the maximum value. Valid values can be 1,2, or 3. The default is 1. See **Local Roads Load Factor** on page 126 for a detailed description and impact of the parameter on routing or matrix calculation. **Note:** The parameter does not accept decimal values.

Examples

Simple Route with start and end points.

```
http://server:port/webApp-context/services/databases/usroutedatabase.json?q=route&startPoint=-73.97,40.79,epsg:4326&endPoint=-73.98,40.74,epsg:4326
```

Response

```
{
 "time": 7.67,
 "timeUnit": "min",
 "distance": 8865,
 "distanceUnit": "m"
}
```

Route with intermediate points.

```
http:
//server:port/webApp-context/services/databases/usroutedatabase.json?
q = route & startPoint = -73.970257, 40.794045, epsg: 4326 & endPoint =
 -73.972103, 40.786605,
epsg: 4326 & intermediatePoints = -73.976266, 40.788717, -73.973562,
40.792193, -73.971802, 40.794630,
epsg: 4326 & oip = true & returnIntermediatePoints = true
```

Response

```
"intermediatePoints": {
 "type": "MultiPoint",
 "coordinates": [
  [-73.971802,
   40.79463
  ],
  [-73.973562,
   40.792193
  ],
  [-73.976266,
   40.788717
  ]
 ]
}
```

Route with directions enabled.

```
http:
//server:port/webApp-context/services/databases/usroutedatabase.json?
q = route & startPoint = -73.97, 40.79, epsg: 4326 & endPoint = -73.98,
 40.74,
epsg: 4326 & language = en & directionsStyle = Normal &
returnDirectionGeometry = true
```

Response

```

{
  "time": 10.58,
  "timeUnit": "min",
  "distance": 9035,
  "distanceUnit": "m",
  "language": "en",
  "directionsStyle": "Normal",
  "routeDirections": [
  {
   "time": 0.03,
   "timeUnit": "min",
   "distance": 25,
   "distanceUnit": "m",
   "instruction": "",
   "directionGeometry":
    {
     "type": "LineString",
     "coordinates": [[[...]]]

    }
  },
  {
   "time": 0.7,
   "timeUnit": "min",
   "distance": 394,
   "distanceUnit": "m",
   "instruction": "Turn right on W 91st St and travel West 394.0 m
   (0.7 min).",
   "directionGeometry":
    {
     "type": "LineString",
     "coordinates": [[[...]]]
     ]
   }
  },
  {
   "time": 0.37,
   "timeUnit": "min",
   "distance": 352,
   "distanceUnit": "m",
   "instruction": "Turn left on Broadway and travel Southwest 352.0
   m (0.4 min).",
   "directionGeometry":
   {
     "type": "LineString",
     "coordinates": [[[...]]]

   }
  }
 ]
}
```

Version specific error response

When you enter an invalid parameter value (for example, points falling outside of the boundaries) in a request, the error response you get depends on the version entered by you. When the version is 1, you get value and error whereas when the version is 2, the response only contains the error.

- Request when version is *1*:

```
http://server:port/webApp-context/services/databases/usroutedatabase.json?
&q=route&startPoint=-14.321600,60.662859,epsg:4326&endPoint=-74.035208,40.695624,
epsg:4326&distanceUnit=km&version=1
```

- Response:

```
{
  "value": "Point outside boundaries: (-14.3216,60.662859,0)",
  "errors": [
    {
      "errorCode": 5008,
     "userMessage": "Point outside boundaries: (-14.3216,60.662859,0)"

    }
  ]
}
```

- Request when version is *2*:

```
http://server:port/webApp-context/services/databases/usroutedatabase.json?
&q=route&startPoint=-14.321600,60.662859,epsg:4326&endPoint=-74.035208,40.695624,
epsg:4326&distanceUnit=km&version=2
```

- Response:

```
{
  "errors": [
    {
      "errorCode": 5008,
     "userMessage": "Point outside boundaries: (-14.3216,60.662859,0)"

    }
  ]
}
```

GetRoute HTTP POST Options

HTTP POST URL Format

In addition to the regular HTTP GET parameters, you can add HTTP POST payload options to your request that specify intermediate points and priority for road types. The content type must be set to application/json. The following format is used for HTTP POST requests:

```
HTTP POST:  /webApp-context/services/databases/dbsource.json?q=route&
query_parameters
POST BODY: Content-Type:application/json {Route Data}
```

Route Data is the POST json body (Content-Type: application/json) for the additional route information to be used in the calculation containing ambient speeds for road types.

Intermediate Points

A list of intermediate points to include along the route. To include a set of intermediate points in a HTTP POST request, add the MultiPoint JSON payload indicating the points that the route will include. If intermediate points are specified both in the URL and in the json payload, the json payload is given preference, and the intermediate points in URL are ignored.

Example intermediate points HTTP POST payload.

```
{
 "intermediatePoints": {
  "type": "MultiPoint",
  "crs": {
   "type": "name",
   "properties": {
    "name": "epsg:4326"
   }
  },
  "coordinates": [[-73.976266, 40.788717], [-73.973562, 40.792193],
[-73.971802, 40.794630]]
 }
}
```

Transient Updates

See **Transient Updates** on page 102 for details.

Road Type Priority

It specifies the priority to be given to different types of roads when determining the route. The following is a description of the road type priority options:

Option	Description
high	Prefer the road type over other road types.
medium	Give this road type equal preference with other road types. If no preference is specified for a road type, the default is Medium.
low	Prefer other road types over this road type.
avoid	Exclude the road type from routes if possible. It is not always possible to exclude a road type from the travel directions. Depending on the situation, the alternative to an avoided road type may be so poor that the software will choose a route that uses an avoided road type. Also, if the starting or ending point lies along a segment whose road type has been avoided, the software will still use that segment.

Example road type priority HTTP POST payload.

```
{
 " roadTypesPriority ": {
  "RoadType.MajorRoadDenseUrban": "High",
  "RoadType.LimitedAccessDenseUrban": "Low",
  "RoadType.LimitedAccessRural": "Medium",
  "RoadType.PrimaryHighwayUrban": "Avoid"
 }
}
```

GetRouteCostMatrix

Description

The `GetRouteCostMatrix` service calculates the travel time and distances between an array of start and end locations and returns the route that is either the fastest or the shortest. The result determines the total time and distance of the individual routes (the route costs). For example if you input four start points and four end points, a total of 16 routes will be calculated.

Note: Response from REST service will be in JSON format and the geometry returned will be in GeoJSON format

HTTP GET URL Format

The following format is used for HTTP GET requests. HTTP GET is used for simple cost calculations where no additional JSON payload is required.

```
HTTP GET
/webApp-context/services/databases/dbsource.json?q=routeCostMatrix&query_parameters
```

Where: *dbsource* is the name of the database that contains the data to use for the route. Use the database name specified in the database resource file (dbList.json file). *webApp-context* is the endpoint to your web application or service. If you are using the default database specified in the database resource file, use `default.json` in the REST URL for *dbsource*.

HTTP POST URL Format

The following format is used for HTTP POST requests:

```
HTTP POST:
/webApp-context/services/databases/dbsource.json?q=routeCostMatrix&query_parameters
POST BODY: Content-Type:application/json {Route Data}
```

Route Data is the POST json body (`Content-Type: application/json`) for the additional route information to be used in the calculation if the list of input points exceeds the limits of the caller's URL buffer, as well as including priority for road-types.

Query Parameters

This operation takes the following query parameters:

Parameter	Type	Required	Description
startPoints	String	Yes	Specifies the start locations of the route in the format: long,lat,long,lat,...,coordSys. For example: -74.2,40.8,-73,42,epsg:4326
endPoints	String	Yes	Specifies the end locations of the route in the format: long,lat,long,lat,...,coordSys. For example: -74.2,40.8,-73,42,epsg:4326
destinationSrs	String	No	Specifies the coordinate system to return the route and resulting geometries. The default is the coordinate system of the data used.
optimizeBy	String	No	The type of optimizing to use for the route. Valid values are *time* or *distance*. The default is time.
returnDistance	Boolean	No	Specifies the route directions including the distance traveled. The default is true. Both returnDistance and returnTime parameters cannot be false in the same request.
distanceUnit	String	No	Specifies the units to return distance. The default is m (meter). Available values are: • m (meter) • km (kilometer) • yd (yard) • ft (foot) • mi (mile)
returnTime	Boolean	No	Specifies the route directions including the time it takes to follow a direction. The default is true. Both returnDistance and returnTime parameters cannot be false in the same request.

Parameter	Type	Required	Description
timeUnit	String	No	Specifies the units to return time. The default is min (minute). Available values are: • min (minute) • msec (millisecond) • s (second) • h (hour)
majorRoads	Boolean	No	Specifies whether to include all roads in the calculation or just major roads. If you choose to include only major roads, performance will improve but accuracy may decrease. The default is false.
returnOptimalRoutesOnly	Boolean	No	Specifies whether to return only the optimized route for each start point or end point combination. The default is true. The optimized route is either the fastest route or the shortest distance, depending on the optimizeBy parameter.
historicTrafficTimeBucket	String	No	Specifies whether the routing calculation uses the historic traffic speeds. These speeds are based on different time-of-day buckets. The data must have historic traffic speeds included in order to use this feature. The data for each country/region has the same bucket definitions, where the speeds for these bucket values may vary. The options are: • **None:** The default value. Historic traffic data is not used in the calculation. Instead an averaged speed value is used. • **AMPeak:** Calculate routes with the peak AM speeds. The AMPeak time bucket is from 07:00 to 10:00 hrs. time of day. • **PMPeak:** Calculate routes with the peak PM speeds. The PMPeak time bucket is from 16:00 to 19:00 hrs. time of day. • **OffPeak:** Calculate routes with the off peak (daytime) speeds. The OffPeak time bucket is from 10:00 to 16:00 hrs. time of day. • **Night:** Calculate routes with the nighttime speeds. The Night time bucket is from 22:00 to 04:00 hrs. time of day.

Parameter	Type	Required	Description
avoid	String	No	Specifies a comma-separated list of road types to be avoided during route calculation. This is a String parameter. When a road type is provided as the value of the parameter, the route excludes that type of roads in route calculation. For example, if tollRoad is provided as the parameter value, the calculated route contains a route without any toll roads.
version	String	No	Specifies the version of the `GetRouteCostMatrix` REST service. Valid values are 1 and 2. The default value for version is *1*.
localRoadsLoadFactor	String	No	Specifies the number of local roads that can be loaded into memory during route or matrix calculation. The number of roads that can load is directly proportional to the value selected in the parameter where 1 is the minimum and 3 is the maximum value. Valid values can be 1,2, or 3. The default is 1. See **Local Roads Load Factor** on page 126 for a detailed description and impact of the parameter on routing or matrix calculation. **Note:** The parameter does not accept decimal values.

Examples

Cost matrix route with two start and two end points.

```
http:
//<server>:<port>/<webApp-context>/services/<databases>/usroutedatabase.json?
q = routeCostMatrix & startPoints = -73.56565, 40.5545, -73.46565,
40.4545, epsg: 4326 &
endPoints = -73.34343, 40.667, -73.14343, 40.267, epsg: 4326 &
returnOptimalRoutesOnly = true &
optimizeBy = distance & distanceUnit = km & timeUnit = min & majorRoads
 = true &
destinationSrs = epsg: 4322 & returnTime = false
```

Response

```
{
 "matrix":
 [{
   "startPoint": {
     "type": "Point",
     "coordinates":
     [
      -73.56567672202618,
      40.554384822358614
     ],
     "crs": {
      "type": "name",
      "properties": {
       "name": "epsg:4322"
      }
     }
    },
    "endPoint": {
     "type": "Point",
     "coordinates":
     [
      -73.34345711862802,
      40.66688488742393
     ],
     "crs": {
      "type": "name",
      "properties": {
       "name": "epsg:4322"
      }
     }
    },
    "distance": 35.268,
    "distanceUnit": "km
                  },
                  {
                      " startPoint ":
                      {
                          " type ": " Point ",
                          " coordinates ":
                          [
                              -73.46567684021008,
                              40.454384834155185
                          ],
                          " crs ":
                          {
                              " type ": " name ",
                              " properties ":
                              {
                                  " name ": " epsg: 4322 "
                              }
                          }
                      },
                      " endPoint ":
                      {
                          " type ": " Point ",
                          " coordinates ":
                          [
                              -73.34345711862802,
                              40.66688488742393
                          ],
                          " crs ":
                          {
                              " type ": " name ",
                              " properties ":
                              {
                                  " name ": " epsg: 4322 "
                              }
                          }
                      },
                      " distance ": 44.444,
                      " distanceUnit ": " km "
                  }
              ]
    }
```

Version specific error response

When you enter an invalid parameter value (for example, points falling outside of the boundaries) in a request, the error response you get depends on the version entered by you. When the version is 1, you get value and error whereas when the version is 2, the response only contains the error.

- Request when version is *1*:

```
http://server:port/webApp-context/services/databases/usroutedatabase.json?
q=routeCostMatrix&startPoints=-73.56565,40.5545,-73.46565,40.4545,epsg:4326&
endPoints=-14.321600,60.662859,-73.14343,40.267,epsg:4326&version=1
```

- Response:

```
{
  "value": "Point outside boundaries: (-14.3216,60.662859,0)",
  "errors": [
    {
      "errorCode": 5008,
     "userMessage": "Point outside boundaries: (-14.3216,60.662859,0)"

    }
  ]
}
```

- Request when version is *2*:

```
http://server:port/webApp-context/services/databases/usroutedatabase.json?
q=routeCostMatrix&startPoints=-73.56565,40.5545,-73.46565,40.4545,epsg:4326&
endPoints=-14.321600,60.662859,-73.14343,40.267,epsg:4326&version=2
```

- Response:

```
{
  "errors": [
    {
      "errorCode": 5008,
     "userMessage": "Point outside boundaries: (-14.3216,60.662859,0)"

    }
  ]
}
```

Avoid Specific Routes

Toll Roads

Avoid Toll Roads feature

This feature allows the user to select a route with or without a toll road. This is a String parameter. Avoid is the parameter in which `tollroad` can be provided as the value in a SOAP request. In this case the resultant route will exclude toll roads while calculating the routes. The following example explains how this parameter is used.

Example with toll road

The following example explains the feature with some imaginary points for a route, which contains the avoid toll road parameter as `<v1:Avoid>tollroad</v1:Avoid>` in the request.

```
<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
                  xmlns:v1="http://www.mapinfo.com/routing/service/v1"

xmlns:v11="http://www.mapinfo.com/midev/service/geometries/v1">
   <soapenv:Header/>
   <soapenv:Body>
      <v1:RouteCostMatrixRequest id="?" locale="?">
         <!--Optional:-->


<v1:DatasetResourceName>${#Project#TollRoad_DB}</v1:DatasetResourceName>


    <v1:StartPoints srsName="epsg:4326">
       <!--Zero or more repetitions:-->
       <v11:Point srsName="epsg:4326">
          <v11:Pos>
```

```
              <v11:X>12.822214</v11:X>
              <v11:Y>47.282809</v11:Y>
           </v11:Pos>
        </v11:Point>
     </v1:StartPoints>

     <v1:EndPoints srsName="epsg:4326">
        <!--Zero or more repetitions:-->
        <v11:Point srsName="epsg:4326">
           <v11:Pos>
              <v11:X>12.873852</v11:X>
              <v11:Y>46.871467</v11:Y>
           </v11:Pos>
        </v11:Point>

     </v1:EndPoints>

     <v1:DistanceUnit>Mile</v1:DistanceUnit>
     <v1:TimeUnit>Minute</v1:TimeUnit>
      <v1:ReturnOptimalRoutesOnly>true</v1:ReturnOptimalRoutesOnly>
      <v1:OptimizeBy>distance</v1:OptimizeBy>
      <v1:MajorRoads>false</v1:MajorRoads>
      <v1:ReturnDistance>true</v1:ReturnDistance>
      <v1:ReturnTime>true</v1:ReturnTime>
      <v1:Avoid>tollroad</v1:Avoid>
      <v1:HistoricTrafficTimeBucket>none</v1:HistoricTrafficTimeBucket>

      </v1:RouteCostMatrixRequest>
   </soapenv:Body>
</soapenv:Envelope>
```

Example without toll road

The following example explains the feature with some imaginary points for a route, which does not contain the avoid toll road parameter in the request.

```
<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
                  xmlns:v1="http://www.mapinfo.com/routing/service/v1"

xmlns:v11="http://www.mapinfo.com/midev/service/geometries/v1">
   <soapenv:Header/>
   <soapenv:Body>
      <v1:RouteCostMatrixRequest id="?" locale="?">
         <!--Optional:-->


<v1:DatasetResourceName>${#Project#TollRoad_DB}</v1:DatasetResourceName>


     <v1:StartPoints srsName="epsg:4326">
```

```
        <!--Zero or more repetitions:-->
        <v11:Point srsName="epsg:4326">
           <v11:Pos>
              <v11:X>12.822214</v11:X>
              <v11:Y>47.282809</v11:Y>
           </v11:Pos>
        </v11:Point>
     </v1:StartPoints>

     <v1:EndPoints srsName="epsg:4326">
        <!--Zero or more repetitions:-->
        <v11:Point srsName="epsg:4326">
           <v11:Pos>
              <v11:X>12.873852</v11:X>
              <v11:Y>46.871467</v11:Y>
           </v11:Pos>
        </v11:Point>

     </v1:EndPoints>

     <v1:DistanceUnit>Mile</v1:DistanceUnit>
     <v1:TimeUnit>Minute</v1:TimeUnit>
      <v1:ReturnOptimalRoutesOnly>true</v1:ReturnOptimalRoutesOnly>
      <v1:OptimizeBy>distance</v1:OptimizeBy>
      <v1:MajorRoads>false</v1:MajorRoads>
      <v1:ReturnDistance>true</v1:ReturnDistance>
      <v1:ReturnTime>true</v1:ReturnTime>

      <v1:HistoricTrafficTimeBucket>none</v1:HistoricTrafficTimeBucket>

      </v1:RouteCostMatrixRequest>
    </soapenv:Body>
</soapenv:Envelope>
```

GetRouteCostMatrix HTTP POST Options

HTTP POST URL Format

In addition to the regular HTTP GET parameters, you can add HTTP POST payload options to your request that specifies priority for road types. The HTTP POST payload can also be used if the list of input points exceeds the limits of the caller's URL buffer. The content type must be set to application/json. The following format is used for HTTP POST requests:

```
HTTP POST: /webApp-context/services/databases/
dbsource.json?q=routeCostMatrix&query_parameters
POST BODY: Content-Type:application/json {Route Data}
```

Route Data is the POST json body (`Content-Type: application/json`) for the additional route information to be used in the calculation containing ambient speeds for road types.

Define Start and End Points

To include a set of start or end points in a HTTP POST request, add the MultiPoint JSON payload indicating the points that the route will include. When Start and End points are defined in the HTTP POST payload, the *startPoints* and *endPoints* parameters are not mandatory query parameters in the URL. If they are defined in the URL they are ignored. A warning message is logged in `spectrum-server.log` file when points in URL are ignored.

Example start points HTTP POST payload.

```
{
 "startPoints": {
  "type": "MultiPoint",
  "crs": {
   "type": "name",
   "properties": {
    "name": "epsg:4326"
   }
  },
  "coordinates": [[-73.976266, 40.788717],
   [-73.973562, 40.792193], [-73.971802, 40.794630]]
 }
}
```

Example end points HTTP POST payload.

```
{
 "endPoints": {
  "type": "MultiPoint",
  "crs": {
   "type": "name",
   "properties": {
    "name": "epsg:4326"
   }
  },
  "coordinates": [[-73.976266, 40.788717],
   [-73.973562, 40.792193], [-73.971802, 40.794630]]
 }
}
```

Commercial Vehicle Restrictions

Commercial vehicle restrictions are composed of directives to the routing engine that guides the behavior and attributes of commercial vehicles making trips along the route. Depending upon vehicle attributes provided (such as height, width, length, weight) and the commercial vehicle restriction attributes present in the road network, decision is made whether to allow to route a particular vehicle over a segment or not. If there is no commercial vehicle restriction attribute present in road network, input restriction parameters will have no effect in the resultant route.

Following are the set of parameters for commercial vehicle restrictions:

looseningBarrierRestrictions

Specifies that the barriers will be removed when determining the route. These restrictions are most often when a commercial vehicle is prohibited from traversing a segment due to local ordinance or a commercial vehicle is allowed on the segment but only when it must (for example, last mile access, local delivery, and so on). Routes where a barrier has been removed will still have a higher route cost even if the route it shorter/faster than a route with no barrier.

vehicleAttributes

Specifies that details of the vehicle that will be restricted based on the type, height, weight, length, or width when determining the route. Commercial vehicles are divided into different types ranging from short trailers to long triples. The Commercial Vehicle Restrictions attribution is organized on a per-vehicle type basis. This means it is entirely possible for a segment to be preferred for one vehicle type and the same segment have a restriction for another type of vehicle. Use the following types of vehicle information:

vehicleAttributes Option	Description
vehicleType	Choose either ALL or one of the types of vehicles: • STRAIGHT • SEMI_TRAILOR • STANDARD_DOUBLE • INTERMEDIATE_DOUBLE • LONG_DOUBLE • TRIPLE • OTHER_LONG_COMBINATION_VEHICLE
weight	Specifies the maximum weight of a vehicle. Any vehicles over this value will be restricted when determining the route. The units of weight are: • kg • lb • mt • t

vehicleAttributes Option	Description
height	Specifies the maximum height of a vehicle. Any vehicles over this value will be restricted when determining the route. The units of height are: • ft • yd • mi • m • km
length	Specifies the maximum length of a vehicle. Any vehicles over this value will be restricted when determining the route. The units of length are: • ft • yd • mi • m • km
width	Specifies the maximum width of a vehicle. Any vehicles over this value will be restricted when determining the route. The units of width are: • ft • yd • mi • m • km

Note: You need to specify either weight/height or length/width along with its corresponding unit.

Transient Updates

See **Transient Updates** on page 102for details.

Examples

Without Vehicle Restrictions

Request

```
HTTP GET /webApp-context/services/databases/US_CVR.json?q=route&
startPoint=-74.7221203,42.9737073,epsg:4326& endPoint
=-74.6671887,42.8097083,epsg:4326
```

Response

```
{
 "distance": 24.87,
 "distanceUnit": "mi",
 "time": 36.57,
 "timeUnit": "min"
}
```

With Vehicle Restrictions Request

```
HTTP POST /webApp-context/services/databases/US_CVR.json?q=route&
startPoint=-74.7221203,42.9737073,
epsg:4326& endPoint=-74.6671887,42.8097083,epsg:4326
POST BODY: Content-Type:application/json {CVR Data}
```

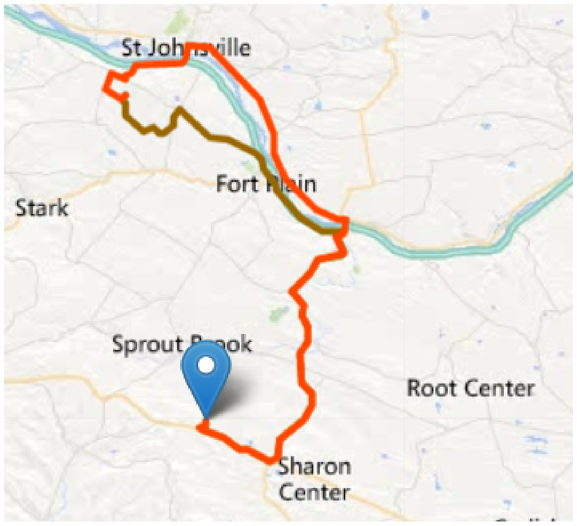
Commercial Vehicle Restriction HTTP POST payload.

```
{
 "cvr": {
  "looseningBarrierRestrictions": "n",
  "vehicleAttributes": {
   "vehicleType": "ALL",
   "heightUnit": "meter",
   "height": "4",
   "weightUnit": "Kilogram",
   "weight": "40000"
  }
 }
}
```

Response:

```
{
 "distance": 27.92,
 "distanceUnit": "mi",
 "time": 37.48,
 "timeUnit": "min"
}
```

The two routes in the map below show the CVR applied for the same start and end locations. The route displayed in the brown color is the one without CVR and the route displayed in the red color is with CVR. Notice the deviation in route in the beginning of the journey; this is due to the height and weight restrictions applied to them.



Road Type Priority

Specifies the priority to give to different types of roads when determining the route. The following is a description of the road type priority options:

Option	Description
high	Prefer the road type over other road types.
medium	Give this road type equal preference with other road types. If no preference is specified for a road type, the default is Medium.
low	Prefer other road types over this road type.
avoid	Exclude the road type from routes if possible. It is not always possible to exclude a road type from the travel directions. Depending on the situation, the alternative to an avoided road type may be so poor that the software will choose a route that uses an avoided road type. Also, if the starting or ending point lies along a segment whose road type has been avoided, the software will still use that segment.

Example road type priority HTTP POST payload.

```
{
 " roadTypesPriority ": {
  "RoadType.MajorRoadDenseUrban": "High",
  "RoadType.LimitedAccessDenseUrban": "Low",
  "RoadType.LimitedAccessRural": "Medium",
  "RoadType.PrimaryHighwayUrban": "Avoid"
 }
}
```

Response for Multiple Error in a Single Request

In case when an error occurs for REST when a request contains invalid query parameters in GET URL or invalid payload for POST, we get a cumulative error response in one go in a JSON array.

For example, in case of a `GetTravelBoundary` request:

```
http://<server>:<port>/<webApp-context>/services/databases/<usroutedatabase>.json?
q=travelBoundary&point=-77.092609,38.871256,epsg:4326&
costs=5,a&costUnit=mindd
```

Response

```
{
 "errors":
 [{
   "errorCode": 4179,
   "userMessage": "Error parsing Point CoordSys"
  }, {
   "errorCode": 3010,
   "userMessage": "Invalid costUnit: minddd"
  }, {
   "errorCode": 3023,
   "userMessage": "Invalid getTravelBoundary cost, must be a positive
number : a, v"
  }
 ],
 "value": "Error parsing Point CoordSys"
}
```

The same error processing applies for `GetRoute` and `GetRouteCostMatrix` operations. The `value` node in the response JSON is deprecated and will be removed in future releases. For error checking only the `errors` node should be utilized.

GetSegmentData

Description

The `GetSegmentData` service returns segment information for a point or segment ID. When a point is specified, the closest route segments are returned. When a segment ID is specified, the route data for that specified route segment data is returned.

> **Note:** The response from the REST service comes in JSON format. When a request contains invalid query parameters in the GET URL , a cumulative error response is returned in a JSON array. The value node in the response JSON is deprecated. For error checking, only the errors node should be utilized.

HTTP GET URL Format

The following format is used for HTTP GET requests. The HTTP GET requests are different for either returning segment data at a point, or returning segment data for a segment ID.

Returning data for a segment at a specified point:

```
HTTP GET /webApp-context/services/databases/dbsource/
segments.json?point=x,y,srsName&query_parameters
```

Returning data for a specified segment:

```
HTTP GET /webApp-context/services/databases/dbsource/segments/
segmentID.json?query_parameters
```

Where `dbsource` is the name of the database that contains the data to use for the route. Use the database name specified in the Database Resource tool. The `segmentID` is segment identifier you want to return the data.

Query Parameters

This operation takes these query parameters.

Parameter	Type	Required	Description
destinationSrs	String	No	Specifies the coordinate system to return the segment data and resulting geometry. The default is the coordinate system of the data used.

Parameter	Type	Required	Description
distanceUnit	String	No	Specifies the units to return distance. The default is m (meter). Available values are: • m (meter) • km (kilometer) • yd (yard) • ft (foot) • mi (mile)
timeUnit	String	No	Specifies the units to return time. The default is min (minute). Available values are: • min (minute) • msec (millisecond) • s (second) • h (hour)
velocityUnit	String	No	Specifies the units in which the speed will be returned. The default is mph (miles per hour). Available values are: • mph (miles per hour) • kph (kilometers per hour)
angularUnit	String	No	Specifies the units to return turn angles. The default is deg (degree). Available values are: • deg (degree) • rad (radian) • minute (minute) • sec (second) • grad (grad)

Parameter	Type	Required	Description
segmentGeometryStyle	String	No	Specifies the format of the geometry that represents a segment of the route. Default value is None. Specify this parameter if you required segment geometries to be returned. The options when specifying route directions are: • **None:** No geometric representation of a segment will be returned. Default, if not specified. • **End:** Each segment of the route will be returned with just its endpoints in a `LineString`. • **All:** Each segment will be returned with all its shape points as a `LineString`. The `LineString` can be used as an overlay on a map.
version	String	No	Specifies the version of the `GetSegmentData` REST service. Valid values are 1 and 2. The default value for version is *1*.

Examples

Return segment data specifying a point

```
http://<server>:<port>/<webApp-context>/services/databases/<US>/
segments.json?point=-77,38,epsg:4326&segmentGeometryStyle=all
```

Response

```
[{
  "segmentID": "aa18eb33:1b7bbe",
  "primaryName": "VA-631",
  "primaryNameLanguage": "en",
  "alternateNames": [{
    "alternateName": "Lloyds Rd",
    "language": "en"
   }, {
    "alternateName": "VA-631",
    "language": "en"
   }
  ],
  "segmentLength": 4.954,
  "segmentLengthUnit": "mi",
  "timeTaken": 5.9333,
  "timeUnit": "min",
  "turnAngle": 0.0,
  "turnAngleUnit": "deg",
  "compassDirection": "",
  "speedOfTravel": 49.9955,
  "speedOfTravelUnit": "mph",
  "roadType": "major road rural",
  "segmentDirection": "bidirectional",
  "startJunctionType": "",
  "endJunctionType": "Other",
  "isRoundabout": false,
  "isTollRoad": false,
  "geometry": {
   "type": "LineString",
   "crs": {
    "type": "name",
    "properties": {
     "name": "epsg:4326"
    }
   },
   "coordinates": [[…]]
  }
 },
 },
 "coordinates": [[…]]
 }
 }
 ]
```

Return segment data specifying a `segmentID`

```
http://<server>:<port>/webApp-context/services/databases/usroutedatabase/
segments/aa18eb33:1b7bbe.json?distanceUnits=mi
```

Response

```
[{
   "segmentID": "aa18eb33:1b7bbe",
   "primaryName": "VA-631",
   "primaryNameLanguage": "en",
   "alternateNames": [{
     "alternateName": "Lloyds Rd",
     "language": "en"
    }, {
     "alternateName": "VA-631",
     "language": "en"
    }
   ],
   "segmentLength": 4.954,
   "segmentLengthUnit": "mi",
   "timeTaken": 5.9333,
   "timeUnit": "min",
   "turnAngle": 0.0,
   "turnAngleUnit": "deg",
   "compassDirection": "",
   "speedOfTravel": 49.9955,
   "speedOfTravelUnit": "mph",
   "roadType": "major road rural",
   "segmentDirection": "bidirectional",
   "startJunctionType": "",
   "endJunctionType": "Other",
   "isRoundabout": false,
   "isTollRoad": false
  }
```

Version specific error response

When you enter an invalid parameter value (for example, point missing) in a request, the error response you get depends on the version entered by you. When the version is 1, you get value and error whereas when the version is 2, the response only contains the error.

- Request when version is *1*:

```
http://server:port/webApp-context/services/databases/usroutedatabase.json?version=1
```

- Response:

```
{
   "value": "Point cannot be empty.",
   "errors": [
{
      "errorCode": 4139,
      "userMessage": "Point cannot be empty."
                                 }
]
}
```

- Request when version is *2*:

```
http://server:port/webApp-context/services/databases/usroutedatabase.json?version=2
```

- Response:

```
{"errors": [
{
   "errorCode": 4139,
   "userMessage": "Point cannot be empty."
}
]
}
```

PersistentUpdates

Description

The `PersistentUpdate` service allows a user to override aspects of the network. The overrides can be done on a per-road type, at a specific point or at a specific segment. The persistent update is valid only for a specific data source and may not be valid after a data update.

Using persistent updates to make these types of modifications, you have the ability to:

- Exclude a point
- Exclude a segment
- Set the speed of a point, segment, or road type
- Change (increase or decrease) the speed of a point, segment, or road type by a value
- Change (increase or decrease) the speed of a point, segment, or road type by a percentage
- List persistent updates

> **Note:** Since persistent updates are changes made on a system-wide basis for routing data and all updates will persist, they should be used with caution. The response from the REST service will be a success message. When a request contains invalid query parameters in the GET URL or an invalid payload for POST, a cumulative error response will be returned in a JSON array. The value node in the response JSON is deprecated. For error checking, only the errors node should be utilized.

Version specific error response

When you enter an invalid parameter value (for example, multiple updates) in a request, the error response you get depends on the version entered by you. When the version is 1, you get value and error whereas when the version is 2, the response only contains the error.

- Request when version is *1*:

```
http://server:port/webApp-context/services/databases/usroutedatabase.json?
velocity=15.912&velocityUnit=KPH&velocityAdjustment=34&velocityPercentage=56&version=1
```

- Response:

```
{
  "value": "One of either Velocity or SpeedIncrease or SpeedDecrease is
 expected.",
  "errors": [
    {
      "errorCode": 3733,
      "userMessage": "One of either Velocity or SpeedIncrease or
SpeedDecrease is expected."
    }
  ]
}
```

- Request when version is *2*:

```
http://server:port/webApp-context/services/databases/usroutedatabase.json
?velocity=15.912&velocityUnit=KPH&velocityAdjustment=34&velocityPercentage=56&version=2
```

- Response:

```
{
  "errors": [
    {
      "errorCode": 3733,
      "userMessage": "One of either Velocity or SpeedIncrease or
SpeedDecrease is expected."
    }
  ]
}
```

Types of Persistent Updates

See the following sections for information and examples of the persistent update request types:

Point Updates

HTTP POST URL Format

The following format is used for HTTP POST requests. HTTP POST is used to set a persistent update to a point.

```
HTTP POST:/webApp-context/services/databases/
dbsource/persistentUpdates.json?point=x,y,srsName&
query_parameters
```

Where *dbsource* is the name of the database to update the route data. Use the database name specified in the Database Resource tool.

HTTP DELETE URL Format

The following format is used for HTTP DELETE requests. HTTP DELETE is used to remove a specific persistent update to a point.

```
HTTP DELETE: /webApp-context/services/databases/
dbsource/persistentUpdates.json?point=x,y,srsName&
resetType=query_parameters
```

Where *dbsource* is the name of the database that contains the persistent update to remove. Use the database name specified in the Database Resource tool.

Query Parameters

The HTTP POST operation takes the following query parameters.

Parameter	Type	Required	Description
exclude	String	no	Specifies whether to exclude the specified point from all route calculations. The parameter's existence in the URL specifies whether to exclude, not the parameter value.
velocity	String	no	Specifies the speed update where you can define the new speed of the point by specifying the new velocity. The default unit is mph (miles per hour) unless you specify the *velocityUnit* parameter.
velocityUnit	String	no	Specifies the unit of speed for the *velocity* or *velocityAdjustment* (miles per hour). For speed updates, the velocity unit can have one of the following values where mph is the default value: • mph (miles per hour) • kph (kilometers per hour).
velocityAdjustment	String	no	Specifies the speed update where you define a change in the speed of the point by specifying the change in velocity (unit and value). Speed values can be increased (positive value) or decreases (negative value). The default unit is mph (miles per hour) unless you specify the *velocityUnit* parameter.

Parameter	Type	Required	Description
velocityPercentage	Integer	no	Specifies the speed update where you define an increase in the speed of the point by specifying a percentage to increase (positive value) or decrease (negative value) the speed.
version	String	No	Specifies the version of the `PointUpdates` REST service. Valid values are *1* and *2*.

Reset Parameter

The HTTP DELETE operation takes the following query parameter.

Parameter	Type	Required	Description
resetType	String	no	Specifies whether to reset (undo) an update for a point. Available options are: • **speed:** Reset the speed update for a specific point. • **exclude:** Reset the exclude for a specific point.

Examples

Exclude a point (HTTP POST)

```
http://<server>:<port>/<webApp-context>/services/databases/<usroutedatabase>/persistentUpdates.json?point=-73.6,43.5,epsg:4326
&exclude=true
```

Remove a point exclude persistent update (HTTP DELETE)

```
http://<server>:<port>/<webApp-context>/services/databases/<usroutedatabase>/persistentUpdates.json?point=-73.6,43.5,
epsg:4326&resetType=exclude
```

Segment Updates

HTTP POST URL Format

The following format is used for HTTP POST requests. HTTP POST is used to set a persistent update to a segment.

```
HTTP POST: /webApp-context/services/databases/
dbsource/persistentUpdates/segments/
segment_id.json?query_parameters
```

Where *dbsource* is the name of the database to update the route data, and *segment_id* is the identifier of the segment to update. Use the database name specified in the Database Resource tool.

HTTP GET URL Format

The following format is used for HTTP GET requests. HTTP GET is used to return a list of persistent updates for segments.

```
HTTP GET: /webApp-context/services/databases/
dbsource/persistentUpdates/segments/segment_id.json
                or
HTTP GET: /webApp-context/services/databases/dbsource/
persistentUpdates/segments.json?segments=segment_id
```

Where *dbsource* is the name of the database to return to persistent updates from, and *segment_id* is the segment to return updates.

> **Note:** The first format is used to return the persistent update for only one segment. The second format is used to return either multiple segments or all segments. For multiple segments, use a comma separated list of segment ids. For all segments, use an empty segments= parameter. See examples below.

HTTP DELETE URL Format

The following format is used for HTTP DELETE requests. HTTP DELETE is used to remove a specific persistent update to a segment.

```
HTTP DELETE: /webApp-context/services/databases/
dbsource/persistentUpdates/segments?mo=segment_id
&resetType=query_parameters
```

Where *dbsource* is the name of the database, and *segment_id* is the identifier of the segment to update that contains the persistent update to remove. Use the database name specified in the Database Resource tool.

Query Parameters

The HTTP POST operation takes the following query parameters.

Parameter	Type	Required	Description
exclude	String	no	Specifies whether to exclude the specified segment from all route calculations. The parameter's existence in the URL specifies whether to exclude, not the parameter value.
velocity	String	no	Specifies the speed update where you can define the new speed of the segment by specifying the new velocity. The default unit is mph (miles per hour) unless you specify the *velocityUnit* parameter.
velocityUnit	String	no	Specifies the unit of speed for the *velocity* or velocityAdjustment. For speed updates, the velocity unit can have one of the following values wheter the mph is the default value: • mph (miles per hour) • kph (kilometers per hour)
velocityAdjustment	String	no	Specifies the speed update where you can define a change in the speed of the segment by specifying the change in velocity (unit and value). Speed values can be increased (positive value) or decreases (negative value). The default unit is mph (miles per hour) unless you specify the *velocityUnit* parameter.
velocityPercentage	Integer	no	Specifies the speed update where you define an increase in the speed of the segment by specifying a percentage to increase (positive value) or decrease (negative value) the speed.
roadType	String	no	Specifies the update where you can define the new road type of the segment.

Parameter	Type	Required	Description
version	String	No	Specifies the version of the `SegmentUpdates` REST service. Valid values are *1* and *2*.

The roadType can be one of the following:

- access way
- back road
- connector
- ferry
- footpath
- limited access dense urban
- limited access rural
- limited access suburban
- limited access urban
- local road dense urban
- local road rural
- local road suburban
- local road urban
- major local road dense urban
- major local road rural
- major local road suburban
- major local road urban
- major road dense urban
- major road rural
- major road suburban
- major road urban
- minor local road dense Urban
- minor local road rural
- minor local road suburban
- minor local road urban
- normal road dense urban
- normal road rural
- normal road rural
- normal road urban
- primary highway dense urban
- primary highway rural
- primary highway suburban
- primary highway urban
- ramp dense urban

- ramp limited access
- ramp major road
- ramp primary highway
- ramp rural
- ramp secondary highway
- ramp urban
- ramp suburban
- secondary highway dense urban
- secondary highway rural
- secondary highway suburban
- secondary highway urban

Reset Parameter

The HTTP DELETE operation takes the following query parameter.

Parameter	Type	Required	Description
resetType	String	no	Specifies whether to reset (undo) an update for a segment. Available options are: • **speed:** Reset the speed update for a specific segment. • **exclude:** Reset the exclude for a specific segment. • **roadType:** Reset the road type for a specific segment.

Examples

Exclude a segment (HTTP POST)

```
http://<server>:<port>/<webApp-context>/services/databases/<US_NE>/persistentUpdates/segments/
9f5c5a5a:5174e2.json?exclude=true
```

Return a list of updates for a single segment (HTTP GET)

```
http://<server>:<port>/<webApp-context>/services/databases/<US_NE>/persistentUpdates/segments/
efed6c1:a59ad5.json?velocityUnit=kph
```

Return a list of all segment updates for the US_NE routing database resource (HTTP GET)

```
http://<server>:<port>/<webApp-context>/services/databases/<US_NE>/persistentUpdates/
segments.json?segments=
```

Return a list of updates for the multiple segments (HTTP GET)

```
http://<server>:<port>/<webApp-context>/services/databases/<US_NE>/persistentUpdates/
segments.json?segments=27e20762:4718d9,7e3396fc:14c9c2c
```

Remove a segment speed persistent update (HTTP DELETE)

```
http://<server>:<port>/<webApp-context>/services/databases/<US_NE>/persistentUpdates/
segments?segmentID=9f5c5a5a:5174e2&resetType=speed
```

Road Type Updates

HTTP POST URL Format

The following format is used for HTTP POST requests. HTTP POST is used to set a persistent update to a road type.

```
HTTP POST:
/webApp-context/services/databases/dbsource/persistentUpdates/roadTypes/roadtype
.json?query_parameters
```

Where *dbsource* is the name of the database to update the route data, and *roadtype* is the type of road to update. Use the database name specified in the Database Resource tool.

HTTP GET URL Format

The following format is used for HTTP GET requests. HTTP GET is used to return a list of persistent updates for road types.

```
HTTP GET: /webApp-context/services/databases/dbsource/
persistentUpdates/roadTypes/road_type.json
                or
HTTP GET: /webApp-context/services/databases/dbsource/
persistentUpdates/roadTypes.json?roadTypes=road_type
```

Where *dbsource* is the name of the database to return to persistent updates from, and *roadtype* is the type of road return updates.

Note: The first format is used to return the persistent update for only one road type. The second format is used to return either multiple road types or all road types. For multiple road types, use a comma separated list of road types. For all road types, use an empty roadtypes= parameter. See examples below.

HTTP DELETE URL Format

The following format is used for HTTP DELETE requests. HTTP DELETE is used to remove a specific persistent update to a road type.

```
HTTP DELETE: /webApp-context/services/databases/
dbsource/persistentUpdates/roadTypes/roadtype
```

Where *dbsource* is the name of the database, and *roadtype* is the type of road that contains the persistent update to remove. Use the database name specified in the Database Resource tool.

The *roadtype* can be one of the following for both the HTTP POST and HTTP DELETE:

- access way
- back road
- connector
- ferry
- footpath
- limited access dense urban
- limited access rural
- limited access suburban
- limited access urban
- local road dense urban
- local road rural
- local road suburban
- local road urban
- major local road dense urban
- major local road rural
- major local road suburban
- major local road urban
- major road dense urban
- major road rural
- major road suburban
- major road urban
- minor local road dense Urban
- minor local road rural
- minor local road suburban
- minor local road urban
- normal road dense urban
- normal road rural

- normal road rural
- normal road urban
- primary highway dense urban
- primary highway rural
- primary highway suburban
- primary highway urban
- ramp dense urban
- ramp limited access
- ramp major road
- ramp primary highway
- ramp rural
- ramp secondary highway
- ramp urban
- ramp suburban
- secondary highway dense urban
- secondary highway rural
- secondary highway suburban
- secondary highway urban

Query Parameters

The HTTP POST operation takes the following query parameters.

Parameter	Type	Required	Description
velocity	String	no	This is a speed update where you can define the new speed of the road type by specifying the new velocity. The default unit is mph (miles per hour) unless you specify the *velocityUnit* parameter.
velocityUnit	String	no	This is a unit of speed for the velocity or velocityAdjustment. For speed updates, the velocity unit can have one of the following values where mph is the default value: • mph (miles per hour) • kph (kilometers per hour)

Parameter	Type	Required	Description
velocityAdjustment	String	no	Specifies the speed update where you can define a change in the speed of the road type by specifying the change in velocity (unit and value). Speed values can be increased (positive value) or decreases (negative value). The default unit is mph (miles per hour) unless you specify the *velocityUnit* parameter.
velocityPercentage	Integer	no	Specifies the speed update where you can define an increase in the speed of the road type by specifying a percentage to increase (positive value) or decrease (negative value) the speed.
version	String	No	Specifies the version of the `RoadTypePointUpdates` REST service. Valid values are *1* and *2*.

Examples

Set a new velocity of a road type (HTTP POST)

```
http://<server>:<port>/<webApp-context>/services/databases/<usroutedatabase>/persistentUpdates/roadTypes/
ferry.json?velocity=5&velocityUnits=mph
```

Return a list of updates for the ferry road type (HTTP GET)

```
http://<server>:<port>/<webApp-context>/services/databases/<US_NE>/persistentUpdates/roadTypes/
ferry.json?velocityUnit=kph
```

Return a list of all road type updates for the US_NE routing database resource (HTTP GET)

```
http://<server>:<port>/<webApp-context>/services/databases/<US_NE>/persistentUpdates/
roadTypes.json?roadTypes=
```

Return a list of updates for the ferry, connector, and normal road urban road types (HTTP GET)

```
http://<server>:<port>/<webApp-context>/services/databases/<US_NE>/persistentUpdates/
roadTypes.json?roadTypes=ferry,connector,normal road urban
```

Remove a road type persistent update (HTTP DELETE)

```
http://<server>:<port>/<webApp-context>/services/databases/<US_NE>/persistentUpdates/roadTypes/back
 road
```

Remove All Updates

HTTP DELETE URL Format

The following format is used for HTTP DELETE requests. HTTP DELETE is used to remove all persistent update for a specified database.

```
HTTP DELETE: /webApp-context/services/databases/
dbsource/persistentUpdates
```

Where *dbsource* is the name of the database that contains the persistent updates to remove. Use the database name specified in the Database Resource tool.

Example

Removes all persistent updates for the US_NE routing database resource.

```
http://<server>:<port>/<webApp-context>/services/databases/<US_NE>/persistentUpdates
```

Get All Updates

HTTP GET URL Format

The following format is used for HTTP GET requests. This HTTP GET operation is used to list all the persistent updates for a specified routing database resource.

```
HTTP GET: /webApp-context/services/databases/
dbsource/persistentUpdates.json
```

Where *dbsource* is the name of the database that contains the persistent updates to remove. Use the database name specified in the Database Resource tool.

Query Parameters

This operation takes the following query parameter:

Parameter	Type	Required	Description
velocityUnit	String	no	Specifies that the updates saved in the server will be returned in this specified unit. If this parameter is not mentioned, response will be returned in default unit. For speed updates, the velocity unit can have one of the following values where mph is the default value: • mph (miles per hour) • kph (kilometers per hour) • mtps (meters per second) • and mtpm (meters per minute)
version	String	No	Specifies the version of the `RoadTypePointUpdates` REST service. Valid values are *1* and *2*.

Example

Return a list of updates for the US_NE routing database resource.

```
http://<server>:<port>/<webApp-context>/services/databases/<US_NE>/persistentUpdates.json
```

Response

```
{
 "roadTypeUpdates":
 [{
   "roadType": "major road dense urban",
   "speed": {
    "velocity": 90,
    "velocityUnit": "MPH"
   }
  }
 ],
 "segmentUpdates":
 [{
   "exclude": true,
   "roadType": "major road dense urban",
   "segmentID": "c75994cc:12d916",
   "speed": {
    "velocity": 65,
    "velocityUnit": "MPH"
   }
  }, {
   "exclude": true,
   "roadType": "major road dense urban",
   "segmentID": "7ac5401f:6b1bf7",
   "speed": {
    "velocity": 65,
    "velocityUnit": "MPH"
   }
  }
 ]
}
```

When velocity unit parameter is specified in kph.

```
http://<server>:<port>/<webApp-context>/services/databases/<database_name>/persistentUpdates.json?velocityUnit=kph
```

Response

```
{
 "roadTypeUpdates": [{
   "roadType": "major road dense urban",
   "speed": {
    "velocity": 145,
    "velocityUnit": "KPH"
   }
  }
 ]
}
```

Transient Updates

Transient Updates

It allows you to set transient updates (point, segment, road type updates) for each request. For instance, you can request that the server attempt to avoid all of the major road types. Each request can contain one or more updates. For speed updates, positive speed value is a speed increase and negative speed value is a speed decrease.

The Transient Updates service allows a user to override aspects of the network. The overrides can be done on a per-road type, at a specific point or a specific segment. The Transient Update is valid only for a specific data source and may not be valid after a data update.

Using the Transient Updates, you can perform the following:

- Exclude a point
- Exclude a segment
- Set the speed of a point, segment, or road type
- Change (increase or decrease) the speed of a point, segment, or road type by a value
- Change (increase or decrease) the speed of a point, segment, or road type by a percentage

The following is a description of the transient update types:

point

Point updates are changes applied to a corresponding point (X, Y). For a particular point, you can exclude the point, set the speed of the point, or change (increase or decrease) the speed of the point by a value or percentage. Use one of the following types of updates:

Point Update Type	Description
percentage	Specifies the speed update where you can replace the speed of the point by specifying a percentage to increase (positive value) or decrease (negative value) the speed.
speed	Specifies the speed update where you can define the new speed of the point by specifying the new velocity. For speed updates, the velocity can have one of the following values: • kph (kilometers per hour) • mph (miles per hour) • mtps (meters per second) • mtpm (meters per minute)
speedAdjustment	Specifies a speed update where you can define a change in the speed of the point by specifying the change in the value. The speed velocity can be increased (positive value) or decreased (negative value). For speed updates, the velocity can have one of the following values: • kph (kilometers per hour) • mph (miles per hour) • mtps (meters per second) • mtpm (meters per minute)
exclude	Specifies a value to exclude the specified point from the route calculation. To exclude a point you need to specify the point and include the exclude parameter defined as Y. Valid values are Y (yes) and N (no). This is a String update type.

SegmentID

Specifies the segment updates applied to a corresponding segment ID. For a particular segment, you can exclude the segment, set the speed of the segment, or change (increase or decrease) the speed of the segment by a value or percentage. Use one of the following types of updates:

SegmentID Update Type	Description
percentage	Specifies a speed update where you can define an increase in the speed of the segmentID by specifying a percentage to increase (positive value) or decrease (negative value) the speed.
speed	Specifies a speed update where you define the new speed of the segmentID by specifying the new velocity. For speed updates, the velocity can have one of the following values: • kph (kilometers per hour) • mph (miles per hour) • mtps meters per second) • mtpm (meters per minute)
speedAdjustment	Specifies the speed update where you can define a change in the speed of the segmentID by specifying the change in velocity. Speed values can be increased (positive value) or decreased (negative value). For speed updates, the velocity can have one of the following values: • kph (kilometers per hour) • mph (miles per hour) • mtps (meters per second) • mtpm (meters per minute)
exclude	Specifies the value to exclude the specified segmentID from the route calculation. To exclude a segmentID you need to specify the segmentID and include the exclude parameter defined as Y. Valid values are Y (yes) and N (no). This is a String update type.
roadType	Specifies a value to change the road type for the segment for the route calculation. See the list of roadType options below.

roadType

Road type updates are changes applied to a corresponding road type. For a particular road type, you can set the speed of the roadtype or change (increase or decrease) the speed of the road type by a value or percentage. Use one of the following types of updates:

roadType Update Type	Description
percentage	Specifies the speed update where you can define an increase in the speed of the road type by specifying a percentage to increase (positive value) or decrease (negative value) the speed.

roadType Update Type	Description
speed	Specifies the speed update where you can define the new speed of the road type by specifying the new velocity. For speed updates, the velocity unit can have one of the following values: • kph (kilometers per hour) • mph (miles per hour) • mtps (meters per second) • mtpm (meters per minute)
speedAdjustment	Specifies the speed update where you can define a change in the speed of the road type by specifying the change in velocity. Speed values can be increased (positive value) or decreased (negative value). For speed updates, the velocity can have one of the following values: • kph (kilometers per hour) • mph (miles per hour) • mtps (meters per second) • mtpm (meters per minute)

The roadType can be one of the following:

- access way
- back road
- connector
- ferry
- footpath
- limited access dense urban
- limited access rural
- limited access suburban
- limited access urban
- local road dense urban
- local road rural
- local road suburban
- local road urban
- major local road dense urban
- major local road rural
- major local road suburban
- major local road urban
- major road dense urban
- major road rural
- major road suburban
- major road urban

- minor local road dense Urban
- minor local road rural
- minor local road suburban
- minor local road urban
- normal road dense urban
- normal road rural
- normal road rural
- normal road urban
- primary highway dense urban
- primary highway rural
- primary highway suburban
- primary highway urban
- ramp dense urban
- ramp limited access
- ramp major road
- ramp primary highway
- ramp rural
- ramp secondary highway
- ramp urban
- ramp suburban
- secondary highway dense urban
- secondary highway rural
- secondary highway suburban
- secondary highway urban

Examples

Example of transient update HTTP POST payload

```
{
 "transientUpdates": [{
   "segmentID": "specify a segment id",
   "updates": [{
     "exclude": "Y"
    }, {
     "roadType": "a road type"
    }, {
     "percentage": 26.0
    }, {
     "speed": {
      "velocity": 25,
      "velocityUnit": "kph"
     }
    }, {
     "percentage": -13.5
    }, {
     "speedAdjustment": {
      "velocity": 5,
      "velocityUnit": "kph"
     }
    }
   ]
  }, {
   "point": "specify a geojson point",
   "updates": [{
     "exclude": "Y"
    }, {
     "percentage": 26.0
    }, {
     "speed": {
      "velocity": 25,
      "velocityUnit": "kph"
     }
    }, {
     "percentage": -13.5
    }, {
     "speedAdjustment": {
      "velocity": 5,
      "velocityUnit": "kph"
     }
    }
   ]
  }, {
   "roadType": "specify a road type",
   "updates": [{
     "percentage": 26.0
    }, {
     "speed": {
      "velocity": 25,
      "velocityUnit": "kph"
     }
    }, {
     "percentage": -13.5
    }, {
     "speedAdjustment": {
      "velocity": 5,
      "velocityUnit": "kph"
     }
    }
   ]
  }
 ]
}
```

Example of geojson point

```
"point": {
 "type": "Point",
 "crs": {
  "type": "name",
  "properties": {
   "name": "epsg:4326"
  }
 },
 "coordinates":
 [-73.979102, 40.785193]
}
```

Version specific error response

When you enter an invalid parameter value (for example, json missing) in a request, the error response you get depends on the version entered by you. When the version is 1, you get value and error whereas when the version is 2, the response only contains the error.

- Request when version is *1*:

```
http://server:port/webApp-context/services/databases/usroutedatabase.json?
startPoint=-73.972033%2C40.794928%2Cepsg%3A4326&distanceUnit=mi
&endPoint=-73.985617%2C40.747%2Cepsg%3A4326&q=route&version=1
```

- Response:

```
{
   "value": "Invalid POST payload specified: JSON is not valid.",
   "errors": [
{
      "errorCode": 4182,
     "userMessage": "Invalid POST payload specified: JSON is not valid."

                                }
]
}
```

- Request when version is *2*:

```
http://server:port/webApp-context/services/databases/usroutedatabase.json?
startPoint=-73.972033%2C40.794928%2Cepsg%3A4326&distanceUnit=mi
&endPoint=-73.985617%2C40.747%2Cepsg%3A4326&q=route&version=2
```

- Response:

```
{
"errors": [
{
   "errorCode": 4182,
   "userMessage": "Invalid POST payload specified: JSON is not valid."
}
]
}
```

GetCapabilities

Description

The `GetCapabilities` service enables user to get metadata about the routing engine deployed. This metadata allows users to explore a service and its capabilities, therefore optimizing their experience using the routing services.

This is available as REST service only.

HTTP GET URL Format

```
http://<server>:<port>/webApp-context/services/v1/capabilities.json
```

Query Parameters

Parameter	Required	Description
acceptVersions	Optional	Placeholder (not functional)
sections	Optional When omitted, returns information about all sections.	Comma-separated unordered list of zero or more names of sections of service metadata document to be returned in the service metadata document. Section values are case-insensitive. Accepted section values are ServiceIdentification, ServiceProvider, operationsMetadata and databases.

Response

The response will be in line with OGC GetCapabilities. It is in JSON format and has these sections:

- serviceIdentification
- serviceProvider
- operationsMetadata
- databases

serviceIdentification

This section contains basic metadata about this specific server. Its content will look as follows:

```
   "serviceIdentification":
       {
           "title": "Routing Service",
           "abstract": "Routing service maintained by Precisely",
           "keywords":
           {
               "keyword":
               [
               ]
           },
           "serviceType": "Routing",
           "serviceTypeVersion": "v1",
           "fees": "none",
           "accessConstraints": "none"
       }
```

This information will be same as what is available in the `getCapabilities.json` configuration file.

This file is present in *`SpectrumDirectory`*`\server\modules\routing`. The server needs to be restarted for any change made to the file to have an effect. The administrator determines which information the user should get and can modify or delete corresponding entries in the JSON file. All fields in the JSON file are optional.

serviceProvider

This section contains metadata about the organization operating this server. Its content will look like as follows:

```
"serviceProvider":
      {
          "providerName": "Routing Service Provider",
          "providerSite":
          {
              "href": "http://www.yourcompany.com/",
              "type": "simple"
          },
          "serviceContact":
          {
              "contactInfo":
              {
                  "address":
                  {
                      "administrativeArea": "Province",
                      "city": "City",
                      "country": "Country",
                      "deliveryPoint": "Mail Delivery Location",

"electronicMailaddress":"mailto://support@yourcompany.com",
                      "postalCode": "PostCode"
                  },
                  "contactInstructions": "Contact Instructions",
                  "hoursOfservice": "24 Hours",
                  "phone":
                  {
                      "facsimile": "1.800.000.0000",
                      "voice": "1.800.000.0000"
                  }
              },
              "individualName": "Contact Person",
              "positionName": "Contact Person's Title",
              "role": "Contact Person's Role"
          }
      }
```

This will also be configured using the `getCapabilities.json` configuration file as described above.

operationsMetadata

This section contains metadata about the operations implemented by this server, including the URLs for operation requests. These fixed operations or services are listed in this section:

- **GetRoute**: point to point service
- **GetRouteCostMatrix**: matrix of points processing service
- **GetTravelBoundary** : generates a drive or walk time or distance boundary
- **DescribeDatasets**: gives information about the datasets configured
- **DescribeDatabases**: gives information about all the databases configured
- **GetSegmentDataForPoint**: returns segment information for a point
- **GetSegmentDataForSegment**: returns segment information for a segment ID
- **ListPersistentUpdates**: lists down all the persistent updates that exists in the server
- **DeletePersistentUpdates**: deletes all the persistent updates that exists in the server
- **SetPersistentUpdatesAtPoint**: saves persistent update for the specified point in the server
- **SetPersistentUpdatesForSegment**: saves persistent update for the specified segment ID in the server
- **SetPersistentUpdatesForRoadType**: saves persistent update for the specified road type in the server

Its content will look like as follows:

```
{"operationsMetadata": [{
  "name": "GetRoute",
  "DCP": {
   "HTTP": {
    "GET":
"<schema>://<server>:<port>/rest/Spatial/erm/databases/<DB_NAME>.json?q=route&version=2",

    "POST":
"<schema>://<server>:<port>/rest/Spatial/erm/databases/<DB_NAME>.json?q=route&version=2"
```

```
    }
   },
   "parameter": {
    "name": "OutputFormat",
    "value": "text/json"
   }
  },

  {
   "name": "GetRouteCostMatrix",
   "DCP": {
    "HTTP": {
     "GET":
"<schema>://<server>:<port>/rest/Spatial/erm/databases/<DB_NAME>.json?q=routeCostMatrix&version=2",

     "POST":
"<schema>://<server>:<port>/rest/Spatial/erm/databases/<DB_NAME>.json?q=routeCostMatrix&version=2"

    }
   },"parameter": {
    "name": "OutputFormat",
    "value": "text/json"}

  },

  {

   "name": "GetTravelBoundary",
   "DCP": {
    "HTTP": {
     "GET":
"<schema>://<server>:<port>/rest/Spatial/erm/databases/<DB_NAME>.json?q=travelBoundary&version=2",

     "POST":
"<schema>://<server>:<port>/rest/Spatial/erm/databases/<DB_NAME>.json?q=travelBoundary&version=2"

    }
   },
   "parameter": {
    "name": "OutputFormat",
    "value": "text/json"
   }
  },

  {
   "name": "DescribeDatasets",
   "DCP": {
    "HTTP": {
     "GET": "<schema>://<server>:<port>/rest/Spatial/erm/v1/datasets.json"

    }
   },
   "parameter": {
    "name": "OutputFormat",
    "value": "text/json"
   }
  },

  {
   "name": "DescribeDatabases",
   "DCP": {
    "HTTP": {
     "GET": "<schema>://<server>:<port>/rest/Spatial/erm/v1/databases.json"

    }
   },
   "parameter": {
    "name": "OutputFormat",
    "value": "text/json"
   }
  },

  {
   "name": "GetSegmentDataForPoint",
   "DCP": {
    "HTTP": {
     "GET":
"<schema>://<server>:<port>/rest/Spatial/erm/databases/<DB_NAME>/segments.json"


    }
   },
   "parameter": {
    "name": "OutputFormat",
    "value": "text/json"
   }
  },

  {
   "name": "GetSegmentDataForSegment",
   "DCP": {
    "HTTP": {
     "GET":
"<schema>://<server>:<port>/rest/Spatial/erm/databases/<DB_NAME>/segments/<segmentID>.json"

    }
   },
   "parameter": {
    "name": "OutputFormat",
    "value": "text/json"
   }
  },

  {
   "name": "ListPersistentUpdates",
   "DCP": {
    "HTTP": {
     "GET":
"<schema>://<server>:<port>/rest/Spatial/erm/databases/<DB_NAME>/persistentUpdates.json"

    }
   },
   "parameter": {
    "name": "OutputFormat",
    "value": "text/json"
   }
  },

  {
   "name": "DeletePersistentUpdates",
   "DCP": {
    "HTTP": {
     "DELETE":
"<schema>://<server>:<port>/rest/Spatial/erm/databases/<DB_NAME>/persistentUpdates"

    }
   },
   "parameter": {
    "name": "OutputFormat",
    "value": "text/json"
   }
  },

  {
   "name": "SetPersistentUpdatesAtPoint",
   "DCP": {
    "HTTP": {
     "POST":
"<schema>://<server>:<port>/rest/Spatial/erm/databases/<DB_NAME>/persistentUpdates.jason"

    }
   },
   "parameter": {
    "name": "OutputFormat",
    "value": "text/json"
   }
  },

  {
   "name": "SetPersistentUpdatesForSegment",
   "DCP": {
    "HTTP": {
     "POST":
"<schema>://<server>:<port>/rest/Spatial/erm/databases/<DB_NAME>/persistentUpdates/segments/<segmentID>.json"

    }
   },
   "parameter": {
    "name": "OutputFormat",
    "value": "text/json"
   }
  },


  {
   "name": "SetPersistentUpdatesForRoadType",
   "DCP": {
    "HTTP": {
     "POST":
"<schema>://<server>:<port>/rest/Spatial/erm/databases/<DB_NAME>/persistentUpdates/roadTypes/<roadtype>.json"

    }
   },
   "parameter": {
    "name": "OutputFormat",
    "value": "text/json"
   }
  }

]
}
```

databases

This section will contain the list of names of databases which are configured in the server.

For example:

```
{
 "databases":
 [
 "US_NE",
 "US"
 ]}
```

If no database is configured on the server, this is returned:

```
{
 "databases":[
]
}
```

DescribeDatasets

Description

The `DescribeDatasets` service enables user to get information metadata about the datasets corresponding to the routing databases added to the GR SDK server. The response will be analogous with the metadata information present in the dataset path.

This feature is available as REST service only.

HTTP GET URL Format (All Datasets)

```
http://<server>:<port>/webApp-context/services/v1/datasets.json
```

HTTP GET URL Format(Single Dataset)

```
http://<server>:<port>/webApp-context/services/v1/datasets/<dataset_ID>.json
```

The *dataset_ID* is the 'id' corresponding to the elements in the 'dataSets' array from the DescribeDatabases service.

Response

The response of this service is a JSON array.

For all datasets, the length of the JSON array is same as the total number of the dataset paths (with metadata available) added against the databases configured in Management Console. If a dataset path does not have metadata available, that entry will be ignored.

For a single dataset, the length of the JSON array will be one if and only if the metadata is available in the dataset path. Otherwise, an empty JSON array will be returned.

Example

Two routing databases have been added in the GR SDK. The name and dataset paths of the databases are as follows:

1. **US_NE**: `E:\\db\\ERM-US\\2014.09\\driving\\northeast`
2. **US**: `E:\\db\\ERM-US\\2014.09\\driving\\midwest` and `E:\\db\\ERM-US\\2014.09\\driving\\south`

Sample Request (All Datasets):

```
http://<server>:<port>/webApp-context/services/v1/datasets.json
```

Response:

```
{
  "dataSets": [{
    "component": "routing",
    "description": "USA Test dataset",
    "ext": {
      "bbox": [68.291015625, 7.9721977144, 97.55859375, 35.4606699515],

      "crs": "epsg:4326",
      "cvr": true,
      "historicTrafficTimeBuckets": {
        "amPeak": {
          "lowerBound": 700,
          "upperBound": 1000
        },
        "nightTime": {
          "lowerBound": 2200,
          "upperBound": 400
        },
        "offPeak": {
          "lowerBound": 1000,
          "upperBound": 1600
        },
        "pmPeak": {
          "lowerBound": 1600,
          "upperBound": 1900
        }
      },
      "locale": "EN",
      "type": "driving"
    },
    "id": "US dataset",
    "name": "USA",
    "product": "Spatial",
    "vintage": "September 2015"
  }]
}
```

Sample Request (Single Dataset):

```
http://<server>:<port>/webApp-context/services/v1/datasets/US%20dataset.json
```

Response:

```
{
  "dataSets": [{
    "component": "routing",
    "description": "USA Test dataset",
    "ext": {
      "bbox": [68.291015625, 7.9721977144, 97.55859375, 35.4606699515],
      "crs": "epsg:4326",
      "cvr": true,
      "historicTrafficTimeBuckets": {
        "amPeak": {
          "lowerBound": 700,
          "upperBound": 1000
        },
        "nightTime": {
          "lowerBound": 2200,
          "upperBound": 400
        },
        "offPeak": {
          "lowerBound": 1000,
          "upperBound": 1600
        },
        "pmPeak": {
          "lowerBound": 1600,
          "upperBound": 1900
        }
      },
      "locale": "EN",
      "type": "driving"
    },
    "id": "US dataset",
    "name": "USA",
    "product": "Spatial",
    "vintage": "September 2015"
  }]
}
```

DescribeDatabases

Description

The `DescribeDatabases` operation returns name of all the database resources that are configured in the system and can be used in a request. This operation returns a list containing the names of all databases in the system and an array containing the datasets for each database.

HTTP GET URL Format (All Databases)

The format below is used for HTTP GET requests. If no data resource exists on the server, an empty list is returned.

```
http://<server>:<port>/webApp-context/services/v1/databases.json
```

Example (All Databases)

Request:

```
http://<server>:<port>/webApp-context/services/v1/databases.json
```

Response:

```
{
 "databases":
 [
  {
   "dataSets":
   [
    "US_Central"
   ],
   "name": "US_CN"
  },
  {
   "dataSets":
   [
    "US_NorthEast"
   ],
   "name": "US_NE"
  },
  {
   "dataSets":
   [
    "US_Central",
    "US_Midwest",
    "US_NorthEast",
    "US_Pacific",
    "US_South"
   ],
   "name": "US"
  }
 ]
}
```

HTTP GET URL Format (Single Database)

The format below is used for HTTP GET requests. This request is used if to get the dataset information for a particular data resource. If no data resource with the specified name exists on the server, an exception is returned.

```
http://<server>:<port>/webApp-context/services/v1/<database_name>.json
```

Example (Single Database)

Request:

```
http://<server>:<port>/webApp-context/services/v1/databases/US.json
```

Response:

```
{
 "databases":
 [
  {
   "dataSets":
   [
    "US_Central",
    "US_Midwest",
    "US_NorthEast",
    "US_Pacific",
    "US_South"
   ],
   "name": "US"
  }
 ]
}
```

4 - Closest Arc Snapping

While calculating route, boundary, or matrix for a given segment, if the start or end points lie off the road network, the ERM attempts to find the closest segment near the start and end points. In this extended calculation, the ERM considers the geometrically closest segments. In this approach, if the closest segment is restricted for routing, an error is received -- “Path could not be calculated” in case of the route. The same error is received when the matrix and circular boundary is calculated according to *maxOffRoadDistance* for the polygon.

The following solution attempts to improve the find closest segment snapping logic that ignores the closest restricted arcs and provides you the next-best-possible route.

> **Note:** This change is applicable and effective only in scenarios where the closest arcs near start or end points are restricted.

In this section

What is a Restricted Arc

A restricted arc is not traversable which means; you cannot find a route through a restricted arc. In case of closed premises, an arc which connects the premises with the outside network may have a gated entry and hence is restricted for driving.

For example – The start point in the following image lies close to a segment which is restricted for traveling.

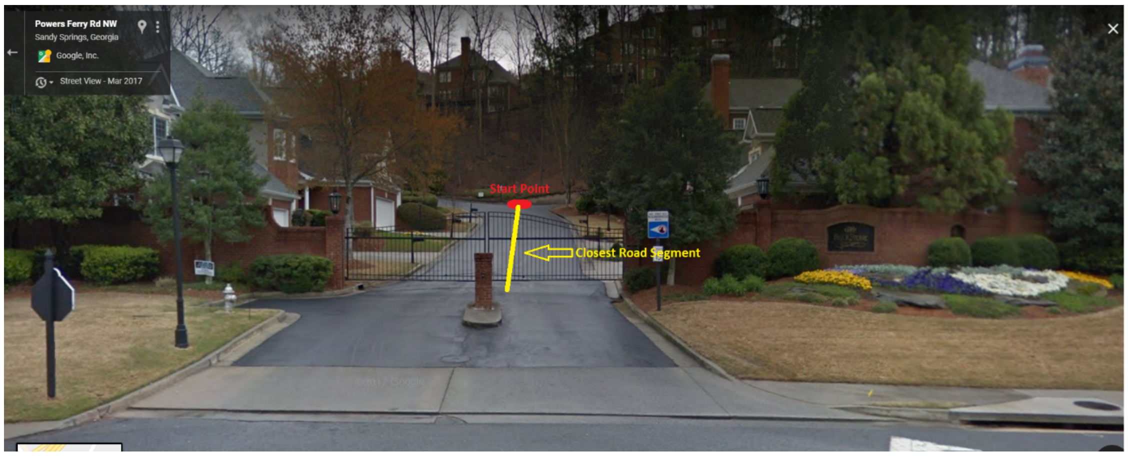
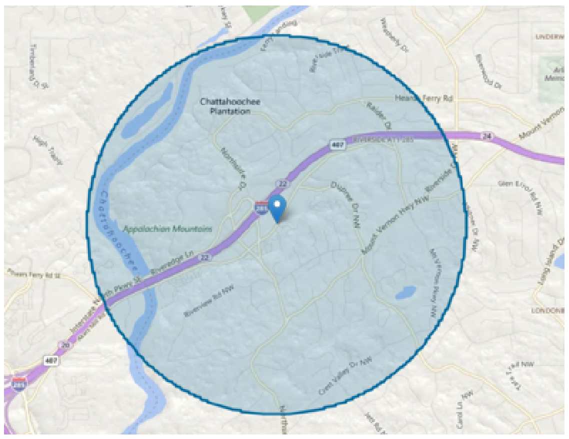
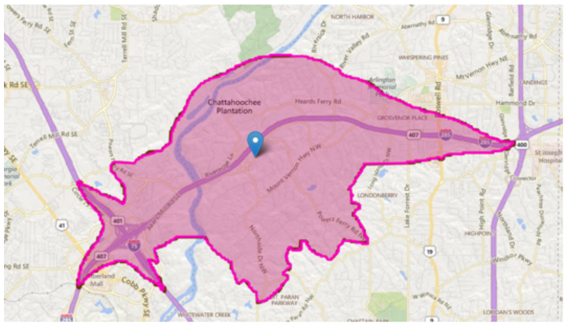


Table 1: Comparing Boundaries

Before	After
	

Impact on Boundary Calculations

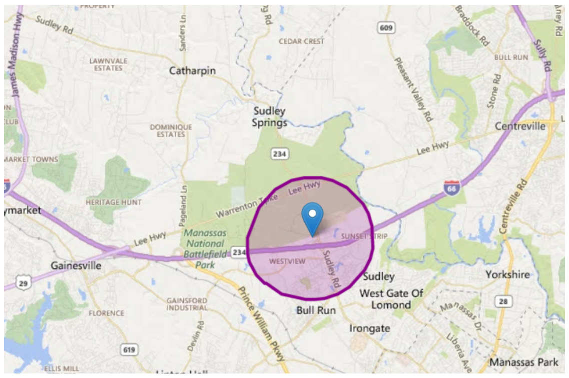
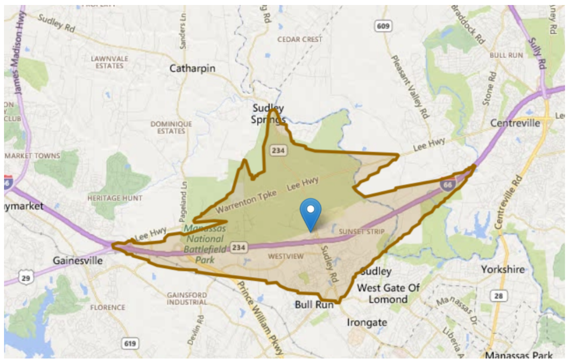
In some cases, user finds circular boundaries due to the presence of a restricted arc near the points.

During boundary calculation, if the closest road segment is restricted, then the boundary is calculated using the *maxOffRoadDistance* parameter. With the new improvements to the algorithm, if the road segment closest to the point is restricted, then that segment is avoided, and the boundary is calculated with open segments.

The following snapshots depict the situation before and after the snapping logic implementation.

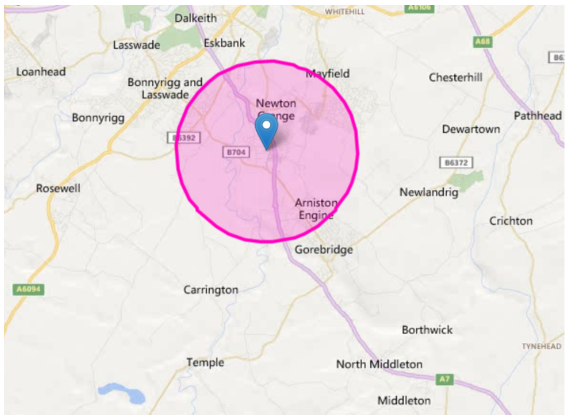
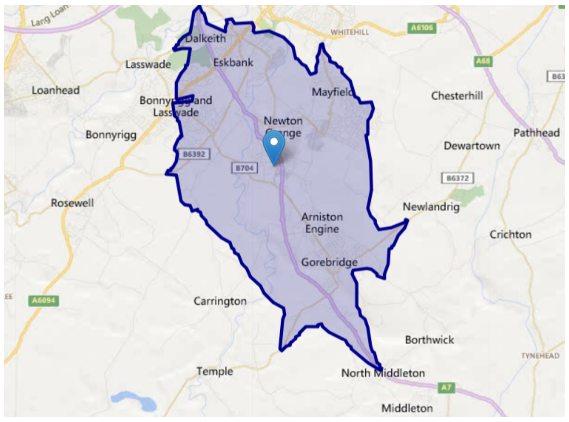
Point: -77.523203, 38.803707 Cost: 5 minutes

Table 2: Comparing Boundaries

Before	After
	

Point: -3.0681250, 55.8612868 **Cost:** 5 minutes

Table 3: Comparing Boundaries

Before	After
	

Impact on Route and Matrix Calculations

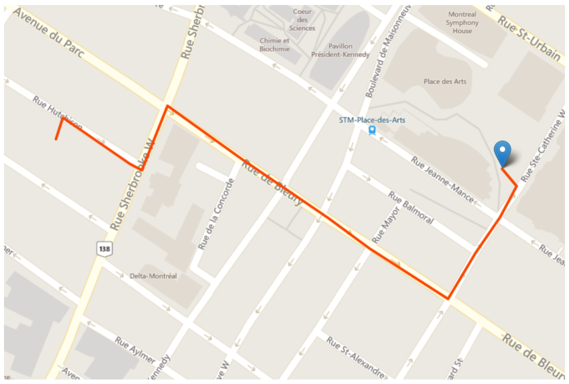
In some cases, the user receives an error - “Path could not be calculated.” This error is caused by restricted closest arcs. The improved snapping logic helps in reducing such errors by avoiding restricted arcs.

The improved logic takes the closest arc into consideration and attempts to ignore such arcs thus reducing errors. For example, if there is any restricted road near the start or end point, the new snapping logic ignores the restricted path and finds another way to calculate the route matrix.

With the reduced errors caused by restricted arcs, the performance of route and matrix calculations increases as the routing algorithm has less number of path complexities to handle.

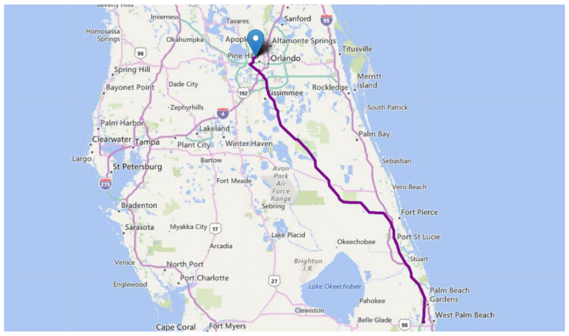
startpoint: -73.5661, 45.5077 **endPoint:** -73.576048, 45.496936

Table 4: Comparing Boundaries

Before	After
“Path could not be calculated” error	

startPoint: -80.146276, 26.707754 **endPoint:** -81.483591, 28.583825

Table 5: Comparing Boundaries

Before	After
“Path could not be calculated” error	

5 - Local Roads Load Factor

The localRoadsLoadFactor is a parameter that allows you to control the number of roads you can include while calculating a route or matrix. It assumes values ranging between 1 and 3.

In this section

Why is it Needed?

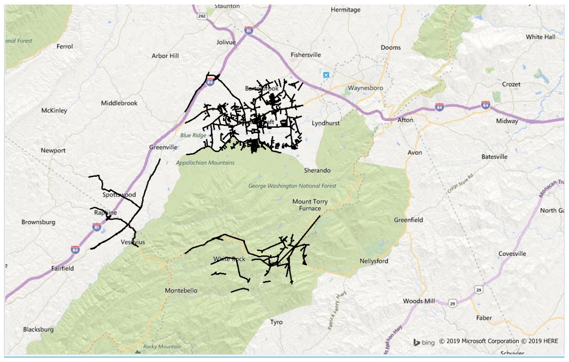
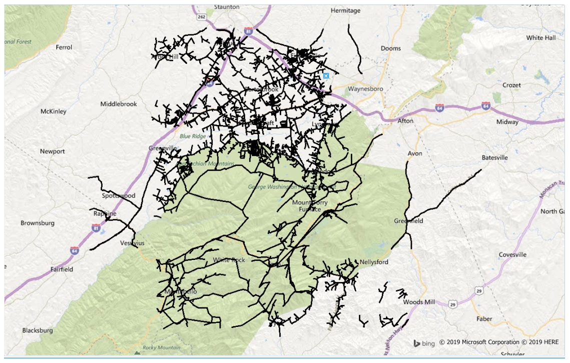
The localRoadsLoadFactor addresses a rare scenario where the routing engine could have calculated a shorter route. This could happen due to the current functionality of the routing engine that loads minor roads in memory only at the time of the calculation.

In such cases, you can use the localRoadsLoadFactor parameter to optimize the results by entering a higher value for this parameter and load more minor roads for calculation.

The number of minor roads loaded in memory is directly proportional to the value of localRoadsLoadFactor parameter. This means that a bigger value (3, for example) signifies more minor roads will be loaded in memory which may result in a more optimized route.

Note: Despite using the localRoadsLoadFactor with higher value, you may or may not achieve a better route or matrix. This parameter only loads more minor roads and therefore increases the probability of a better calculation.

Table 6: Impact of smaller and bigger values

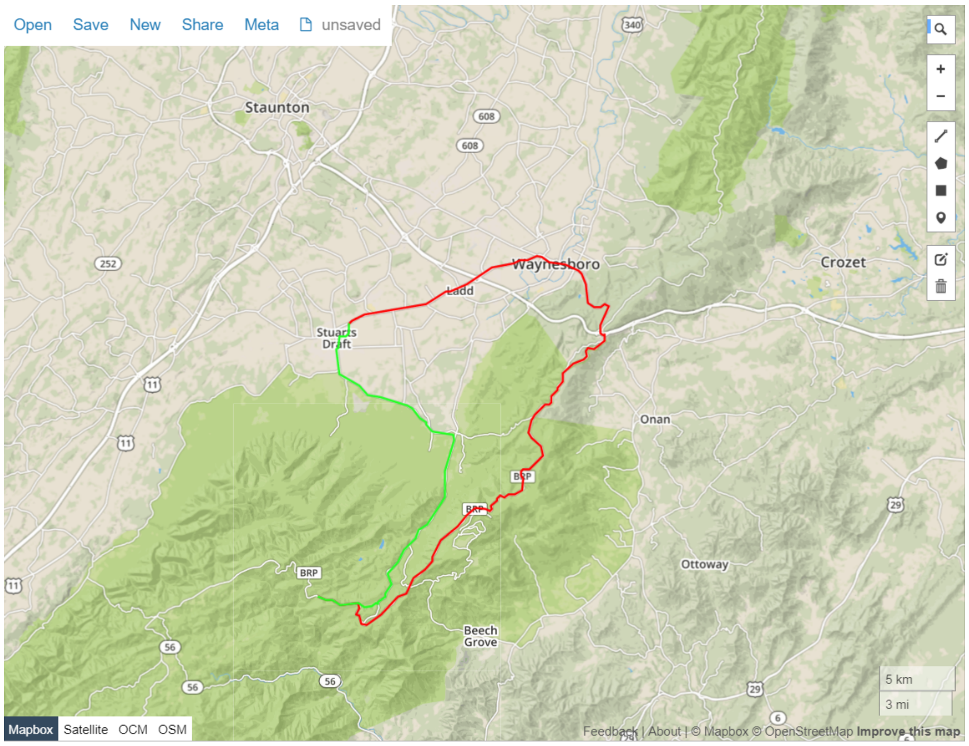
When localRoadsLoadFactor is 1	When localRoadsLoadFactor is 3
	

Note: As the comparison above depicts the density of minor roads loaded in memory based on the respective values of localRoadsLoadFactor.

Impact on Routes and Matrices

For Route and Matrix calculations, minor roads are loaded in memory, but in some scenarios, due to a smaller number of minor roads loaded in memory, the route calculation may result in longer routes. If the user feels that the route returned from the engine is not optimized, they can use the localRoadsLoadFactor to load more minor roads in memory to improve/optimize the route or matrix results.

The following example depicts two routes: one with a lower value and other with the highest value of the parameter:



Note: The brown route was calculated with localRoadsLoadFactor=1 while the orange route with localRoadsLoadFactor=3 is much shorter.

Impact on Performance

The use of this factor may impact the performance of the engine, so it is highly recommended to use only in cases where it is indispensable. The general observation is that there can be a performance degradation of about 50-60 percent on an average in the throughput.

6 - Sample Applications

Included with the GR SDK are three Java API sample applications demonstrating the three routing capabilities (GetRoute, GetTravelBoundary, and GetRouteCostMatrix), and one REST API sample application demonstrating the GetRoute capability. These samples all have executable files, as well as sample data included (Washington DC dataset). Refer the readme.txt file for detailed instructions on how to setup and run the samples. The readme.txt is located at: `install_dir\samples`.

The source for the samples are located at: `install_dir\samples\src\com\pb\routing\gra\samples\`.

In this section

7 - Appendix

In this section

Java API Road Type Enumeration

When defining roadtypes for ambient speeds, the following road type enumerations can be used:

- RoadType.ACCESS_WAY
- RoadType.BACKROAD
- RoadType.CONNECTOR
- RoadType.FERRY
- RoadType.FOOTPATH
- RoadType.LIMITED_ACCESS_DENSE_URBAN
- RoadType.LIMITED_ACCESS_RURAL
- RoadType.LIMITED_ACCESS_SUBURBAN
- RoadType.LIMITED_ACCESS_URBAN
- RoadType.LOCAL_ROAD_DENSE_URBAN
- RoadType.LOCAL_ROAD_RURAL
- RoadType.LOCAL_ROAD_SUBURBAN
- RoadType.LOCAL_ROAD_URBAN
- RoadType.MAJOR_LOCAL_ROAD_DENSE_URBAN
- RoadType.MAJOR_LOCAL_ROAD_RURAL
- RoadType.MAJOR_LOCAL_ROAD_SUBURBAN
- RoadType.MAJOR_LOCAL_ROAD_URBAN
- RoadType.MAJOR_ROAD_DENSE_URBAN
- RoadType.MAJOR_ROAD_RURAL
- RoadType.MAJOR_ROAD_SUBURBAN
- RoadType.MAJOR_ROAD_URBAN
- RoadType.MINOR_LOCAL_ROAD_DENSE_URBAN
- RoadType.MINOR_LOCAL_ROAD_RURAL
- RoadType.MINOR_LOCAL_ROAD_SUBURBAN
- RoadType.MINOR_LOCAL_ROAD_URBAN
- RoadType.NORMAL_ROAD_DENSE_URBAN
- RoadType.NORMAL_ROAD_RURAL
- RoadType.NORMAL_ROAD_URBAN
- RoadType.PRIMARY_HIGHWAY_DENSE_URBAN
- RoadType.PRIMARY_HIGHWAY_RURAL
- RoadType.PRIMARY_HIGHWAY_SUBURBAN
- RoadType.PRIMARY_HIGHWAY_URBAN
- RoadType.RAMP_DENSE_URBAN
- RoadType.RAMP_LIMITED_ACCESS

- RoadType.RAMP_MAJOR_ROAD
- RoadType.RAMP_PRIMARY_HIGHWAY
- RoadType.RAMP_RURAL
- RoadType.RAMP_SECONDARY_HIGHWAY
- RoadType.RAMP_URBAN
- RoadType.RAMP_SUBURBAN
- RoadType.SECONDARY_HIGHWAY_DENSE_URBAN
- RoadType.SECONDARY_HIGHWAY_RURAL
- RoadType.SECONDARY_HIGHWAY_SUBURBAN
- RoadType.SECONDARY_HIGHWAY_URBAN

REST API Road Type Enumeration

When defining roadtypes for ambient speeds, the following road type enumerations can be used:

- access way
- back road
- connector
- ferry
- footpath
- limited access dense urban
- limited access rural
- limited access suburban
- limited access urban
- local road dense urban
- local road rural
- local road suburban
- local road urban
- major local road dense urban
- major local road rural
- major local road suburban
- major local road urban
- major road dense urban
- major road rural
- major road suburban
- major road urban
- minor local road dense Urban
- minor local road rural
- minor local road suburban

- minor local road urban
- normal road dense urban
- normal road rural
- normal road rural
- normal road urban
- primary highway dense urban
- primary highway rural
- primary highway suburban
- primary highway urban
- ramp dense urban
- ramp limited access
- ramp major road
- ramp primary highway
- ramp rural
- ramp secondary highway
- ramp urban
- ramp suburban
- secondary highway dense urban
- secondary highway rural
- secondary highway suburban
- secondary highway urban

Java API Language Enumeration

When defining language for returning routing results, the following language enumerations can be used:

- Language.ALBANIAN
- Language.CHINESE
- Language.TAIWANESE
- Language.CROATIAN
- Language.CZECH
- Language.DANISH
- Language.DUTCH
- Language.ENGLISH
- Language.ENGLISH_US
- Language.ESTONIAN
- Language.FINNISH
- Language.FRENCH

- Language.GERMAN
- Language.HUNGARIAN
- Language.ITALIAN
- Language.JAPANESE
- Language.LATVIAN
- Language.LITHUANIAN
- Language.NORWEGIAN
- Language.PORTUGUESE
- Language.ROMANIAN
- Language.SLOVAK
- Language.SLOVENIAN
- Language.SPANISH
- Language.SWEDISH
- Language.RUSSIAN
- Language.TURKISH

REST API Language Enumeration

When defining language for returning routing results, the following language enumerations (ISO language code) can be used:

- sq
- zh_CN
- zh_TW
- hr
- cs
- da
- nl
- en
- en-US
- et
- fi
- fr
- de
- hu
- it
- ja
- lv
- lt

- no
- pt
- ro
- sk
- sl
- es
- sv
- ru
- tr