SuperMap Objects .NET
6R Technical Document
——— NetworkAnalyst

SuperMap Software Co., Ltd.
Beijing · China
Legal Statement

1. The copyright of this document is covered by SuperMap Software Co., Ltd. in accordance with the Copyright Law of the People’s Republic of China and the Universal Copyright Convention. If, without the written permission of the company, any part of the document shall not in any way or any reason be used, copied, modified, transmitted, or bundled with other products to be used, sold, tort reserved.

2. “超图”, “SuperMap”, and are the registered trademarks of SuperMap Software Co., Ltd., protected by the Copyright Law of the People’s Republic of China. If, without the written permission of the company, the trademarks shall not in any way or any reason be used, copied, modified, transmitted, or bundled with other products to be used, sold, tort reserved.

3. This document represents no responsibilities of any supplier or agent. Without statement, SuperMap Software Co., Ltd. has right to do modifications to this document.

4. The copyright of trademarks mentioned in this document belongs to the corresponding companies. Without the written permission of these companies, the trademarks shall not in any way or any reason be used, copied, modified, or transmitted.

5. The concerned software products and the updated products hereinafter in this document are developed and sold by SuperMap Software Co., Ltd.

Hereby declare

SuperMap Software Co., Ltd.:
Add: 7/F Tower B, Technology Fortune Center, No. 8 Xueqing Road,
Haidian District, Beijing, 100192, P. R. China
Tel: +86-10-82736655-4170
Fax: +86-10-82734630
HomePage: www.supermap.com

Sales: request@supermap.com

Tech Support: globalsupport@supermap.com

SuperMap Software welcomes all advices and suggestions from you.
Content

1 Summary ................................................................................................................................................. 1

2 Network Model Introduction .................................................................................................................. 1

2.1 Traffic Network Model ......................................................................................................................... 2

2.2 Public Pipeline Facilities Network ......................................................................................................... 2

3 Network Analysis ...................................................................................................................................... 3

4 Summary of Network Analysis Functionality ......................................................................................... 6

5 Building Network Analysis .................................................................................................................... 7

6 Functions of Traffic Network Analysis .................................................................................................... 9

6.1 Environment Configuration of Traffic Network Analysis ........................................................................ 9

6.2 Loading Environment Configuration of Traffic Network Analysis ..................................................... 12

6.3 Setting Parameters of Traffic Network Analysis ................................................................................... 13

6.4 Path Finding ........................................................................................................................................... 14

6.4.1 Steps to Achieve Path Finding .......................................................................................................... 15

6.4.2 How to Call the Path Finding ........................................................................................................... 15

6.4.3 The Outcome of the Path Finding ..................................................................................................... 16

6.5 The Closest Facility Analysis ................................................................................................................ 17

6.5.1 Steps to Achieve the Closest Facility Analysis ................................................................................ 18

6.5.2 How to Call the Closest Facility Analysis ......................................................................................... 18

6.5.3 The Outcome of the Closest Facility Analysis .................................................................................. 19

6.6 Traveling Salesman Problem Analysis ................................................................................................... 20

6.7 Service Area Analysis ........................................................................................................................... 23

6.7.1 Steps to achieve the Service Area Analysis ..................................................................................... 23

6.7.2 How to Call the Service Area Analysis ............................................................................................ 24

6.7.3 The Outcome of the Service Area Analysis ..................................................................................... 25

6.8 Logistics Vehicle Routing Analysis ...................................................................................................... 27
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.8.1 Steps to Achieve the Logistic Vehicle Routing Analysis</td>
<td>28</td>
</tr>
<tr>
<td>6.8.2 How to Call the Logistic Vehicle Routing Analysis</td>
<td>28</td>
</tr>
<tr>
<td>6.8.3 The Outcome of the Logistic Vehicle Routing Analysis</td>
<td>29</td>
</tr>
<tr>
<td>6.9 Allocation and Location</td>
<td>32</td>
</tr>
<tr>
<td>6.9.1 How to Call the Allocation and Location Analysis</td>
<td>35</td>
</tr>
<tr>
<td>6.9.2 The Outcome of the Allocation and Location Analysis</td>
<td>36</td>
</tr>
<tr>
<td>7 Facility Network Analysis</td>
<td>41</td>
</tr>
<tr>
<td>7.1 Steps to Achieve Facility Network Analysis</td>
<td>41</td>
</tr>
<tr>
<td>7.2 Building Network Dataset Flow Direction and Class</td>
<td>42</td>
</tr>
<tr>
<td>7.2.1 Building Network Dataset Flow Direction</td>
<td>42</td>
</tr>
<tr>
<td>7.2.2 The Result of Building Network Dataset Flow Direction</td>
<td>44</td>
</tr>
<tr>
<td>7.2.3 Building Network Dataset Hierarchy</td>
<td>47</td>
</tr>
<tr>
<td>7.3 Environment Configuration of the Facility Network Analysis</td>
<td>48</td>
</tr>
<tr>
<td>7.4 Loading the Facility Network Analysis</td>
<td>49</td>
</tr>
<tr>
<td>7.5 A List of Facility Network Analysis Functions</td>
<td>49</td>
</tr>
<tr>
<td>7.5.1 Checking Network Loop</td>
<td>49</td>
</tr>
<tr>
<td>7.5.2 Finding Interlinked Loop</td>
<td>49</td>
</tr>
<tr>
<td>7.5.3 Finding Unconnected Edges</td>
<td>50</td>
</tr>
<tr>
<td>7.5.4 Finding Connected Edges</td>
<td>51</td>
</tr>
<tr>
<td>7.5.5 Upstream Tracing</td>
<td>51</td>
</tr>
<tr>
<td>7.5.6 Downstream Tracing</td>
<td>51</td>
</tr>
<tr>
<td>7.5.7 Common Upstream Tracing</td>
<td>53</td>
</tr>
<tr>
<td>7.5.8 Common Downstream Tracing</td>
<td>55</td>
</tr>
<tr>
<td>7.5.9 Upstream Path Analysis</td>
<td>57</td>
</tr>
<tr>
<td>7.5.10 Downstream Path Analysis</td>
<td>58</td>
</tr>
<tr>
<td>7.5.11 Facility Network Path Analysis</td>
<td>60</td>
</tr>
</tbody>
</table>
Summary

With the development of human civilization, the public infrastructure has been greatly improved, such as the power facilities, telecommunications and cable television, road traffic with GPS monitor, water pipe network, heating pipe networks and etc. In order to provide maintenance, management, and further planning to these large scale infrastructures, computers become the necessity for helping our work. In GIS, these facilities are considered as "network system", which is a network-shape system consisted by many connected line segments. The network model is an abstraction of this system about real life. For instance, in the city traffic network, roads are represented by line segments, named as network arcs; similarly, road intersections and stops are taken as points, named as network nodes, etc. In the network model, resources and information can go from one point to another.

The network analysis is a process to solve the practical problems by analyzing on the network model, like path analysis, service zone analysis, closest facility analysis, etc. Currently, network analysis has been widely applied in electric navigation, traffic tourism, urban planning and management, logistic distribution, power industry, telecommunication, as well as layout design and query of various pipe line networks.

The following contents are introduced in this section:

1) General Introduction of Network Model
2) Basic Concepts of Network Analysis
3) Summary of Network Analysis Functionality
4) Building Network Analysis
5) Functionality of Traffic Network Analysis
6) Facility Network Analysis
Network Model Introduction

Network Model: it stores data which has network topology relationship with each other, including network line datasets, network point datasets as well as their spatial topology relationship. The line datasets are main datasets, while the point datasets are sub datasets.

Network Datasets
There are two type network models in GIS, traffic network model and public pipeline facilities network model.

### 2.1 Traffic Network Model

In traffic network model, the traveling pedestrians and transported resources inside the network can have various directions, speed and destinations, although the lines have been set direction in the model. For instance, a driver is able to turn and drive in any direction, choosing the parking time; while on a one-way road, he is not allowed to turn around. And this is totally different from the public pipeline facilities network situation.

### 2.2 Public Pipeline Facilities Network

Public pipeline facilities network has directions to represent the water stream, electric current, or any other mediums, which flow in the network according to particular principles. For instance, the water routes are designed beforehand, but the flow of the water could be different by changing some switches which work as the circulation principles in the network.
Network model not only has abstract topology relationship between edges and nodes, but also has the geometry positions and geography property features of GIS spatial data. Topology relationship is the spatial position between the geographic objects, such as points between lines, lines between regions, etc. The followings are the basic concepts of network model.

Diagram 2 Network Illustration

**Node**

A node is the connection place of two edges, please see Diagram one. Nodes are used to represent the crossroads, river meeting points in real life. The property tables of nodes and edges are used to record their adjacent relations.

**Edge**
An edge is the section line between two nodes. An edge connects with the rest of edges by nodes, please see Diagram one. It is used to deputy the high express, railway, transport lines of electric network, and rivers of hydrology network. There is topology structure between the edges.

**Network**

The network consists of a group of related edges, points and their properties, shown in Diagram one. Network represents the roads and pipelines in our life. The following factors should be defined to produce information such as demands, requirements, and centre points and so on.

**Impendensity**

In real life, the cost of a journey could be measured by distance, time, and cost. The cost on nodes or edges is abstracted into network impendence which is stored in the impendence field in the property table.

**Center**

Center is the discrete facilities which have functions to receive or supply resources, locating on the nodes. Facilities represent substance, resources, information, management, and cultural environment. For instance, in a school, the education resources are necessary for students who have to go for study; in a retail warehouse, the goods are stored to distribute to all the retail points. A center is actually a node in the network.

**Barrier Nodes and Barrier Edges**

The traffic jam is not rare in city life at all. Since it is a dynamic procedure and always happens randomly, there are no regulations to follow. In order to display the traffic conditions in real time, the edge in the traffic jam must be defined with property that is able to temporarily ban the trip and then free the edge when the traffics flow freely. Barrier nodes and barrier edges are designed to meet this requirement, which have comparatively independent features from the current network environment parameters.

**Turn Table**

Turning is a trip from one edge, passing a central node, to another edge. The cost of turning is the consumption during this trip, which stores in the turn table. All the turning directions of a crossroad should be listed in the turn table. Usually, there are four fields, related with the fields of nodes and edges, used to record the turning information, the initial edge field (FromEdgeID), terminal edge field (ToEdgeID), node annotation field (NodeID), and the cost of the turning (TurnCost). Each record lists the turning cost for one turning possibility. The minus cost value means forbidden turning.

There are many crossroads and divergences in the road network analysis. The turn table should be used for them to record the turning cost, please refer to Diagram three. The left diagram is a crossroad, and its turn table is the right table.
Diagram 3 the Structure of a Turn Table
Summary of Network Analysis Functionality

The network analysis functions offered by SuperMap Objects .NET are path finding, service area analysis, traveling salesman problem analysis, closest facility analysis, allocation and location, logistics vehicle routing analysis, as well as related analysis functions of facility network analysis. Facility network analysis is used to trace upstream and downstream analysis, routes analysis of the facility network, and connectivity analysis.

These powerful functions provided by SuperMap GIS are widely used in the following practical questions:

Place B is short of water, and in order to transmit water from Place A to Place B, the connectivity situation of the pipeline between the two places must be checked first. If it is in good shape, water will be switched on. This problem requires the connectivity function.

The tracing analysis is able to locate the pollution source according to local polluted areas.

The salesman path finding is extremely useful to delivers of the express delivery companies. A suitable shortest route can pass by all the delivery destinations. The fire fighting truck needs to choose a route that takes the least time to the accident place, and path finding is the best analysis method for them.

The service area analysis of SuperMap GIS can help to find a suitable location for a new post office in a city.

The police station receives a crime report, and they want the patrol car which is nearest to the accident place to deal with it. The closest facility analysis can give them satisfied answer.

A snack bar has M branches, and there are N customers order the take-out. The dispatcher of the snack bar can work out the most convenient routes for all the deliverers by using logistics vehicle routing analysis of SuperMap GIS.
Building Network Analysis

In order to conduct the network analysis functions offered by SuperMap Objects .NET, you have to build network datasets. First, you have to abstract the analyzing objects into lines and points, and then build the topology relationship between them; finally, a model containing the objects to imitate the relations between things in the real world will be built.

Building network datasets use the function BuildNetwork() of the NetworkBuilder class of the SuperMap Objects .NET. This is a reloading method. It can establish a network datasets by given a line dataset, building the topology relationship. Alternatively, it can build option parameters through a network datasets, controlling the topological transaction content during the building procedure of network datasets. Besides, to build a network dataset by using multiple point datasets and line datasets is the third way. We are going to introduce two building methods in the following sections.

**Method One: building a network model dataset according to specified line datasets, datasources and datasets names.**

**Syntax:**

```java
public static DatasetVector BuildNetwork(DatasetVector lineDataset, Datasource outputDatasource, String networkDatasetName )
```

**Parameters:**

- `lineDataset`: the line datasets that is used to build network datasets.
- `outputDatasource`: storing datasources of the established network datasets.
- `networkDatasetName`: names of the specified network datasets.

**Return Value:**

Returns the established datasets, which is vector dataset.

**Method Two: building network datasets by using the line dataset array and point dataset array.**

**Syntax:**

```java
public static DatasetVector BuildNetwork( DatasetVector[] lineDatasets, DatasetVector[] pointDatasets, Datasource outputDatasource, String networkDatasetName, NetworkSplitMode networkSplitMode, Double tolerance)
```
Parameters:

lineDatasets: the line dataset array that is used to build network datasets. This array includes one or more line datasets to establish network datasets.

pointDatasets: the point dataset array that is used to build network datasets. This array includes one or more point datasets to establish network datasets.

outputDatasource: storing the datasource of the established network datasets.

networkDatasetName: names of the specified network datasets.

networkSplitMode: specified split mode.

tolerance: the tolerance value of points in the point dataset that could be considered as the points on the network.

Return Value:

Returns the established network dataset, which is vector dataset.
Functions of Traffic Network Analysis

Functions of traffic network analysis includes: path finding, service area analysis, traveling salesman problem analysis, the closest facility analysis, allocation and location, and logistics vehicle routing analysis.

These stated analyses should be achieved through three steps: environment configuration of the traffic network analysis, loading environment configuration of the traffic network analysis, and setting parameters of the traffic network analysis.

6.1 Environment Configuration of Traffic Network Analysis

All the functions of traffic network analysis are achieved by functions provided by class TransportationAnalyst. The functions of the class require environment configurations to get the necessary parameter information. Environment configuration could be done by setting an attribute AnalystSetting of the class TransportationAnalyst. In a word, to set environment configuration is actually to set attributes in the class TransportationAnalystSetting. Please note that the environment configuration will affect the analysis result very much.

Syntax:

TransportationAnalystSetting TransportationAnalyst .AnalystSetting

<table>
<thead>
<tr>
<th>Properties</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NetworkDataset</td>
<td>Obtaining or setting the analysing network datasets.</td>
</tr>
</tbody>
</table>

Syntax: DatasetVector TransportationAnalystSetting .NetworkDataset

<table>
<thead>
<tr>
<th>Properties</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NodeIDField</td>
<td>Obtaining or setting the ID fields of network nodes.</td>
</tr>
</tbody>
</table>
### Functions of Traffic Network Analysis

<table>
<thead>
<tr>
<th>Syntax: String TransportationAnalystSetting .NodeIDField</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>EdgIDField</strong>: Obtaining or setting the ID fields of network edges.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Syntax: String TransportationAnalystSetting .EdgIDField</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>FNodeIDField</strong>: Obtaining or setting the ID field of initial node of an edge.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Syntax: String TransportationAnalystSetting .FNodeIDField</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>TNodeIDField</strong>: Obtaining or setting the ID field of terminal node of an edge.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Syntax: String TransportationAnalystSetting .TNodeIDField</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>WeightFieldInfos</strong>: Obtaining or setting weight fields' information set object. WeightFieldInfos is the set of WeightFieldInfo. In the WeightFieldInfo, the fields of positive weight value and negative weight value, which represent the cost of the network analysis, could be set.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Syntax: WeightFieldInfos TransportationAnalystSetting .WeightFieldInfos</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Tolerance</strong>: Obtaining or setting the tolerance between nodes and edges.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Syntax: Double TransportationAnalystSetting .Tolerance</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>BarrierNodes</strong>: Obtaining or setting the ID list of barrier nodes.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Syntax: Int32[] TransportationAnalystSetting .BarrierNodes</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>BarrierEdges</strong>: Obtaining or setting the ID list of barrier edges.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Syntax: Int32[] TransportationAnalystSetting .BarrierEdges</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>TurnDataset</strong>: Obtaining or setting datasets of turn table.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Syntax: DatasetVector TransportationAnalystSetting.TurnDataset</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>TurnNodeIDField</strong>: Obtaining or setting the ID field of the turning node.</td>
</tr>
</tbody>
</table>

<p>| Syntax: String TransportationAnalystSetting .TurnNodeIDField |</p>
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TurnFEdgeIDField</td>
<td>Obtaining or setting the ID field of the initial turning edge.</td>
</tr>
<tr>
<td>Syntax: String</td>
<td>TransportationAnalystSetting.TurnFEdgeIDField</td>
</tr>
<tr>
<td>TurnTEdgeIDField</td>
<td>Obtaining or setting the ID field of the terminal turning edge.</td>
</tr>
<tr>
<td>Syntax: String</td>
<td>TransportationAnalystSetting.TurnTEdgeIDField</td>
</tr>
<tr>
<td>TurnWeightFields</td>
<td>Obtaining or setting the field set of turning weight.</td>
</tr>
<tr>
<td>Syntax: String[]</td>
<td>TransportationAnalystSetting.TurnWeightFields</td>
</tr>
<tr>
<td>NodeNameField</td>
<td>Obtaining or setting the names of fields that stores the node names.</td>
</tr>
<tr>
<td>Syntax: String</td>
<td>TransportationAnalystSetting.NodeNameField</td>
</tr>
<tr>
<td>EdgeNameField</td>
<td>Obtaining or setting names of fields that stores the edge names.</td>
</tr>
<tr>
<td>Syntax: String</td>
<td>TransportationAnalystSetting.EdgeNameField</td>
</tr>
<tr>
<td>RuleField</td>
<td>Obtaining or setting the fields that represent the traffic principles of the edges.</td>
</tr>
<tr>
<td>Syntax: String</td>
<td>TransportationAnalystSetting.RuleField</td>
</tr>
<tr>
<td>FTSingleWayRuleValues</td>
<td>Obtaining or setting the arrays that represent the positive single way string. Exactly speaking, if the value of attribute RuleField is one of the strings in the array, the correspondent edge is a positive single way road.</td>
</tr>
<tr>
<td>Syntax: String[]</td>
<td>TransportationAnalystSetting.FTSingleWayRuleValues</td>
</tr>
<tr>
<td>TFSingleWayRuleValues</td>
<td>Obtaining or setting string arrays that represent the reverse single way. Exactly speaking, if the value of attribute RuleField is one of the strings in the array, the correspondent edge is a negative single way road.</td>
</tr>
<tr>
<td>Syntax: String[]</td>
<td>TransportationAnalystSetting.TFSingleWayRuleValues</td>
</tr>
<tr>
<td>ProhibitedWayRuleValues</td>
<td>Obtaining or setting string arrays that represent the prohibited lines; i.e. if the value of attribute RuleField is one of the strings in the array, the correspondent edge is prohibited.</td>
</tr>
<tr>
<td>Syntax: String[]</td>
<td>TransportationAnalystSetting.ProhibitedWayRuleValues</td>
</tr>
</tbody>
</table>
the correspondent network edge is prohibited line.

Syntax: String[]TransportationAnalystSetting.ProhibitedWayRuleValues

TwoWayRuleValues | Obtaining or setting string arrays that represent the double-way road, which means if the value of attribute RuleField is one of the strings in the array, the correspondent network edge is a double-way road.

Syntax: String[]TransportationAnalystSetting.TwoWayRuleValues

<table>
<thead>
<tr>
<th>Properties</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FTWeightField</td>
<td>Forward Resistance Field</td>
</tr>
<tr>
<td>Syntax: String WeightFieldInfo.FTWeightField</td>
<td></td>
</tr>
<tr>
<td>TFWeightField</td>
<td>Reverse Resistance Field</td>
</tr>
<tr>
<td>Syntax: String WeightFieldInfo.TFWeightField</td>
<td></td>
</tr>
<tr>
<td>Name</td>
<td>Names of Weight Field information</td>
</tr>
<tr>
<td>Syntax: String WeightFieldInfo.Name</td>
<td></td>
</tr>
</tbody>
</table>

### 6.2 Loading Environment Configuration of Traffic Network Analysis

After you finish the setting of traffic network analysis environment, the next step is to load your configuration by calling `Load()` function in class `TransportationAnalyst`. This function will make your custom settings effective.

Syntax:

```csharp
public Boolean TransportationAnalyst.Load()
```

Return Value:

Returns True if the loading activity is achieved, otherwise returns False.
6.3 Setting Parameters of Traffic Network Analysis

TransportationAnalystParameter class is used as parameters in many network analyses, such as path finding, traveling salesman problem analysis, closest facility analysis, service area analysis. Hence, we can set the properties of the class object transportationAnalystParameter to provide traffic network analysis function to corresponding network analysis functions. It allows setting barriers in nodes or lines, name mark of weight field information, turning weight field. Meanwhile, you may also able to get a desired result by setting if the node set, arc set, and route object set and station set are shown in the outcome. The following table is an introduction of class transportationAnalystParameter properties.

<table>
<thead>
<tr>
<th>Properties</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BarrierEdges</td>
<td>Obtaining or configuration of network barrier edge set</td>
</tr>
<tr>
<td>Syntax: Int32[]</td>
<td>TransportationAnalystParameter.BarrierEdges</td>
</tr>
<tr>
<td>BarrierNodes</td>
<td>Obtaining or configuration network barrier point set.</td>
</tr>
<tr>
<td>Syntax: Int32[]</td>
<td>TransportationAnalystParameter.BarrierNodes</td>
</tr>
<tr>
<td>IsEdgesReturn</td>
<td>Obtaining or configuration if contain the past edges set in the result</td>
</tr>
<tr>
<td>Syntax: Boolean</td>
<td>TransportationAnalystParameter.IsEdgesReturn</td>
</tr>
<tr>
<td>IsNodesReturn</td>
<td>Obtaining or configuration if contain the past points set in the result</td>
</tr>
<tr>
<td>Syntax: Boolean</td>
<td>TransportationAnalystParameter.IsNodesReturn</td>
</tr>
<tr>
<td>IsPathGuidesReturn</td>
<td>Obtaining or configuration if contain driving guide set in the result</td>
</tr>
<tr>
<td>Syntax: Boolean</td>
<td>TransportationAnalystParameter.IsPathGuidesReturn</td>
</tr>
<tr>
<td>IsRoutesReturn</td>
<td>Obtaining or configuration if contain any route object (i.e. set of GeoLineM) set in the result.</td>
</tr>
</tbody>
</table>
6 Functions of Traffic Network Analysis

Syntax: Boolean TransportationAnalystParameter.IsRoutesReturn

IsStopIndexesReturn
Obtaining or configuration if contain index set of past stops in the result.

Syntax: Boolean TransportationAnalystParameter.IsStopIndexReturn

Nodes
Obtaining or configuration the ID set of past nodes. Mutual exclusion with attribute Points. Only one could be used in each time.

Syntax: Int32 [] TransportationAnalystParameter.Nodes

Points
Obtaining or configuration the past points set. Mutual exclusion with attribute Nodes. Only one of them could be used in each time.

Syntax: Point2Ds TransportationAnalystParameter.Points

TurnWeightField
Obtaining or configuration the turning weight field.

Syntax: String TransportationAnalystParameter.TurnWeightField

WeightName
Obtaining or configuration name mark of weight field information, i.e. the value of Name of a weight field information object (WeightFieldInfo) in weight field information set (WeightFieldInfos), which is in network analysis environment configuration (NetworkAnalystSetting).

Syntax: String TransportationAnalystParameter.WeightName

We are going to introduce you, together with the components products of SuperMap Objects .NET, all these functions and how to achieve them.

6.4 Path Finding

Path finding is looking for a route between two nodes in the network that has the least resistance. The route should be accessed by the selecting sequence of nodes. The "least resistance" at here means a lot, according to your practical situation it could be any simple factors, for instance, the least time to use, the lowest cost to pay, the most enjoyable scenery alone the road, the best
traffic to go, the fewest bridges and toll gates in the total journey, or the most villages to pass by. For instance, we want to visit four nodes 1, 2, 3, 4 in order of their number. The first thing that needs to be done is obtaining the optimum path R_{1,2} between node 1 and 2. Second, the optimum path R_{2,3} between node 2 and node 3 should be located. Third, the optimum path R_{3,4} between node 3 and 4 should be found. Finally, the best route visiting the four nodes from number 1 is R = R_{1,2} + R_{2,3} + R_{3,4}.

6.4.1 Steps to Achieve Path Finding

1. Create Network Dataset. If there is no network dataset used to analyze, we should use BuildNetwork() method in NetworkBuilder class to create network dataset.

2. Set traffic network analysis environment, namely set AnalystSetting property in TransportationAnalyst class.

3. Set traffic network analysis parameter class object, namely set parameter object of TransportationAnalystParameter class.

4. Call path analysis method to implement Path Analysis and get the analysis result.

6.4.2 How to Call the Path Finding

Syntax:

Public TransportationAnalystResult TransportationAnalyst.FindPath(TransportationAnalystParameter parameter, Boolean hasLeastEdgeCount)

Parameters:

parameter: traffic network analysis parameter object.

Through setting parameter of TransportationAnalystParameter type in the method, N points in the analysis and their sequence are set. There are two modes:

(1) Use Nodes property of TransportationAnalystParameter type's parameter object to specify points Path Analysis passes through in the form of nodes ID array of network dataset. So points Path Analysis passes through are the corresponding network nodes, and the sequence of the points are the sequence of the nodes in the nodes ID array;

(2) Use Points property of TransportationAnalystParameter type's parameter object to specify points Path Analysis passes through in the form of coordinate points string of the past points.
Points property of TransportationAnalystParameter type's parameter object is corresponding to coordinate points set, so the sequence of the points Path Analysis passes through is the sequence of coordinate point in the points set.

Besides setting the points Path Analysis passes through, some information required in Path Analysis can also be set in the parameter of TransportationAnalystParameter type.

hasLeastEdgeCount: whether number of arcs is smallest. True represents that it will query according to the smallest arcs. Small arcs don't represent short arcs, so the query result may not be the shortest path.

**Return Value:**

Returns the result of Path Analysis, which is saved in the class of TransportationAnalystResult.

### 6.4.3 The Outcome of the Path Finding

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Routes</td>
<td>Gets the route set of the analyst result, that is, the set of the GeoLineM class.</td>
</tr>
<tr>
<td>Syntax: GeoLineM[] Routes</td>
<td></td>
</tr>
<tr>
<td>Nodes</td>
<td>Gets the array of the past node ID in the result of analysis.</td>
</tr>
<tr>
<td>Syntax: Int32[][] Nodes</td>
<td></td>
</tr>
<tr>
<td>Edges</td>
<td>Gets the array of the past edge ID in the result of analysis.</td>
</tr>
<tr>
<td>Syntax: Int32[][] Edges</td>
<td></td>
</tr>
<tr>
<td>PathGuides</td>
<td>Get PathGuide array.</td>
</tr>
<tr>
<td>Syntax: PathGuide[]PathGuides</td>
<td></td>
</tr>
<tr>
<td>StopIndexes</td>
<td>Gets the two-dimensional arrays of the stop index. The values in this array indicate the order of the stops after analyzing. The first array number is 1 if the analyze has a result, otherwise 0.</td>
</tr>
</tbody>
</table>

**SuperMap Objects .NET Technology Documents**
6 Functions of Traffic Network Analysis

Note:

(1) The node mode of FindPath analysis: If the array of node id we set is {1, 3, 5}, as the result order of passed nodes id should be the same to the node order we set, so the value of StopIndexes will be {0, 1, 2}, it is the index order of result node in the initial node array.

(2) The point mode of FindPath analysis: The result node id passed has no relations to the point index we set, so the value of StopIndexes has not any meanings.

<table>
<thead>
<tr>
<th>Syntax: Int32[][] StopIndexes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Weights</td>
</tr>
<tr>
<td>Gets the array of the analyst weights. The number in the array is 1 if the analyze has a result, otherwise 0, that is, Weights[0] indicates the total weight while analyzing.</td>
</tr>
<tr>
<td>Syntax: Double[] Weights</td>
</tr>
<tr>
<td>StopsWeights</td>
</tr>
<tr>
<td>Gets the weight in each stop, that is, the cost in each stop.</td>
</tr>
<tr>
<td>Syntax: Double[][], StopsWeights</td>
</tr>
</tbody>
</table>

The similarities and differences between the path finding and the traveling salesman problem analysis:

Similarities: looking for a route passing by necessary nodes with the least cost.

Differences: in path finding, the visiting to all the nodes should follow the specified sequence. However, the traveling salesman problem analysis requires an optimum sequence to access all the nodes.

6.5 The Closest Facility Analysis

The closest facility analysis aims to find a route between an event node to a group of facilities points with the least cost. The outcome displays the optimum routes, the lowest cost, and the best driving directions between the event location and one or more facilities.

Facility point: the basic elements of the closest facility analysis. Usually, the service facilities such as school, supermarket, and gas station are the facility points.
Event point: another basic element of the closest facility analysis, which is the event place that requires the service facility.

Scene: there is a traffic accident reported on the event point, requiring a hospital which must be within ten minutes driving distance. The hospitals around are the facility points; while the accident happened place is an event point. Since the closest facility analysis is a route analysis, the route with barrier points and edges will not be included in the results.

6.5.1 Steps to Achieve the Closest Facility Analysis

1. Create Network Dataset. If there is no network dataset used to analyze, we should use BuildNetwork() method in NetworkBuilder class to create network dataset.

2. Set traffic network analysis environment, namely set AnalystSetting property in TransportationAnalyst class.

3. Set traffic network analysis parameter class object, namely set parameter object of TransportationAnalystParameter class.

4. Call Finding Closest Facility analysis method to implement Finding Closest Facility Analysis and get the analysis result.

6.5.2 How to Call the Closest Facility Analysis

Syntax:

public TransportationAnalystResult TransportationAnalyst.FindClosestFacility(TransportationAnalystParameter parameter, Int32 eventId, Int32 facilityCount, Boolean isFromEvent, double maxWeight)

public TransportationAnalystResult TransportationAnalyst.FindClosestFacility(TransportationAnalystParameter parameter, Point2D eventPoint, Int32 facilityCount, Boolean isFromEvent, double maxWeight)

Parameters:

parameter: traffic network analysis parameter object.

Through setting parameter of TransportationAnalystParameter type in the method, facility points are set. There are two modes:
Use Nodes property of TransportationAnalystParameter type's parameter object to specify facility points in the form of node ID array of network dataset. So the facility points in the process of analysis are the corresponding network nodes.

Use Points property of TransportationAnalystParameter type's parameter object to specify facility points in the form of coordinate points string of the facility points. Points property of TransportationAnalystParameter type's parameter object is corresponding to coordinate points set.

Besides setting the facility points, some information required in Finding Closest Facilities Analysis can also be set in the parameter of TransportationAnalystParameter type.

- **eventPoint/ eventide**: specify event point.
  
  In Finding Closest Facilities Analysis event point can be specified in two methods. One is through coordinate point, that is, eventPoint parameter in mode; the other is through node ID in network dataset, regarding network nodes as event points, that is, eventID parameter in mode 1.

- **facilityCount**: count of facility points to query.

- **isFromEvent**: whether to query from event point to facility point.

- **maxWeight**: Search radius. In the same unit of the impedance. If you want to search all the network, set the radius to 0.

**Return Value:**

Returns the result of Finding Closest Facilities Analysis, which is saved in TransportationAnalystResult class.

### 6.5.3 The Outcome of the Closest Facility Analysis

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Routes</td>
<td>Gets the GeoLineM object which contains the routes in the analysis result.</td>
</tr>
<tr>
<td>Syntax: GeoLineM[] Routes</td>
<td></td>
</tr>
<tr>
<td>Nodes</td>
<td>Gets the array of the past node ID in the result of analysis.</td>
</tr>
<tr>
<td>Syntax: Int32[][] Nodes</td>
<td></td>
</tr>
</tbody>
</table>
6.6  Traveling Salesman Problem Analysis

Salesman path finding is looking for a route passing by a group of nodes, with the custom visiting order to achieve as low travelling cost sum as possible.

If a terminal point is designed in the salesman path finding, the salesman should visit it as the last one in his trip. His order of visiting to other points is still up to himself.

The following steps are able to help you achieve the salesman path finding:

First, building a network dataset. You need a network dataset to be used in the analysis. Please use the BuildNetwork() function in class NetworkBuilder.

Setting analysis environment of traffic network, i.e. setting the attribute AnalystSetting in class TransportationAnalyst

Setting the objects of parameter class in traffic network analysis, i.e. setting the parameter objects in class TransportationAnalystParameter
Call the traveling salesman problem analysis function to achieve the analysis as well as the outcome.

Syntax:

```csharp
public TransportationAnalystResult
TransportationAnalyst.FindTSPPath(TransportationAnalystParameter parameter, Boolean isEndNodeAssigned)
```

Parameters:

- `parameter`: parameters objects in traffic network analysis.
  
  To designate the nodes which are going to be passed by salesmen is set in the parameter "parameter" of TransportationAnalystParameter parameter objects. There are two methods to designate the nodes to pass:

  1. The attribute `Nodes` of parameter object TransportationAnalystParameter could be used to designate the points as a node ID array. Hence, the past points in the analysis are the correspondent nodes in the network. If there is a point selected as the terminal point, this point is the last node in the node ID array.

  2. If you have the coordinate information of the points that the route is going to pass, the attribute `Points` of the TransportationAnalystParameter could be used. This attribute is a set of coordinates. If there is a point selected as the terminal point, this coordinate of this point would be the last one in `Points`.

There are more parameters in TransportationAnalystParameter, which are necessary in the salesman path finding analysis.

- `hasLeastEdgeCount`: it has value True if you set a terminal point which should be the last point for salesmen to visit. The visiting order of the rest points could be decided by the salesmen.

Return Value:

Returns the result of traveling salesman problem analysis, which saves in the class of TransportationAnalystResult.

The Outcome of Traveling Salesman Problem Analysis List

Table 6 The Properties Table of Class TransportationAnalystResult

<table>
<thead>
<tr>
<th>Properties</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Routes</td>
<td>Obtaining the object set of the outcome routes. This</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>----------</td>
<td>-----</td>
</tr>
<tr>
<td>object set is named GeoLineM.</td>
<td></td>
</tr>
<tr>
<td>Syntax: GeoLineM[] Routes</td>
<td></td>
</tr>
<tr>
<td>Nodes</td>
<td>Obtaining the node ID set along the analysing outcome route.</td>
</tr>
<tr>
<td>Syntax: Int32[][] Nodes</td>
<td></td>
</tr>
<tr>
<td>Edges</td>
<td>Obtaining the edge set contributed to the outcome route.</td>
</tr>
<tr>
<td>Syntax: Int32[][] Edges</td>
<td></td>
</tr>
<tr>
<td>PathGuides</td>
<td>Obtaining driving guidance set.</td>
</tr>
<tr>
<td>Syntax: PathGuide[]PathGuides</td>
<td></td>
</tr>
<tr>
<td>StopIndexes</td>
<td>Obtaining the two dimension array of nodes' index, it represents the order of the nodes. If there is any outcome, the first dimension is 1, otherwise, it is zero.</td>
</tr>
<tr>
<td>Syntax: Int32[][] StopIndexes</td>
<td></td>
</tr>
<tr>
<td>Weights</td>
<td>Obtaining the weight array that represents cost. If there is any outcome, the length of the array is 1, otherwise, it is zero. In a word, the Weights [0] is the total cost.</td>
</tr>
<tr>
<td>Syntax: Double[] Weights</td>
<td></td>
</tr>
</tbody>
</table>

**Note:**

(1) The node mode of FindTSPPath analysis: If the array of node id we set is {1, 3, 5}, as the result order of passed nodes id is {3, 5, 1}, so the value of the StopIndexes will be {1, 2, 0}, it is the index order of result node in the initial node array.

(2) The point mode of FindPath analysis: If the array of points we set is {Pnt1, Pnt2, Pnt3}, as the result order of passed points is {Pnt2, Pnt3, Pnt1}, so the value of the StopIndexes will be {1, 2, 0}, it is the index order of result point in the initial point array.
6 Functions of Traffic Network Analysis

### StopsWeights

<table>
<thead>
<tr>
<th>StopsWeights</th>
<th>Obtaining weights of the past stops.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Syntax: Double[][] StopsWeights</td>
<td></td>
</tr>
</tbody>
</table>

6.7 **Service Area Analysis**

Service area analysis displays the service scope of a service centre on the network.

**Relative Concepts:**

Service area: a zone displays all the reachable borders and points, restricted by certain resistance, from a specified center. Simply speaking, service area analysis tells us the service scope of a service center with certain restriction.

Service area analysis calculates the service scope of a specified position on the network. For instance, to display the 30-minute driving service area centered by a node on the network. In the outcome, a zone with the node as a center is illustrated. Starting from the central node, you are able to drive to any place within the zone in 30 minutes.

Mutex of multiple service area: we will apply mutex operation to the overlapped areas of adjacent service areas.

Where to start the analysis: there are two ways to conduct the analysis. Sometimes, it is necessary to analyze from the central point, which means the service centre should deliver their service to the destination, for instance a dairy station has to sent milk to nearby neighborhoods, the analysis should use the dairy station as a centre. In other cases, it is necessary to analyze from the destination, which indicates the service could only be obtained in the service center, for instance, in order to know the service scope of a school, the analysis should be conducted from students' neighborhoods, as every student has to go to school to have classes.

6.7.1 **Steps to achieve the Service Area Analysis**

1. Create Network Dataset. If there is no network dataset used to analyze, we should use BuildNetwork() method in NetworkBuilder class to create network dataset.

2. Set traffic network analysis environment, namely set AnalystSetting property in TransportationAnalyst class.

3. Set traffic network analysis parameter class object, namely set parameter object of TransportationAnalystParameter class.
4. Call Service Area analysis method to implement Service Area Analysis and get the analysis result.

### 6.7.2 How to Call the Service Area Analysis

**Syntax:**

```csharp
public ServiceAreaResult FindServiceArea(TransportationAnalystParameter parameter, double[] weights, Boolean isFromCenter, Boolean isCenterMutuallyExclusive)
```

**Parameters:**

- **Parameter:** traffic network analysis parameter object.

In Service Area analysis, through setting parameter of TransportationAnalystParameter type, service center is set, and we can set multi service center points. There are two modes:

Use Nodes property of TransportationAnalystParameter type's parameter object to specify service center point in the form of node ID array of network dataset. So service center point in the process of analysis is the corresponding network nodes.

Use Points property of TransportationAnalystParameter type's parameter object to specify service center point in the form of coordinate point's string of the service center points. Point’s property of TransportationAnalystParameter type's parameter object is corresponding to coordinate points set.

Some information required in Service Area Analysis can also be set in the parameter of TransportationAnalystParameter type.

- **Weights:** cost array.
- **isFromCenter:** whether to start analyzing from center.
- **isCenterMutuallyExclusive:** whether center points are exclusive, that is, judge whether to make exclusive disposal according to the distance of center points. If analyzed service areas overlap, the parameter is set to make exclusive disposal. For example, the left graph doesn't make exclusive disposal, and the right does. Notice: centers exclusive analysis isn't supported for the moment.
Return Value:

Returns the result of Service Area Analysis, which is saved in ServiceAreaResult class.

### 6.7.3 The Outcome of the Service Area Analysis

Result class of Service Area analysis: ServiceAreaResult class, inherits from TransportationAnalystResult class, so it owns all properties of TransportationAnalystResult class.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Routes</td>
<td>Gets the GeoLineM object which contains the routes in the analysis result.</td>
</tr>
<tr>
<td>Syntax: GeoLineM[] Routes</td>
<td></td>
</tr>
</tbody>
</table>

*Table 7 ServiceAreaResult class property table*
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nodes</td>
<td>Gets the array of the passed node ID in the result of analysis. Syntax: <code>Int32[][] Nodes</code></td>
</tr>
<tr>
<td>Edges</td>
<td>Gets the array of the passed edge ID in the result of analysis. Syntax: <code>Int32[][] Edges</code></td>
</tr>
<tr>
<td>StopIndexes</td>
<td>Gets the two-dimensional arrays of the stop index. The values in this array indicate the order of the stops after analyzing. The first array number is 1 if the analyze has a result, otherwise 0. Syntax: <code>Int32[][] StopIndexes</code></td>
</tr>
<tr>
<td>Weights</td>
<td>Gets the array of the analyst weights. The number in the array is 1 if the analyze has a result, otherwise 0, that is , <code>Weights[0]</code> indicates the total weight while analyzing. Syntax: <code>Double[] Weights</code></td>
</tr>
<tr>
<td>StopsWeights</td>
<td>Gets the weight in each stop, that is, the cost in each stop. Syntax: <code>Double[][] StopsWeights</code></td>
</tr>
<tr>
<td>ServiceRegions</td>
<td>Gets the set of the service region objects. Syntax: <code>GeoRegion[] ServiceRegions</code></td>
</tr>
<tr>
<td>ServiceRouteCounts</td>
<td>Gets the count array of the route of each service region in <code>ServiceAreaResult</code>. Syntax: <code>Int32[] ServiceRouteCounts</code></td>
</tr>
</tbody>
</table>
The following graph shows what problem Service Area analysis will resolve and what information the analysis result provides. Blue points represent service center providing service, and regions with kinds of colors represent service area within the limit of specified resistance centered on the relevant service center.

### 6.8 Logistics Vehicle Routing Analysis

Logistic vehicle routing analysis aims to deliver goods to N destinations from M distribution centers (M and N are positive integer) by providing economic and effective distribution routes, as well as the correspondent travelling route.

Logistic vehicle routing analysis is helpful to reasonably distribute the send-out order and sending routes, containing the cost of a center or the sum of the total distribution cost within the lowest scale.

Logistic vehicle routing analysis outcome will display all the destinations of each distribution centre, the order of the destinations to deliver, and the correspondent travelling road. As a result, the cost of a center or the sum of the total distribution cost will be the lowest one.
6.8.1 Steps to Achieve the Logistic Vehicle Routing Analysis

1. Create Network Dataset. If there is no network dataset used to analyze, we should use BuildNetwork() method in NetworkBuilder class to create network dataset.

2. Set traffic network analysis environment, namely set AnalystSetting property in TransportationAnalyst class.

3. Set traffic network analysis parameter class object, namely set parameter object of TransportationAnalystParameter class.

4. Call Logistics Analysis method to implement Logistics Analysis and get the analysis result.

6.8.2 How to Call the Logistic Vehicle Routing Analysis

Syntax:

```csharp
public TransportationAnalystResult TransportationAnalyst.FindMTSPPath(TransportationAnalystParameter parameter, Int32[] centerNodes, Boolean hasLeastTotalCost)

public TransportationAnalystResult TransportationAnalyst.FindMTSPPath(TransportationAnalystParameter parameter, Point2Ds centerPoints, Boolean hasLeastTotalCost)
```

Parameters:

- parameter: traffic network analysis parameter object.
  Through setting parameter of TransportationAnalystParameter type in Logistics Analysis method, destinations are set. There is two modes:

  Use Nodes property of TransportationAnalystParameter type's parameter object to specify destinations in the form of nodes ID array of network dataset. So destinations in the process of analysis are the corresponding network nodes.

  Use Points property of TransportationAnalystParameter type's parameter object to specify destinations in the form of coordinate point’s string of the destinations. Point’s property of TransportationAnalystParameter type's parameter object is corresponding to coordinate points set.
Some information required in Logistics Analysis can also be set in the parameter of TransportationAnalystParameter type.

centerNodes/Point2Ds centerPoints: specify distribution center.

The necessary parameters in Logistics Analysis are distribution center and distribution destination. Two modes are used to specify distribution center.

1. It's set by centerNodes parameter in Logistics Analysis method 1, which is a network node ID array, so the relevant network node is set to be distribution center.

2. It's set by centerPoints parameter in Logistics Analysis method 2, which is a coordinate point set, so the relevant coordinate point is set to be distribution center.

hasLeastTotalCost: whether the distribution mode is the least cost scheme. If true, it will distribute according to the least cost, when the result may be that some distribution centers spend more and others spend less. If false, it represents the partially best, which will control how much every distribution center spends to make the cost relatively even, when the total cost may be not the least. The parameter is set False by default.

Return:

Returns the result of Logistics Analysis, which is saved in the class of TransportationAnalystResult.

6.8.3 The Outcome of the Logistic Vehicle Routing Analysis

Result information of Logistics Analysis is saved in relevant properties of TransportationAnalystResult class object. The following is a detailed introduction to the meaning of every property in TransportationAnalystResult class, combining the result of Logistics Analysis, as shown in the table below.

Table 8 TransportationAnalystResult class property table

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Routes</td>
<td>Gets the route set of the analyst result, that is, the set of the GeoLineM class.</td>
</tr>
<tr>
<td>Syntax: GeoLineM[] Routes</td>
<td></td>
</tr>
<tr>
<td>Nodes</td>
<td>Gets the array of the past node ID in the result of</td>
</tr>
</tbody>
</table>
### Functions of Traffic Network Analysis

**Nodes**

The property is corresponding to a two-dimensional array, which represents storing the past nodes according to the distribution program of every destination got by the analysis result, that is, the array's dimension 1 marks distribution program number of distribution destination which is used to differentiate distribution programs of different destinations; dimension 2 stores node ID number of the relevant distribution program.

**Syntax:** Int32[][] Nodes

**Edges**

Gets the array of the past edge ID in the result of analysis. The property is corresponding to a two-dimensional array, which represents storing the past edges according to the distribution program of every destination got by the analysis result, that is, the array's dimension 1 marks distribution program number of distribution destination which is used to differentiate distribution programs of different destinations; dimension 2 stores edge ID number of the relevant distribution program.

**Syntax:** Int32[][] Edges

**PathGuides**

Get PathGuide array.

**Syntax:** PathGuide[] PathGuides

**StopIndexes**

Get a two-dimensional array of stop index, which reflects the sequence of stops after analysis. The property is corresponding to a two-dimensional array, which represents storing indexes of the past stops (that is, distribution destinations) according to the distribution program of every destination got by the analysis result, that is, the array's dimension 1 marks distribution program number of distribution destination which is used to differentiate distribution programs of different destinations; dimension 2 stores indexes of the passed stops (that is, distribution destinations) of the relevant distribution program.

**Note:**

1. The node mode of FindMTSPPath analysis: It is
<table>
<thead>
<tr>
<th>Syntax: Int32[][] StopIndexes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Weights</td>
</tr>
<tr>
<td>Syntax: Double[] Weights</td>
</tr>
<tr>
<td>StopsWeights</td>
</tr>
<tr>
<td>Syntax: Double[][] StopsWeights</td>
</tr>
</tbody>
</table>

**Example:** There are 50 book retailers (distribution destinations), and 4 book distribution centers. We now want to find the best route for each of these 4 distribution centers to distribute books to the book retailers. This is a logistics analysis problem, which is illustrated in the figures below.

Figure 1 shows the result of the logistics analysis. The 4 big spots represent the book distribution centers, and the 50 smaller spots represent book retailers. The route found for each center has been highlighted in a different color.
6.9 Allocation and Location

Location selection and location analysis is able to find the most optimum or best location for one or more facilities to build to provide service in the most economic and effective way. Allocation
and location analysis does not only include selecting address procedure but also the partition process which means to locate reasonable demands into these new facilities.

6.9.1 Steps to Achieve Allocation and Location Analysis

1. Create Network Dataset. If there is no network dataset used to analyze, we should use BuildNetwork() method in NetworkBuilder class to create network dataset.

2. Set traffic network analysis environment, namely set AnalystSetting property in TransportationAnalyst class.

3. Set location-allocation analysis parameter class object, namely set parameter object of LocationAnalystParameter class.

6.9.2 Parameters Configuration for the Allocation and Location Analysis

Through LocationAnalystParameter class set location-allocation analysis parameters to provide the necessary parameter information for location-allocation analysis. The detailed properties information is as shown as the following table:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SupplyCenters</td>
<td>Gets or sets the supply centers. SupplyCenters class is SupplyCenter class set. Property of SupplyCenter, see in table 11.</td>
</tr>
<tr>
<td>Syntax: SupplyCenters SupplyCenters</td>
<td></td>
</tr>
<tr>
<td>NodeDemandField</td>
<td>Gets or sets the field representing the demand of the nodes which are regarded as the demand centers in the network dataset.</td>
</tr>
<tr>
<td>Syntax: String NodeDemandField</td>
<td></td>
</tr>
<tr>
<td>ExpectedSupplyCenterCount</td>
<td>Gets or sets the number of the supply centers which will be used to build some establishments. If the parameter is set to zero, the minimal number of required supply</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------------------------</td>
<td>------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>ExpectedSupplyCenterCount</td>
<td>Centers will be computed to cover the whole analysis area.</td>
</tr>
<tr>
<td>Syntax: Int32 ExpectedSupplyCenterCount</td>
<td></td>
</tr>
<tr>
<td>IsFromCenter</td>
<td>Gets or sets whether to analyze from the supply centers or not. If true it represents from the supply centers; if false it doesn't.</td>
</tr>
<tr>
<td>Syntax: Boolean IsFromCenter</td>
<td></td>
</tr>
<tr>
<td>BarrierEdges</td>
<td>Gets or sets ID set of the barrier edges in the network.</td>
</tr>
<tr>
<td>Syntax: Int32 BarrierEdges</td>
<td></td>
</tr>
<tr>
<td>BarrierNodes</td>
<td>Gets or sets ID set of the barrier nodes in the network.</td>
</tr>
<tr>
<td>Syntax: Int32 BarrierNodes</td>
<td></td>
</tr>
<tr>
<td>TurnWeightField</td>
<td>Gets or sets the turn cost field and this field must be in the collection of turn cost fields which is defined in the transportation network analysis environment settings.</td>
</tr>
<tr>
<td>Syntax: String TurnWeightField</td>
<td></td>
</tr>
<tr>
<td>WeightName</td>
<td>Gets or sets the name of the WeightFieldInfo object which provides the weight information for analysis. The WeightFieldInfo object must be in the WeightFieldInfos object which is defined in the transportation network analysis environment settings. The default of this property is the first WeightFieldInfo object in the WeightFieldInfos object.</td>
</tr>
<tr>
<td>Syntax: String WeightName</td>
<td></td>
</tr>
<tr>
<td>BarrierEdges</td>
<td>Gets or sets ID list of the barrier edges.</td>
</tr>
<tr>
<td>Syntax: Int32 BarrierEdges</td>
<td></td>
</tr>
<tr>
<td>BarrierNodes</td>
<td>Gets or sets ID list of the barrier nodes.</td>
</tr>
<tr>
<td>Syntax: Int32 BarrierNodes</td>
<td></td>
</tr>
</tbody>
</table>
### 6.9.1 How to Call the Allocation and Location Analysis

When calling location-allocation analysis method to realize location-allocation, the demand sides used in the process of location-allocation analysis are all network nodes, that is, except network nodes corresponding to kinds of centers other network nodes are all regarded as resource demand points in location-allocation analysis. If we want to exclude some nodes, we can set them as barrier points.

**Syntax:**

```csharp
public LocationAnalystResult TransportationAnalyst.FindLocation(LocationAnalystParameter parameter)
```

**Parameters:**

Parameter: location-allocation analysis parameter class object. It provides the necessary parameter information for location-allocation analysis.

**Return Value:**

Returns the result of location-allocation analysis, which is saved in location-allocation analysis result class.
6.9.2 The Outcome of the Allocation and Location Analysis

The result of location-allocation analysis is saved in location-allocation analysis result class, that is, LocationAnalystResult class object. We can get result information from relevant properties of LocationAnalystResult class object. The properties are shown as the following table:

Table 10 LocationAnalystResult class property

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SupplyResults</td>
<td>The array of the SupplyResult objects.</td>
</tr>
<tr>
<td>Syntax: SupplyResult[] SupplyResults</td>
<td></td>
</tr>
<tr>
<td>DemandResults</td>
<td>The array of the DemandResult objects.</td>
</tr>
<tr>
<td>Syntax: DemandResult[] DemandResults</td>
<td></td>
</tr>
</tbody>
</table>

In which, supply center result class, that is, SupplyResult class provides type, ID, the maxima impedance, resource value, etc.

Table 11 SupplyResult class property

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>SupplyCenterType Type.</td>
</tr>
<tr>
<td>Syntax: SupplyCenterType Type</td>
<td></td>
</tr>
<tr>
<td>ID</td>
<td>Gets the ID of SupplyCenter.</td>
</tr>
<tr>
<td>Syntax: Int32 ID</td>
<td></td>
</tr>
<tr>
<td>MaxWeight</td>
<td>Gets the maximum impedance of SupplyCenter. If cost from demand point (edge or node) to the center is bigger than the maximum impedance, the demand point is filtered.</td>
</tr>
<tr>
<td>Syntax: Double MaxWeight</td>
<td></td>
</tr>
<tr>
<td>ResourceValue</td>
<td>The resource's value of SupplyCenter.</td>
</tr>
</tbody>
</table>
### Functions of Traffic Network Analysis

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>Gets the network node ID corresponding to the demand node.</td>
</tr>
<tr>
<td>Syntax: Int32 ID</td>
<td></td>
</tr>
<tr>
<td>SupplyCenterID</td>
<td>Gets the ID of the supply center. It represents which supply center resources got by the demand point are from.</td>
</tr>
<tr>
<td>Syntax: Int32 SupplyCenterID</td>
<td></td>
</tr>
<tr>
<td>ActualResourceValue</td>
<td>Gets the actual allocated resource value for the demand side.</td>
</tr>
<tr>
<td>Syntax: Double ActualResourceValue</td>
<td></td>
</tr>
<tr>
<td>IsEdge</td>
<td>Represents whether the demand side type is edge or not. If the property is set true it represents edge; if false, it represents node. (Here, it's the result of</td>
</tr>
</tbody>
</table>

---

**Table 12 DemandResult class property**

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TotalWeights</td>
<td>Gets the total weight of SupplyCenter.</td>
</tr>
<tr>
<td>Syntax: Double TotalWeights</td>
<td></td>
</tr>
<tr>
<td>DemandCount</td>
<td>Gets the count of demand points which the supply center provides service for.</td>
</tr>
<tr>
<td>Syntax: Int32 DemandCount</td>
<td></td>
</tr>
<tr>
<td>ActualResourceValue</td>
<td>Gets the actual resource supply value from supply center.</td>
</tr>
<tr>
<td>Syntax: Double ActualResourceValue</td>
<td></td>
</tr>
<tr>
<td>AverageWeight</td>
<td>Gets the average weight of the supply center.</td>
</tr>
<tr>
<td>Syntax: Double AverageWeight</td>
<td></td>
</tr>
</tbody>
</table>
Example: As shown in Figure 1, we want to set up a number of post offices for the region. There are 15 candidate locations for the post offices (As shown in Figure 1, the blue boxes represent the candidate locations), and we want to select 7 target locations based on the following conditions: the residents in the region should have access to at least one post office within 30 minutes’ walk, and each post office has a maximum number of residents it can serve. The Location and Allocation analysis will locate these 7 post offices and outline the area that can be served by each post office (as shown in Figure 2, the red points are the post offices located).

Notes: In the following two figures, all the nodes in the network are considered as the residents involved in the Location and Allocation analysis.
Diagram 6 Network and Candidate Locations

Diagram 7 The Result of Location and Allocation Analysis

Concepts in the Allocation and Location Analysis:

SuperMap Objects .NET Technology Documents
The resource supply center: the node data that is used to hold the relative information of the resource supply center (resource amount, the resistance limits, the type and the ID of resource supply center, and so on), which prepare for the analysis.

Resource amount: the largest amount of service or the number of goods could be offered by the center

Resistance limits: to limit the cost between the supply center and the demand points during delivery. The points, which are outside of the reach from the centre even with the largest cost, will be excluded. In a word, the center supply cannot provide service to the excluded points.

Resource supply center type: permanent center, optional center, and improper center. The permanent center is the established facilities (the resource supply sides); optional centers are the location options for building a new center; the improper center will be excluded at all in the analysis.

**The delivery modes:**

The "deliver" mode (from supply side to demand side):

The power station, as the network centre, generates electric to clients by electricity network which is the edges in the network model. In this situation, the energy is distributed from the supply side to the demand side.

The "fetch" mode (from demand side to supply side):

Schools offer education service to school-age children who are the demand sides. Children are allocated along the neighborhood network, requiring the enrollment places from the school.
7

Facility Network Analysis

Using functions offered by class FacilityAnalyst to achieve correspondent network analysis functions. We are going to introduce all the facility network analysis functions provided by class FacilityAnalyst in the flowing sections, as well as their relative concepts.

7.1 Steps to Achieve Facility Network Analysis

1. Build network datasets

2. Build flow direction or class for network data. Facility network analysis needs a network dataset designed for itself. As a result, a network dataset must be built at the very beginning. Finally, use the BuildFacilityNetworkDirections() in the class NetworkBuilder to plant the specific data information for facility network analysis into the network dataset, i.e. building the flow direction for the network dataset which is the basic condition for the facility network analysis. At this stage, you are able to use various functions in the facility network analysis. The BuildFacilityNetworkDirections() of the class NetworkBuilder also allows you to add your class information.

3. The configuration of facility network analysis is to set the attribute FacilityAnalystSetting FacilityAnalyst.AnalystSetting.


5. Use various facility network analysis methods offered by class FacilityAnalyst.
7.2 Building Network Dataset Flow Direction and Class

There are three parts in this chapter to illustrate the procedures of building flow direction and class for the network dataset:

7.2.1 Building Network Dataset Flow Direction

Creating direction for network dataset is realized by BuildFacilityNetworkDirections method in NetworkBuilder class:

Syntax:

```csharp
public static Boolean NetworkBuilder.BuildFacilityNetworkDirections(FacilityAnalystSetting facilityAnalystSetting, Int32[] sourceIDs, Int32[] sinkIDs, String weightName, String nodeTypeField)
```

Parameters:

- FacilityAnalystSetting facilityAnalystSetting: Facility network analysis environment sets the class object. The parameter is FacilityAnalystSetting class object. For contents of the parameter setting, see table 18 which lists the property information.

- sourceIDs: The array of network node ID corresponding to sources. Sources and sinks are used to establish the flow of network datasets. The flow of network datasets is determined by positions of sources and sinks.

- sinkIDs: The array of network node ID corresponding to sinks. Sources and sinks are used to establish the flow of network datasets. The flow of network datasets is determined by positions of sources and sinks.

- weightName: The name of the specified weightFieldInfo object, that is, WeightFieldInfo.Name.

- nodeTypeField: The name of node type field. Types of nodes contain source node, sink node, common node. The field exists in network node dataset. If it doesn't exist, we should create the field.

Table 13 FacilityAnalystSetting class property

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NetworkDataset</td>
<td>Gets or sets the network dataset.</td>
</tr>
</tbody>
</table>

*SuperMap Objects .NET Technology Documents*
Remarks:

The NetworkDataset will represent different network dataset in different steps of the facility analysis. As belows:

1 When building the dataset for the facility analysis:

BuildFacilityNetworkDirections() and BuildFacilityNetworkHierarchies() in NetworkBuilder class both use parameter of FacilityAnalystSetting type, so they relate to the setting of NetworkDataset property in FacilityAnalystSetting type parameter. When creating flow the property is corresponding to the ordinary network dataset; when creating grade the property corresponding to network dataset with flow information.

2 When using kinds of methods in FacilityAnalyst class to conduct facility analysis:

Before using kinds of methods in FacilityAnalyst class to conduct facility analysis, we need set network analysis environment through AnalystSetting property of FacilityAnalyst which is type of FacilityAnalystSetting, so it relates to the setting of NetworkDataset. Here, NetworkDataset represents network dataset which has created flow or has created flow and grade.

<table>
<thead>
<tr>
<th>NodeIDField</th>
<th>Gets or sets node ID field.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Syntax: String NodeIDField</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>EdgeIDField</th>
<th>Gets or sets edge ID field.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Syntax: String EdgeIDField</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>FNodeIDField</th>
<th>Gets or sets the ID field indicating the start node of an edge in the network dataset.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Syntax: String FNodeIDField</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TNodeIDField</th>
<th>Gets or sets the ID field indicating the end node of an edge in the network dataset.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Syntax: String TNodeIDField</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>WeightFieldInfos</th>
<th>Gets or sets the WeightFieldInfos object. WeightFieldInfos is the set of WeightFieldInfo. You can set the weight filed, which represents the cost in the analysis, in the WeightFieldInfo. Two weight fields are provided in the WeightFieldInfo, they are FTWeightField and the TFWeightField. The FTWeightField represents the cost from the start to the end of the edge while the TFWeightField indicates the cost from the end to the start of</th>
</tr>
</thead>
</table>
### 7.2.2 The Result of Building Network Dataset Flow Direction

After building the network dataset direction successfully, the network dataset has a direction. Two kinds of information will be added to the network dataset.

The node type information will be written into the network dataset. There are three types of nodes. They are the source node, sink node and general node. The following table lists the node types and the corresponding values.

<table>
<thead>
<tr>
<th>Number</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>It is neither a source node nor a sink node.</td>
</tr>
</tbody>
</table>

Table 14 The number which represents the network node type.
The flow direction will be written into the network node dataset. The flow direction information is recorded in the flow direction field and represents a certain direction. The numbers and meanings are shown in the following table:

<table>
<thead>
<tr>
<th>Number</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Same with the line direction in digitalization</td>
</tr>
<tr>
<td>1</td>
<td>Opposite to the line direction in digitalization</td>
</tr>
<tr>
<td>2</td>
<td>Loop (it is also called the invalid direction)</td>
</tr>
<tr>
<td>3</td>
<td>Uninitialized direction (it is also called unconnected direction)</td>
</tr>
</tbody>
</table>

The system sets the flow direction of the network dataset according to the source and sink. The flow direction information is automatically built and written into the specified fields. The range of the values is shown in the table above.

**Digitalization direction:**

Figure 1: There are two points A (1, 1) and B (2, 2). The digitalization direction of line AB is northeastern. The digitalization direction of line BA is southwest. The digitalization direction is the order direction of the coordination points along the line. The digitalization direction of line AB is A→B, and the digitalization direction of line BA is B→A.
Figure 2: Take line AB as an example. If A is the source point, the flow direction (suppose the facility is a river) is A→B, so it is the same with the digitalization direction (A→B) of line AB. If A is the sink node, the flow direction is B→A, it is opposite to the digitalization direction (A→B).

Figure 3: take line AB as an example. If A is the source point, the flow direction (suppose the facility is a river) is A→B; If B is the source point, the flow direction (suppose the facility is a river) is B→A; If A and B are both source nodes, the flow direction could be A→B or B→A. This is an invalid flow direction, it is also called loop.
Unconnected edge:

Figure 4: If edge AB is not connected with the source node or sink node, it is called unconnected edge.

Diagram 11 unconnected edge

7.2.3 Building Network Dataset Hierarchy

If the data used for facility analysis has the hierarchy, based on the flow direction, you can use the BuildFacilityNetworkHierarchies() method of the NetworkBuilder class to add the hierarchy data to the network dataset.BuildFacilityNetworkDirections() method.

Syntax:

```csharp
public static Boolean NetworkBuilder.BuildFacilityNetworkHierarchies(FacilityAnalystSetting
```
7 Facility Network Analysis

facilityAnalystSetting, Int32[] sourceIDs, Int32[] sinkIDs, String weightName, String hierarchyField, Boolean isLoopValid)

Parameters:

FacilityAnalystSetting facilityAnalystSetting: The properties of this object have been introduced when building the flow direction, the difference is the network dataset set by FacilityAnalystSetting have built the direction in advance.

sourceIDs: The source network node ID array. Source and sink are used to build the flow direction of the network dataset. The flow direction is defined by the position of the source and sink.

sinkIDs: The sink network node ID array. Source and sink are used to build the flow direction of the network dataset. The flow direction is defined by the position of the source and sink.

weightName: The specified name of the weightFieldInfo object, namely WeightFieldInfo.Name.

hierarchyField: The specified name of hierarchy field. The hierarchy information of the network dataset will be recorded in this field after the hierarchy is built.

isLoopValid: Specifies whether the loop is valid. When the loop is valid this parameter is True, otherwise false.

Return Value:

Returns True, if successful; otherwise, returns False.

After building the hierarchy, the hierarchy information of the network dataset will be recorded in the hierarchy field. The field value is integer. The higher hierarchy, the smaller value. For example, building the river hierarchy, the first level river is recorded as 0, the second level river is recorded as 1 and so on.

7.3 Environment Configuration of the Facility Network Analysis

The environment configuration of the network analysis will directly affect the outcome of the analysis. Parameters used in facility network analysis includes datasets, ID field of nodes, ID field of edges, ID field of the starting point of an edge, ID field of the end point of an edge, weight information, distance tolerance between the point and edge, barrier nodes, barrier edges, flow direction field.

The environment configuration of facility network analysis achieved through attribute AnalystSetting in the class FacilityAnalyst. This attribute is correspondent to class FacilityAnalystSetting. The properties of FacilityAnalystSetting have been introduced in the previous sections. It should be note that the attribute NetworkDataset in the class
FacilityAnalystSetting must be network datasets that has flow direction or network datasets that has both flow direction and class.

### 7.4 Loading the Facility Network Analysis

After setting the environment configuration of the facility network analysis, you have to call the Load () method in FacilityAnalyst class to load the facility network data, making your configuration in effect.

### 7.5 A List of Facility Network Analysis Functions

The following are detail instructions of all the facility network analysis functions offered by class FacilityAnalyst, as well as the relative concepts.

#### 7.5.1 Checking Network Loop

**Syntax:**

```
public Int32[] FacilityAnalyst.CheckLoops()
```

**Return value:**

Returns the ID array of the network loop edges.

#### 7.5.2 Finding Interlinked Loop

There are two methods to search the connected loops:

Method one: Based on the array of the specified network node IDs, search the loops that connect with these nodes.

**Syntax:**

```
public Int32[] FacilityAnalyst.FindLoopsFromNodes(Int32[] nodeIDs)
```

**Parameters:**

nodeIDs: The array of the specified network node IDs.

**Return value:**
Returns the array of the corresponding network edge IDs of loops that connect with the network nodes.

Method Two: Based on the array of the specified network edge IDs, search the loops that connect with these edges.

**Syntax:**

```csharp
public Int32[] FacilityAnalyst.FindLoopsFromEdges(Int32[] edgeIDs)
```

**Parameters:**

- `edgeIDs`: The array of the specified network edge IDs.

**Return value:**

Returns the array of the corresponding network edges of loops that connect with the specified network edges.

### 7.5.3 Finding Unconnected Edges

**The methods of searching the unconnected edges:**

Method One: Based on the array of the specified network node IDs, search the edges that do not connect with these nodes.

**Syntax:**

```csharp
public Int32[] FacilityAnalyst.FindUnconnectedEdgesFromNodes(Int32[] nodeIDs)
```

**Parameters:**

- `nodeIDs`: The array of the specified network node IDs.

**Return Value:**

Returns the network edges that do not connect with the specified network nodes.

Method Two: Based on the array of the specified network edge IDs, search the edges that do not connect with these nodes.

**Syntax:**

```csharp
public Int32[] FacilityAnalyst.FindUnconnectedEdgesFromEdges(Int32[] edgeIDs)
```

**Parameters:**

- `edgeIDs`: The array of the specified network edge IDs.
Return value:
Returns the network edges that do not connect with the specified network edges.

7.5.4 Finding Connected Edges

The methods of searching the connected edges:

Method One: Based on the array of the specified Network node IDs, search the edges that connect with these nodes.

Syntax:
public Int32[] FacilityAnalyst. FindConnectedEdgesFromNodes (Int32[] nodeIDs)

Parameters:
nodeIDs: The array of the specified network node IDs.

Return value:
Returns the network edges that connect with the specified network nodes.

Method Two: Based on the specified network edge IDs, search the edges that connect with these nodes.

Syntax:
public Int32[] FacilityAnalyst. FindConnectedEdgesFromEdges (Int32[] edgeIDs)

Parameters:
edgeIDs: The array of the specified network edge IDs.

Return value:
Returns the network edges that connect with the specified network edges.

7.5.5 Upstream Tracing

Concept——Upstream:

As shown in the illustration: Suppose water flows in the network along the specified flow direction. Q is a sink and water from Node A, B, C, L, E, F, and G all goes to Q, so these nodes are called the upstream (nodes) of Q. Similarly, Edge AB, BC, CL, LQ, EB, FG, and GL are all upstream (edges) for Q.
For Edge LQ, Node A, B, C, L, E, F, and G are all its upstream nodes; and Edge AB, BC, CL, EB, FG, and GL are all upstream edges for Edge LQ.

Diagram 12 Upstream

Upstream Tracing Analysis Method one: Get and return all the upstream edges for the specified node.

**Syntax:**

```csharp
public Int32[] FacilityAnalyst.TraceUpFromNode(Int32 nodeID, String weightName, Boolean isLoopValid)
```

**Parameters:**

- `nodeID`: The ID of the specified node.
- `weightName`: the name of the WeightFieldInfo object, that is, WeightFieldInfo.Name.
- `isLoopValid`: Specifies whether the loop is valid or not.

**Return value:**

Returns the ID array of upstream edges for the specified node.

**Example:** As shown in the illustration, when Q is specified as the sink node for the upstream tracing analysis, the ID array for Edge AB, BC, CL, LQ, EB, FG and GL will be returned as the result.

Upstream Tracing Analysis Method 2: Get and return all the upstream edges for the specified edge.
Syntax:

public Int32[] FacilityAnalyst.TraceUpFromEdge(Int32 edgeID, String weightName, Boolean isLoopValid)

Parameters:

edgeID: The ID of the specified edge.

weightName: the name of the WeightFieldInfo object, that is, WeightFieldInfo.Name.

isLoopValid: Specifies whether the loop is valid or not.

Return value:

Returns the ID array of upstream edges for the specified edge.

Example: As shown in the illustration, when LQ is specified as the edge for the upstream tracing analysis, the ID array for Edge AB, BC, CL, EB, FG and GL will be returned as the result.

7.5.6 Downstream Tracing

Concept—Downstream:

As shown in the illustration: Suppose water flows in the network along the specified flow direction. Q is a source node and water from Source Q flows to Node A, B, C, L, E, F, and G. So these nodes are called the downstream (nodes) of Q. Similarly, Edge BA, CB, LC, QL, BE, GF, and LG are all downstream (edges) for Q.

For Edge QL, Node A, B, C, L, E, F, and G are all its downstream nodes; and Edge BA, CB, LC, BE, GF, and LG are all downstream edges for Edge QL.
Downstream Tracing Analysis Method 1: Get and return all the downstream edges for the specified node.

Syntax:

```csharp
public Int32[] FacilityAnalyst.TraceDownFromNode(Int32 nodeID, String weightName,Boolean isLoopValid)
```

Parameters:

- `nodeID`: The ID of the specified node.
- `weightName`: the name of the `WeightFieldInfo` object, that is, `WeightFieldInfo.Name`.
- `isLoopValid`: Specifies whether the loop is valid or not.

Return value:

Returns the ID array of downstream edges for the specified node.

As shown in the illustration, when Q is specified as the source node for the downstream tracing analysis, the ID array for Edge BA, CB, LC, QL, BE, GF, and LG will be returned as the result.

Downstream Tracing Analysis Method 2: Get and return all the upstream edges for the specified edge.

Syntax:

```csharp
public Int32[] FacilityAnalyst.TraceDownFromEdge(Int32 edgeID, String weightName,Boolean
```

Diagram 13 Downstream
isLoopValid)

**Parameters:**

dedgeID: The ID of the specified edge.

weightName: the name of the WeightFieldInfo object, that is, WeightFieldInfo.Name.

isLoopValid: Specifies whether the loop is valid or not.

**Return value:**

Returns the ID array of downstream edges for the specified edge.

As shown in the illustration, when QL is specified as the edge for the downstream tracing analysis, the ID array for BA, CB, LC, BE, GF, and LG will be returned as the result.

### 7.5.7 Common Upstream Tracing

Common Upstream Tracing Analysis Method 1: Get the common upstream edges for the nodes in the specified ID array.

**Syntax:**

```csharp
public Int32[] FacilityAnalyst.FindCommonAncestorsFromNodes(Int32[] nodeIDs, String weightName, Boolean isLoopValid)
```

**Parameters:**

nodeIDs: The ID of the specified node.

weightName: the name of the WeightFieldInfo object, that is, WeightFieldInfo.Name.

isLoopValid: Specifies whether the loop is valid or not.

**Return value:**

Returns the ID array of upstream edges for the specified node.

As shown in the illustration, the ID array for Edge GL and FG will be returned as they are the common upstream edges for Node Q and P.
Diagram 14 Common Upstream

Common Upstream Tracing Analysis Method 2: Get the common upstream edges for the edges in the specified ID array.

**Syntax:**

```csharp
public Int32[] FacilityAnalyst.FindCommonAncestorsFromEdges(Int32[] edgeIDs, String weightName, Boolean isLoopValid)
```

**Parameters:**

- `edgeIDs`: The ID of the specified edge.
- `weightName`: the name of the WeightFieldInfo object, that is, `WeightFieldInfo.Name`.
- `isLoopValid`: Specifies whether the loop is valid or not.

**Return value:**

Returns the ID array of upstream edges for the specified edge.

As shown in the illustration, the ID array for Edge GL and FG will be returned as they are the common upstream edges for Edge LQ and MP.
### 7.5.8 Common Downstream Tracing

Common Downstream Tracing Analysis Method 1: Get the common downstream edges for the nodes in the specified ID array.

**Syntax:**

```csharp
public Int32[] FacilityAnalyst.FindCommonAncestorsFromNodes(Int32[] nodeIDs, String weightName, Boolean isLoopValid)
```

**Parameters:**

- `nodeIDs`: The ID of the specified node.
- `weightName`: the name of the WeightFieldInfo object, that is, WeightFieldInfo.Name.
- `isLoopValid`: Specifies whether the loop is valid or not.

**Return Value:**

Returns the ID array of common upstream edges for the specified node.

As shown in the illustration, the ID array for Edge BA, CB, LC, BE, LG and GF will be returned as they are the common downstream edges for Node Q and P.

![Diagram 15 Common Downstream](image)
Common Downstream Tracing Analysis Method 2: Get the common downstream edges for the edges in the specified ID array.

**Syntax:**

```csharp
public Int32[] FacilityAnalyst.FindCommonAncestorsFromEdges(Int32[] edgeIDs, String weightName, Boolean isLoopValid)
```

**Parameters:**

- `nodeIDs`: The ID of the specified edge.
- `weightName`: the name of the WeightFieldInfo object, that is, WeightFieldInfo.Name.
- `isLoopValid`: Specifies whether the loop is valid or not.

**Return Value:**

Returns the ID array of common upstream edges for the specified edge.

As shown in the illustration, the ID array for Edge BA, CB, LC, BE, LG and GF will be returned as they are the common downstream edges for Edge LQ and MP.

### 7.5.9 Upstream Path Analysis

Upstream Path Analysis Method 1: Find the shortest upstream path or the upstream path with the least edges for the node of the specified ID.

**Syntax:**

```csharp
public FacilityPathResult FacilityAnalyst.FindPathUpFromNode(Int32 nodeID, String weightName)
```

**Parameters:**

- `nodeID`: The ID of the specified node.
- `weightName`: the name of the WeightFieldInfo object, that is, WeightFieldInfo.Name.

**Return Value:**

Returns the result of upstream path analysis which is saved in class of FacilityPathResult.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nodes</td>
<td>Get the ID array of the nodes that are passed by the result</td>
</tr>
</tbody>
</table>

*SuperMap Objects .NET Technology Documents*
As shown in the figure, we want to find the shortest upstream path for Node Q, and return the path. We can find that the upstream paths for Node Q end at Node A, E, and F, and FGLQ is the shortest upstream path.

**Diagram 16 Upstream Path**

Upstream Path Analysis Method 2: Find the shortest upstream path or the upstream path with the least edges for the edge of the specified ID.

**Syntax:**

```csharp
public FacilityPathResult FacilityAnalyst.FindPathUpFromEdge(Int32 edgeID, String weightName)
```

**Parameters:**

- `edgeID`: The ID of the specified edge.
- `weightName`: The name of the WeightFieldInfo object, that is, `WeightFieldInfo.Name`.

SuperMap Objects .NET Technology Documents
Return Value:

Returns the result of upstream path analysis which is saved in the class of FacilityPathResult.

As shown in the illustration, we want to find the shortest upstream path for Edge LQ, and return the path.

We can find that the upstream paths for Edge LQ end at Node A, E, and F, and FGL is the shortest upstream path.

7.5.10 Downstream Path Analysis

Downstream Path Analysis Method 1: Find the shortest downstream path or the downstream path with the least edges for the node of the specified ID.

Syntax:

public FacilityPathResult FacilityAnalyst.FindPathDownFromNode(Int32 nodeID, String weightName)

Parameters:

nodeID: The ID of the specified node.

weightName: the name of the WeightFieldInfo object, that is, WeightFieldInfo.Name.

Return Value:

Returns the result of the downstream path analysis which is saved in the class of FacilityPathResult.

As shown in the illustration, we want to find the shortest downstream path for Node Q, and return the path.

We can find that the downstream paths for Node Q end at Node A, E, and F, and QLGF is the shortest downstream path.
Diagram 17 Downstream Path

Downstream Path Analysis Method 2: Find the shortest downstream path or the downstream path with the least edges for the edge of the specified ID.

**Syntax:**

```csharp
public FacilityPathResult FacilityAnalyst.FindPathDownFromEdge(Int32 edgeID, String weightName, Boolean hasLeastEdgeCodfdfunt)
```

**Parameters:**

edgeID: The ID of the specified edge.

weightName: the name of the WeightFieldInfo object, that is, WeightFieldInfo.Name.

**Return Value**

Returns the result of downstream path analysis which is saved in the class of FacilityPathResult.

As shown in the illustration, we want to find the shortest downstream path for Edge QL, and return the path.

We can find that the downstream paths for Edge QL end at Node A, E, and F, and LGF is the shortest downstream path.
7.5.11 Facility Network Path Analysis

Facility Analysis Method 1: finds the path which is the shortest or has the least edges between the specified from-node and to-node.

Syntax:
public FacilityPathResult FacilityAnalyst.FindPathFromNodes(Int32 startNodeID, Int32 endNodeID, String weightName)

Parameters:
startNodeID: From-node ID.
endNodeID: To-node ID.
weightName: the name of the WeightFieldInfo object, that is, WeightFieldInfo.Name.

Return Value:
Returns the result of facility network path analysis which is saved in the class of FacilityPathResult.

Facility Analysis Method 2: finds the path which is the shortest or has the least edges between the specified from-edge and to-edge.

Syntax:
public FacilityPathResult FacilityAnalyst.FindPathFromEdges(Int32 startEdgeID, Int32 endEdgeID, String weightName, Boolean hasLeastEdgeCount)

Parameters:
startEdgeID: From-edge ID.
endEdgeID: To-edge ID.
weightName: the name of the WeightFieldInfo object, that is, WeightFieldInfo.Name.

Return Value:
Returns the result of facility network path analysis which is saved in the class of FacilityPathResult.