VISSIM 4.30-00

COM Interface Manual

PTV
Planung
Transport
Verkehr AG

Copyright

Planung Transport Verkehr AG
Stumpfstraße 1
D-76131 Karlsruhe
Germany

April 2007

Table of Contents

1 Introduction

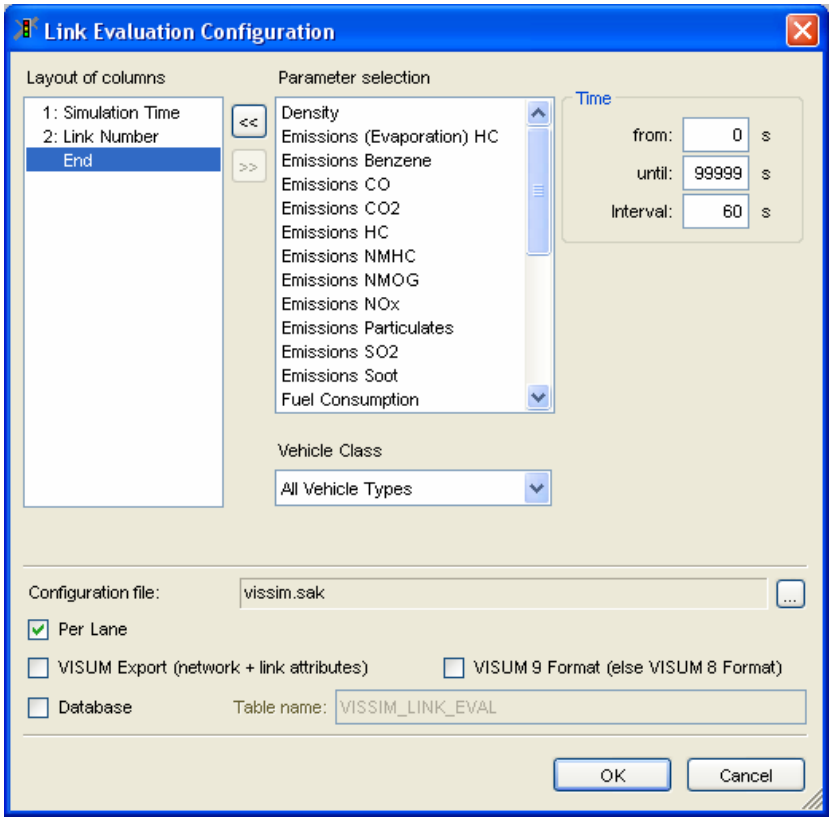
VISSIM can be applied as a powerful tool in analysis of a huge variety of transportation problems. Occasionally projects will require extensive pre- or post-processing or numerous scenarios to be investigated. For these cases VISSIM can be run from within other applications serving as a toolbox for transportation planning algorithms. Access to model data and simulations is provided through a COM interface, which allows VISSIM to work as an Automation Server and to export the objects, methods and properties described in this document. The VISSIM COM interface supports Microsoft Automation, so you can use any of the RAD (Rapid Application Development) tools ranging from scripting languages like Visual Basic Script or Java Script to programming environments like Visual C++ or Visual J++. The examples presented in this manual are mainly in Visual Basic, see page 199 for a little introduction. You will also find a short introduction on how to use it with Visual C++ on page 209.

An introductory Visual Basic example will illustrate the flexibility of this approach. Assume that we want to run several simulations with different random seeds and get their respective link evaluations. You can use Excel and VBA (Visual Basic for Applications) to load the network, set the different random seeds and start the process. For example, if you want to use the network “fixed_time.inp” and to run four simulations with the random seeds 10, 20, 30 and 42 you can edit the following Excel sheet:

random seed	simulation file
10	G:\PTV\DATA\FIXED_TIME.INP
20	
30	
42	

You can also insert a button “START” (from the toolbox) to link with the Visual Basic code presented at the end of this section:

To get the link evaluations by running the Excel macro you have to create a *.SAK file within the same directory beforehand. You may use the VISSIM *Link Evaluation* Dialog (or the LinkEvaluation interface) for this purpose:



It is also necessary to check the box for the link evaluations in the Evaluations → Files Dialog and to save the options in a *.INI file, for example LINK_EVAL.INI, with SAVE AS… from the View menu.

1.1 Introductory Example

We will now present the Visual Basic code that will run four simulations, accessing the Excel sheet to get the network file name and the respective random seeds. The code is defined in a procedure called RandomSeed2VISSIM that must be called from the “OnClick” command of the START button in the Excel sheet. The lines starting with an apostrophe mark (‘) are comments that explain the code lines or parts.

```
SUB RandomSeed2VISSIM()
' Declare VISSIM COM types for a Vissim and a Simulation object
DIM Vissim AS Vissim
DIM Simulation AS Simulation
' Declare further types
DIM SimulationFile AS String   ' Name of the network file (with the full path)
DIM RandomSeed AS Integer  ' Current random seed
DIM RunIndex AS Integer        ' Simulation running index
' Start Vissim and create an instance of a Vissim object and a Simulation object
SET Vissim = CreateObject("VISSIM.Vissim")
SET Simulation = Vissim.Simulation
' Get the network file name
Sheets("VISSIM").Select                  ' Select the example sheet named VISSIM
Range("C2").Select                         ' Select the cell with the network file name
SimulationFile = Selection.Value    ' Get the network file name
' Load the network and the *.ini file
Vissim.LoadNet SimulationFile
Vissim.LoadLayout "link_eval.ini"
' Initialize simulation values
Simulation.Period = 100        ' 100 second simulations
Simulation.Resolution = 1    ' 1 step per second resolution
RunIndex = 0                         ' Simulation running index initialization
' Loop of simulation runs
Range("A2").Select       ' Select the first random seed cell
WHILE Selection <> ""    ' Run until no more random seeds are available
   ' Get current random seed from the current selected cell
   RandomSeed = Selection.Value
   ' Set simulation parameters for next simulation
   Simulation.RunIndex = RunIndex  ' Set the simulation run index
   Simulation.Comment = "Random Seed = " & RandomSeed  ' Set simulation comment
   Simulation.RandomSeed = RandomSeed  ' Set random seed
   ' Run simulation continuously
   Simulation.RunContinuous
  ' Initilize next simulation values
   ActiveCell.Offset(1, 0).Select    ' Select the next random seed cell
   RunIndex = RunIndex + 1         ' Next simulation running index
WEND
END SUB
```

After the creation of a VISSIM object the code demonstrates how to obtain references to other VISSIM data objects (in this case a Simulation object) and how to access their properties and methods.

In a similar way it is possible to use the Visual Basic programming environment to create a desktop application to control simulation runs having access not only to the random seeds but also to the other simulation parameters.

It is necessary that VISSIM has been registered to the Windows system before it can be started automatically from a COM client. You can do this registration manually (see page 13) or simply by starting VISSIM once which will cause the program to register itself. Furthermore a reference to the VISSIM COM Library can be set within the programming environment to help code editing and improve performance by early binding. In Excel's Basic Editor this can be done from the EXTRAS menu REFERENCES….

When using early binding (refer to page 206 for more information) a reference to the VISSIM COM server type library must be set within the VB programming environment. This can cause some conflicts when installing newer versions of VISSIM, which could have modified the type library. Unselecting and selecting again the reference to the VISSIM COM server forces VB to reinterpret the type library. Please refer to the annex "Tips and hints" on page 239 for more details about VISSIM versions and its COM server interface

1.2 License and Registration

The appropriate VISSIM license is necessary to use the VISSIM COM server. This license actives the self registering procedure of all COM interfaces, which will be called automatically when VISSIM is installed, and allows the instantiation and use of the VISSIM COM objects from other programs or development environments, like Visual Basic or Visual C++.

A manual registration of the COM Server is also possible using the following parameters from the command line:

```
VISSIM -RegServer
VISSIM -UnregServer
```

Calling VISSIM with one of this two parameters doesn't start the main VISSIM window. It registers / unregisters silently the VISSIM COM Server and its interfaces. A dialog message will show up with an error message if the registration/unregistration failed (detailed error information is also written to the VISSIM.ERR file in this case):

Please refer to the annex for more information about the necessary Registry entries to use VISSIM as a COM server.

1.3 Instances

Surely you are used to working with several instances of VISSIM by simply starting the program several times. When working with COM applications VISSIM's behavior is exactly specified as follows:

- if there are one or more running instances of VISSIM, a COM application connects to the first started VISSIM instance.
- if there is no instance running, creating a COM object automatically invokes a new instance of VISSIM (see page 20 on how to create a Vissim object).
- by default each VISSIM instance can be used as server for more than one COM application. If more COM applications are started later, each of them will link the first started VISSIM instance. Use the Embedded mode (see below) to grant that two COM programs do not access the same VISSIM instance simultaneously.

In some cases it may be useful to allow one VISSIM instance to be the server for several COM programs. You can control VISSIM behavior using these parameters:

- Automation:with this parameter you start a VISSIM instance which will behave exactly as described above (default behavior). A VISSIM instance started that way serves **all** COM applications until it is closed manually. The VISSIM instance must be closed explicitly from the COM client.

While a client is connected to the instance no other COM applications should be started. Automation servers are only useful if you don't want to run several COM applications simultaneously.

- Embedded:with this parameter each VISSIM instance can be used as server for only one COM application. If more COM applications are started later, each of them opens a new VISSIM instance. That way it's granted that two COM programs do not access the same VISSIM instance simultaneously. Additionally the instance will be close automatically if the COM-object has been released, explicitly or by ending the COM client.

It is possible instanciate a concrete version of VISSIM if several VISSIM versions are installed on your computer. Please refer to the Vissim object chapter on page 20 for this purpose)

1.4 Conventions

This manual follows an uniform pattern for the description of the objects and interface. Nevertheless some conventions should be taken in account before reading this manual:

- Object Model

  The name of objects representing single entities or elements in VISSIM are singular, like Vissim, Net, Simulation, Link, SignalHead, ... and the name of the objects representing a set (collections, lists, arrays) of other objects are in plural, like Links, SignalHeads, ...

- Description of properties and methods:

  The full signature with a description and examples is presented for each property and method of every interface, making it self-explaining. The following notation is used for the attributes and type of the parameters:

[in] :	input parameter
[out, retval] :	output parameter
unsigned char :	A 8-bit integer type (0 to 255)
long :	A 32-bit integer type (-2,147,483,648 to 2,147,483,647)
double:	A 64-bits floating point type (-1.7E308 to 1.7E308)
BSTR:	A 16-bit string type
VARIANT:	A data type capable of representing various different types.
IObjekt:	The interface of a VISSIM COM object; for example ILink

  Reference types (pointer types or simply pointers) are marked with an asterisk (*) operator. A reference type is one that points (refers) to one specific instance of a type. For example long*, ILink*, VARIANT*.

See a description of the IDL (Interface Definition Language) for more details.

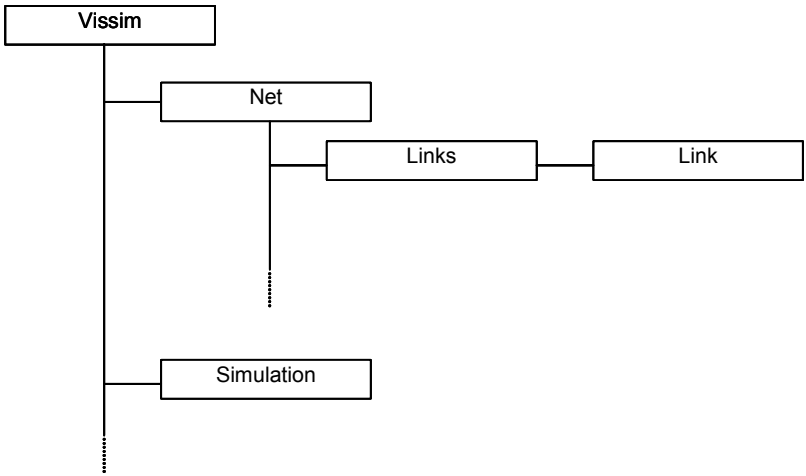
No int type is being used in the interfaces, because of the language mapping to long in Visual Basic and because of the change in size across different platforms.

Visual Basic examples:

All examples in this manual are written in the Visual Basic (with the exception of a C++ section on page 209). The language keywords are written in capital keys, for example DIM, SET, FOR EACH … NEXT, … . See a program example on the section below.

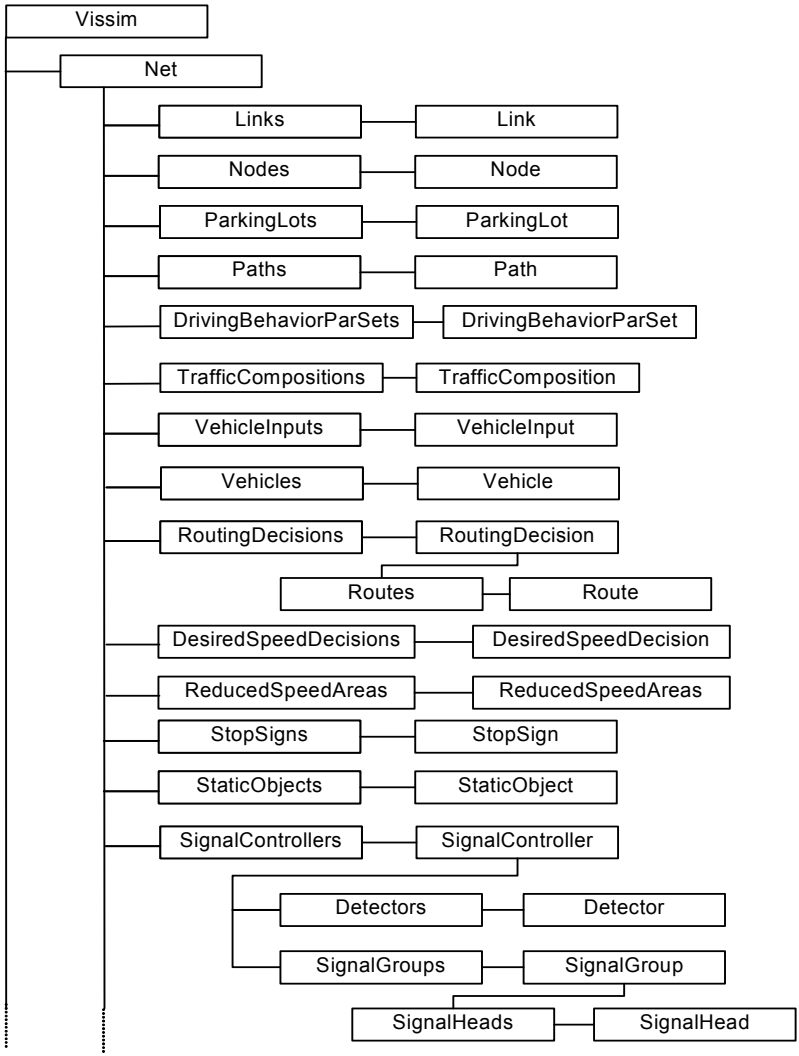
2 Object Model

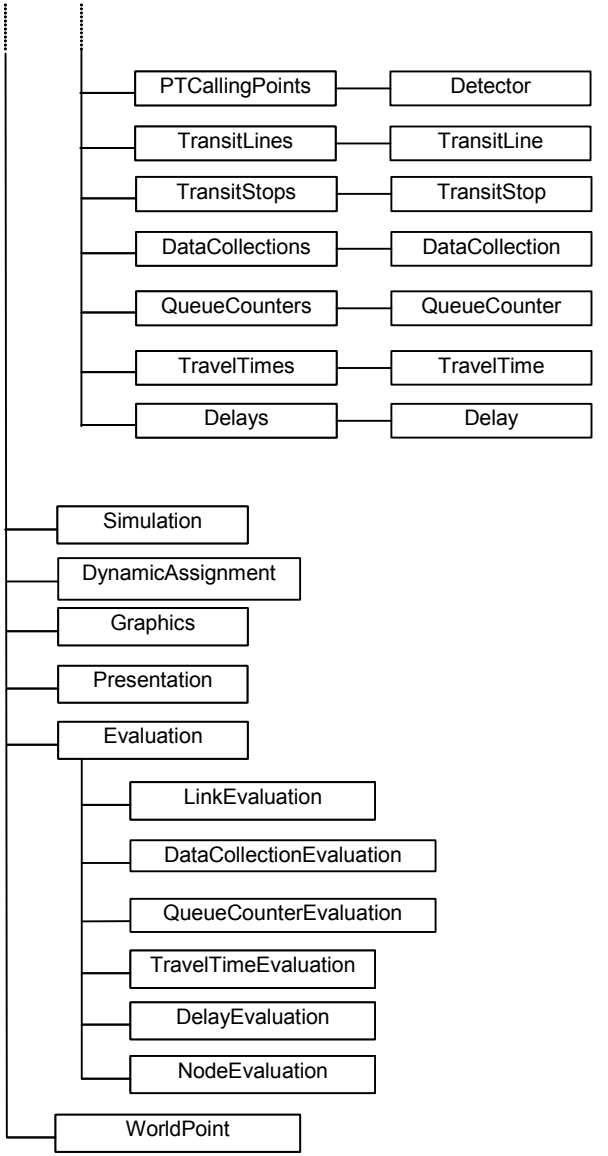
The VISSIM COM object model is based on a strict object hierarchy. To access the different lower-level objects, e.g. a Link object of a Net object, you have to follow this hierarchy. Vissim is the highest object; all other objects belong to Vissim. The following figure illustrates some of the object instantiation dependences (see the model overview on page 18 for the complete hierarchy):



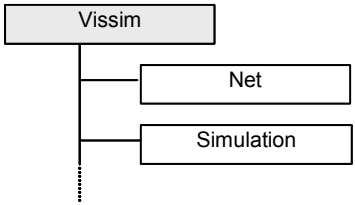
Collections are a special object type; they serve as a container for single objects and are used to enumerate network elements. As a rule their name is in plural. Two examples are the objects Links and Vehicles. Visual Basic provides a special language element For Each ... Next to iterate through a collection; see page 202 for more details.

2.1 Model Overview




PTCallingPoints
Detector
TransitLines
TransitLine
TransitStops
TransitStop
DataCollections
DataCollection
QueueCounters
QueueCounter
TravelTimes
TravelTime
Delays
Delay
Simulation
DynamicAssignment
Graphics
Presentation
Evaluation
LinkEvaluation
DataCollectionEvaluation
QueueCounterEvaluation
TravelTimeEvaluation
DelayEvaluation
NodeEvaluation
WorldPoint

2.2 Vissim



Vissim is the highest object of the model; all other objects belong to Vissim and can only be instantiated through the IVissim interface. With the creation of a Vissim object the COM-Applications gains access to the first VISSIM instance running or, without any instance present, a new instance is started (see page 14).

The interface IVissim allows, besides other things, to load networks and to create second level objects like Net or Simulation.

Examples

- Creation of a Vissim Object:

```
DIM vissim AS Vissim
SET vissim = NEW Vissim
```

  - Alternatively, the CreateObject function (specific of VBScript) can be used with the identifier string for the Vissim object “VISSIM.Vissim.420”, allowing the instantiation of concrete VISSIM versions. VISSIM stands for VISSIM-COMServer, Vissim for the Vissim class object and 420 for the VISSIM version:

```
SET vissim = CreateObject ("VISSIM.Vissim.420")
```

  - If you have installed and registered an earlier version of VISSIM 4.20 use the string “VISSIM.Vissim.1” to instanciate it. If you obviate the version number the newest registered version will be instanciated:

```
SET vissim = CreateObject ("VISSIM.Vissim")
```


Please refer to the Visual Basic issues on page 206 for more concrete information about the difference between NEW and CreateObject

- Deletion of a Vissim Object

  In Visual Basic and VBScript an object is deleted by assigning the language keyword “Nothing”:

```
SET vissim = NOTHING
```

  In this case, if the Automation mode is being used, the VISSIM application itself won’t be closed (compare to the method Exit below). In Embedded mode has the same effect as calling vissim.Exit().

Depending on the used client environment (for example Visual Basic Scripting or Excel) the working directory of VISSIM can be set to the system directory. If a vissim.ini file exists in this directory it is used for the initialization (of window position, view and evaluation settings).

2.2.1 Properties of the IVissim Interface

Net ([out, retval] INet **ppNet)

Instantiates a Net object, that gives access to the network functionality (see page 26).

Parameters

```
[out, retval] INet **ppNet: returned Net object
```

Example

```
DIM net AS Net
SET net = vissim.Net
```

Simulation ([out, retval] ISimulation **ppSimulation)

Instantiates a Simulation object, that gives access to the simulation functionality (see page 30).

Parameters

```
[out, retval] ISimulation **ppSimulation: returned Simulation object
```

Example

```
DIM simulation AS Simulation
SET simulation = vissim.Simulation
```

Graphics ([out, retval] IGraphics **ppGraphics)

Instantiates a Graphics object, that gives access to the graphics options (see page 43).

Parameters

```
[out, retval] IGraphics **ppGraphics :    returned Graphics object
```

Example

```
DIM graphics AS Graphics
SET graphics = vissim.Graphics
```

Evaluation ([out, retval] IEvaluation **ppEvaluation)

Instantiates an Evaluation object, that gives access to the evaluation options (see page 46).

Parameters

```
[out, retval] IEvaluation **ppEvaluation :                returned Evaluation object
```

Example

```
DIM evaluation AS Evaluation
SET evaluation = vissim. Evaluation
```

AttValue ([in] BSTR Attribute, [out, retval] VARIANT *pValue)

Returns a VISSIM general option. Please get the language independent attribute tags from the table at the end of this section.

Parameters

```
[in] BSTR Attribute :            attribute name (see below)
[out, retval] VARIANT *pValue :  returned value of the attribute
```

Example

```
distance_unit1 = vissim.AttValue(„UNITDISTANCE1“)
```

AttValue ([in] BSTR Attribute, [in] VARIANT Value)

Sets a VISSIM general option. Please get the language independent attribute tags from the table at the end of this section.

Parameters

```
[in] BSTR Attribute :          attribute name (see below)
[in] VARIANT Value :           new attribute value. (type according to the attribute)
```

Example

```
vissim.AttValue(„UNITDISTANCE1“) = 1  'in feet
```

Attribute outline :

R	W	Attribute	Description
✓	✓	MENU	Enable/Disable the main menu.
✓	✓	TOOLBAR	Enable/Disable all toolbars except Zoom (File, Selection, Run Control, Network Elements, Animation, Test, and Simulation).
✓		VERSION	VISSIM version in text format.
✓	✓	UNITDISTANCE1	0 = [m], 1 = [ft]
✓	✓	UNITDISTANCE2	0 = [km], 1 = [mi]
✓	✓	UNITSPEED	0 = [km/h], 1 = [mph]
✓	✓	UNITACCEL	0 = [m/s²], 1 = [ft/s²]

2.2.2 Methods of the IVissim Interface

New ()

Creates a new empty network.

Example

```
SET vissim = NEW Vissim
vissim.New
```

LoadNet ([in, defaultvalue(“”)] BSTR NetPath, [in, defaultvalue(0)] BYTE Additive)

Loads the VISSIM network specified within the string NetPath.

Parameters

```
[in] BSTR NetPath : path + filename (*.inp) of the network file to be loaded. This
parameter is optional. If no path is passed a browser file dialog will show up.
```

[in] BYTE Additve : if the value is different than 0 the file will be read additionally. This parameter is optional with 0 as default value (non additive reading form).

Example

```
SET vissim = NEW Vissim
vissim.LoadNet "c:\vissim\data\example.inp"
vissim.LoadNet "c:\vissim\data\example_bis.inp", 1  'read additionally
```

SaveNet ()

Saves the network under the same name.

Example

```
SET vissim = NEW Vissim
vissim.LoadNet "c:\vissim\data\example.inp"
vissim.SaveNet
```

SaveNetAs ([in, defaultvalue("")] BSTR NetPath)

Saves the network under the name specified in the string NetPath

Parameters

[in] BSTR NetPath : path + filename (*.inp) where the network will be saved. If an empty or no path is passed a browser file dialog will show up.

Example

```
SET vissim = NEW Vissim
vissim.LoadNet "c:\vissim\data\example.inp"
vissim.SaveNetAs "c:\vissim\data\example_bis.inp"
```

ImportANM ([in, defaultvalue("")] BSTR NetPath)

Imports a network in ANM format

Parameters

[in] BSTR NetPath : path + filename (*.anm) of the network file to be loaded.

Example

```
SET vissim = NEW Vissim
vissim.ImportANM "c:\vissim\data\example.anm"
```

LoadLayout ([in, defaultvalue("")] BSTR LayoutPath)

Loads a VISSIM layout file (*.INI) specified in the string LayoutPath. If no or an empty LayoutPath parameter is given the browser file dialog appears.

Parameters

[in] BSTR LayoutPath : path + filename (*.ini) of the layout file to be read

Example

```
SET vissim = NEW Vissim
vissim.LoadLayout "c:\vissim\data\example.ini"
```

SaveLayout ([in, defaultvalue("")] BSTR LayoutPath)

Saves the current VISSIM layout into a file (*.INI) specified in the string LayoutPath. If an empty LayoutPath parameter is given the default name "vissim.ini" will be used. If no path is passed the browser file dialog appears.

Parameters

```
[in] BSTR LayoutPath :          path + filename (*.ini)
```

Example

```
SET vissim = NEW Vissim
vissim.SaveLayout "c:\vissim\data\example.ini"
```

Exit ()

Exits the VISSIM program and implicitly deletes the Vissim object instance (also in Automation mode. See 14).

Example

```
SET vissim = NEW Vissim
vissim.Exit
```

ShowMaximized ()

The VISSIM main window is displayed with maximum size.

ShowMinimized ()

The VISSIM main window is only visible in the task bar.

GetWindow ([out] VARIANT *Top, [out] VARIANT *Left, [out] VARIANT *Bottom, [out] VARIANT *Right)

Retrieve the position of the VISSIM main window. The dimensions are given in screen coordinates that are relative to the upper-left corner of the screen.

Parameters

```
[out] VARIANT *Top :     the screen coordinate of the top edge
[out] VARIANT *Left :    the screen coordinate of the left edge
[out] VARIANT *Bottom :  the screen coordinate of the bottom edge
[out] VARIANT *Right :   the screen coordinate of the right edge
```

Example

```
SET vissim = NEW Vissim
vissim.GetWindow top, left, bottom, right
```

SetWindow ([in] VARIANT Top, [in] VARIANT Left, [in] VARIANT Bottom, [in] VARIANT Right)

Set the position of the VISSIM main window. The dimensions are given in screen coordinates that are relative to the upper-left corner of the screen.

Parameters

```
[in] VARIANT Top :       the screen coordinate of the top edge
[in] VARIANT Left :      the screen coordinate of the left edge
[in] VARIANT Bottom :    the screen coordinate of the bottom edge
[in] VARIANT Right :     the screen coordinate of the right edge
```

Example

```
SET vissim = NEW Vissim
vissim.SetWindow 0, 0, 800, 1000
```

NewWorldPoint([in, defaultvalue(0.0)] double X, [in, defaultvalue(0.0)] double Y, [in, defaultvalue(0.0)] double Z, [out, retval] IWorldPoint **ppWorldPoint)

Method for the creation of WorldPoint objects.

Parameters

```
[out, retval] IWorldPoint **ppWorldPoint :                returned WorldPoint object
```

Example

```
DIM wp AS WorldPoint
SET wp = vissim.NewWorldPoint(100.0, 100.0, 100.0)
SET so = vissim.Net.StaticsObjects.GetStaticObjectByCoord(wp)
```

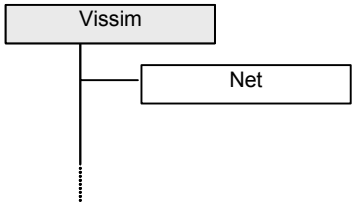
DoEvents()

Allows VISSIM to process its queued events. Only useful when executing scripts from VISSIM self.

Example

```
Set vis = CreateObject("VISSIM.Vissim")
set sim = vis.simulation
for i = 1 to (sim.Period * sim.resolution)
  sim.runsinglestep
  vis.doevents
next
```

2.3 Net



The Net object belongs to Vissim and can be accessed through the property Net of the IVissim interface. It gives access to the network objects like links, signal controllers or vehicles. VISSIM is a single project program, i.e. it allows to work with no more than one network at a time. Therefore, a Net instance always references the currently opened network of its Vissim instance (see example below).

Example

```
DIM vissim AS Vissim
DIM net1, net2 AS Net
SET vissim = NEW Vissim
SET net1 = vissim.Net
vissim.LoadNet "c:\vissim\daten\example.inp"
SET net2 = vissim.Net
```

The objects „Net1“ und „Net2“ refer to the same network, that is to “example.inp“.

2.3.1 Properties of the INet Interface

Name ([out, retval] BSTR *pName)

Returns the simulation’s comment.

Parameters

```
[out, retval] BSTR *pName :      returned name.
```

Example

```
name = net.Name
```

Name ([in] BSTR Name)

Sets the simulation’s comment.

Parameters

```
[in] BSTR Name : new name.
```

Example

```
net.Name = „Barcelona Eixample Sim1“
```

Links ([out, retval] ILinks **ppLinks)

Creates an instance of a Links object (see page 51), that gives individual access to the link elements of the network.

Parameters

```
[out, retval] ILinks **ppLinks : returned Links object
```

Example

```
DIM links AS Links
SET links = net.Links
```

Nodes ([out, retval] INodes **ppNodes)

Creates an instance of a Nodes object (see page 59), that gives individual access to the node elements of the network.

Parameters

```
[out, retval] INodes **ppNodes : returned Nodes object
```

Example

```
DIM nodes AS Nodes
SET nodes = net.Nodes
```

Paths ([out, retval] IPaths **ppPaths)

Creates an instance of a Paths object (see page 69), that gives individual access to the path elements of the network inserted through the IPaths interface..

Parameters

```
[out, retval] IPaths **ppPaths : returned Paths object
```

Example

```
DIM paths AS Paths
SET paths = net.Paths
```

Vehicles ([out, retval] IVehicles **ppVehicles)

Creates an instance of a Vehicles object (see page 84), that gives individual access to the vehicles on the network, including the parked ones.

Parameters

```
[out, retval] IVehicles **ppVehicles :   returned Vehicles object
```

Example

```
DIM vehicles AS Vehicles
SET vehicles = net.Vehicles
```

VehicleInputs ([out, retval] IVehicleInputs **ppVehicleInputs)

Creates an instance of a VehicleInputs object (see page 84), that gives individual access to the vehicle inputs of the network.

Parameters

```
[out, retval] IVehicleInputs **ppVehicleInputs :  returned VehicleInputs object
```

Example

```
DIM inps AS VehicleInputs
SET inps = net.VehicleInputs
```

RoutingDecisions ([out, retval] IRoutingDecisions **ppRoutingDecisions)

Creates an instance of a RoutingDecisions object (see page 99), that gives individual access to the routing decisions of the network.

Parameters

```
[out, retval] IRoutingDecisions **ppRDs : returned RoutingDecisions object
```

Example

```
DIM rds AS RoutingDecisions
SET rds = net.RoutingDecisions
```

SignalControllers ([out, retval] ISignalControllers **ppSignalControllers)

Creates an instance of a SignalControllers object (see page 112), that gives individual access to the signal controller elements of the network.

Parameters

[out, retval] ISignalControllers **ppSignalControllers : returned SignalControllers object

Example

```
DIM scs AS SignalControllers
SET scs = net.SignalControllers
```

DataCollections ([out, retval] IDataCollections **ppDataCollections)

Creates an instance of a DataCollections object (see page 151), that gives individual access to the data collections defined in the network.

Parameters

```
[out, retval] IDataCollections **ppDataCollections : returned DataCollections object
```

Example

```
DIM colls AS DataCollections
SET colls = net.DataCollections
```

QueueCounters ([out, retval] IQueueCounters **ppQueueCounters)

Creates an instance of a QueueCounters object (see page 169), that gives individual access to the queue counters defined in the network.

Parameters

```
[out, retval] IQueueCounters **ppQueueCounters : returned QueueCounters object
```

Example

```
DIM qcs AS QueueCounters
SET qcs = net.QueueCounters
```

TravelTimes ([out, retval] ITravelTimes **ppTravelTimes)

Creates an instance of a TravelTimes object (see page 174), that gives individual access to the trave times defined in the network.

Parameters

```
[out, retval] ITravelTimes **ppTravelTimes : returned TravelTimes object
```

Example

```
DIM tts AS TravelTimes
SET tts = net.TravelTimes
```

Delays ([out, retval] IDelays **ppDelays)

Creates an instance of a Delays object (see page 179), that gives individual access to the delays defined in the network.

Parameters

```
[out, retval] IDelays **ppDelays : returned Delays object
```

Example

```
DIM dels AS Delays
SET dels = net.Delays
```

AttValue ([in] BSTR Attribute, [out, retval] VARIANT *pValue)

Returns a network property. Please get the language independent attribute tags from the table at the end of this section.

Parameters

```
[in] BSTR Attribute :            attribute name (see below)
[out, retval] VARIANT *pValue :  returned value of the attribute
```

Example

```
width = net.AttValue("WIDTH")
```

AttValue ([in] BSTR Attribute, [in] VARIANT Value)

Sets a network property. Please get the language independent attribute tags from the table at the end of this section.

Parameters

```
[in] BSTR Attribute :            attribute name (see below)
[in] VARIANT Value :             new attribute value. (type according to the attribute)
```

Example

```
net.AttValue("NAME") = "my network"
```

Attribute outline :

R	W	Attribute	Description
✓		ID	Not used
✓	✓	NAME	Name (currently equivalent to the simulation comment)
✓		HEIGHT	Network vertical dimension ([m], [ft])
✓		WIDTH	Network horizontal dimension ([m], [ft])

2.3.2 Methods of the INet Interface

Rotate ([in] double Angle)

Rotates the network counterclockwise by the specified angle in degrees.

Parameters

```
[in] double Angle :              angle in degrees
```

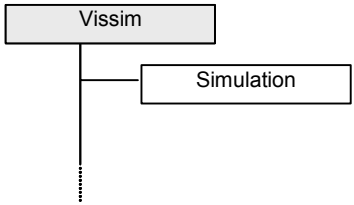
Translate ([in] double X, [in] double X, [in] double Z)

Translates the network by the specified X, Y and Z distance using the current units setting.

Parameters

```
[in] double X :          X coordinate in current units
[in] double Y :          Y coordinate in current units
[in] double Z :          Z coordinate in current units
```

2.4 Simulation



The Simulation object belongs to Vissim and can be accessed through the property Simulation of the IVissim interface. It enables configuration and control of simulation runs. VISSIM is a single project program, i.e. it can simulate only one network simultaneously. Therefore, a Simulation instance always references the current simulation and the parameters set for the currently opened network.

Example

```
DIM vissim AS Vissim
DIM simulation AS Simulation
SET vissim = NEW Vissim
SET simulation = vissim.Simulation
vissim.LoadNet "example.inp"
simulation.RunContinuous  ' it runs an entire simulation synchronously
```

2.4.1 Properties of the ISimulation Interface

Comment ([out, retval] BSTR *pComment)

Returns the comment in the simulation parameters.

Parameters

```
[out, retval] BSTR *pComment :   returned comment
```

Example

```
comment = simulation.Comment
```

Comment ([in] BSTR Comment)

Sets the comment in the simulation parameters. Maximal 199 charatcters.

Parameters

```
[in] BSTR Comment :      new comment
```

Example

```
simulation.Comment = „Karlsruhe-Durlach Test11"
```

Period ([out, retval] double *pPeriod)

Returns the simulation period.

Parameters

```
[out, retval] double *pPeriod :  returned period in simulation seconds
```

Example

```
period = simulation.Period
```

Period ([in] double Period)

Sets the simulation period.

Parameters

```
[in] double Period :     new period in simulation seconds
```

Example

```
simulation.Period = 3600
```

StartTime ([out, retval] BSTR *pStartTime)

Returns the start time of the simulation.

Parameters

```
[out, retval] BSTR *pStartTime : returned start time in hh:mm:ss
```

Example

```
starttime = simulation.StartTime
```

StartTime ([in] BSTR StartTime)

Sets the start time of the simulation.

Parameters

```
[in] BSTR StartTime :    new start time in hh:mm:ss
```

Example

```
simulation.StartTime = „10:30:00“
```

Speed ([out, retval] double *pSpeed)

Returns the speed of the simulation runs (simulated seconds per real time second rate). It returns a negative value if maximum is selected.

Parameters

```
[out, retval] double *pSpeed :   returned speed in simulation seconds per real second (≤
0 if maximum speed is set)
```

Example

```
speed = simulation.Speed
```

Speed ([in] double Speed)

Sets the speed of the simulation runs (simulated seconds per real time second rate). A negative value or 0 switches to maximum speed but stores, if the value is lesser than 0, the unsigned value as the new simulation speed rate.

Parameters

```
[in] double Speed :      new speed in simulation seconds per real second (≤ 0 for maximal
speed)
```

Example

```
simulation.Speed = 10.0
simulation.Speed = 0     'maximum speed, without setting the rate
simulation.Speed = -10.0 'sets the rate to 10 Sim.sec/sec but uses maximum speed
```

Resolution ([out, retval] int *pResolution)

Returns the simulation resolution.

Parameters

```
[out, retval] int *pResolution :          returned resolution in time steps per simulation
second
```

Example

```
resolution = simulation.Resolution
```

Resolution ([in] int Resolution)

Sets the simulation resolution .

Parameters

```
[in] int Resolution :             new resolution in time steps per simulation second
```

Example

```
simulation.Resolution = 5   ' Time step(s) / Sim.sec.
```

ControllerFrequency ([out, retval] int *pControllerFrequency)

Returns the controller frequency of the simulation.

Parameters

```
[out, retval] int *pControllerFrequency : returned controller frequency (number of passes
per simulation second)
```

Example

```
controllerfreq = simulation.ControllerFrequency  ' Number of passes / Sim.sec.
```

ControllerFrequency ([in] int ControllerFrequency)

Sets the controller frequency of the simulation.

Parameters

```
[in] int ControllerFrequency :   new controller frequency (number of passes per
simulation second)
```

Example

```
simulation.ControllerFrequency = 5  ' Number of passes / Sim.sec.
```

RandomSeed ([out, retval] long *pRandomSeed)

Returns the random seed of the simulation.

Parameters

```
[out, retval] long *pRandomSeed : returned random seed.
```

Example

```
randomseed = simulation.RandomSeed
```

RandomSeed ([in] long RandomSeed)

Sets the random seed of the simulation.

Parameters

```
[in] long RandomSeed :   new random seed.
```

Example

```
simulation.RandomSeed = 13
```

BreakAt ([out, retval] double *pBreakAt)

Returns the second at which the simulation shall halt and change to single step mode.

Parameters

```
[out, retval] double *pBreakAt : returned break-at-simulation-second value.
```

Example

```
breakat = simulation.BreakAt
```

BreakAt ([in] double BreakAt)

Sets the second at which the simulation shall halt and change to single step mode.

Parameters

```
[in] double BreakAt :    new break-at-simulation-second value.
```

Example

```
simulation.BreakAt = 1000 'stop simulation when 1000 seconds have been simulated
```


After a continuous simulation has been halted because of the property BreakAt, it is possible to continue the simulation process in single step or continuous mode with the method RunSingleStep or RunContinuously respectively. If a whole new simulation run is to be started, the method Stop must be called before calling a Run… function again.

LeftSideTraffic ([out, retval] unsigned char *pLeftSideTraffic)

Returns true if the side of traffic flow is set to “left”, otherwise false.

Parameters

```
[out, retval] unsigned char *pLeftSideTraffic :   returned 1 for left side traffic,
otherwise 0.
```

Example

```
is_leftsidetraffic = simulation.LeftSideTraffic
```

LeftSideTraffic ([in] unsigned char LeftSideTraffic)

Turns the left side traffic switch on/off.

Parameters

```
[in] unsigned char LeftSideTraffic :      0 for left side traffic off, otherwise on
```

Example

```
simulation.LeftSideTraffic= 1  'left side traffic on
```

RunIndex ([out, retval] long *pIndex)

Returns the internal simulation run index, that is used when several simulations are run consecutively. The index can only be changed with the put version of this property (see below).

Parameters

```
[out, retval] long *pIndex :      returned index
```

Example

```
index = simulation.RunIndex
```

RunIndex ([in] long Index)

Sets the simulation run index. It affects some evaluation files (like link and path evaluation) and is useful when running several consecutive simulations. The index must be initialized to 0 at incremented by 1 on each iteration when the Microsoft Access database is used to collect the evaluation results.

Parameters

```
[in] long Index :        new index
```

Example

```
simulation.RunIndex = 2
```

AttValue ([in] BSTR Attribute, [out, retval] VARIANT *pValue)

Returns a simulation attribute. Please get the language independent attribute tag from the table at the end of this section.

Parameters

```
[in] BSTR Attribute :            attribute name (see below)
[out, retval] VARIANT*pValue :   return of the attribute value
```

Example

```
period = simulation.AttValue(„PERIOD")
```

AttValue ([in] BSTR Attribute, [in] VARIANT Value)

Sets a simulation attribute. Please get the language independent attribute tag from the table at the end of this section.

Parameters

```
[in] BSTR Attribute :            Attribute name (see below)
[in] VARIANT Value :             Attribute value. (type according to attribute)
```

Example

```
simulation.AttValue(„PERIOD") = 3600
```

Attribute outline :

R	W	Attribute	Description
✓	✓	BREAKAT	Break at a given simulation second
✓	✓	COMMENT	Comment as a string
✓	✓	CONTROLLERFREQUENCY	Controller frequency: Number of passes per simulation second of signal controllers
✓		ELAPSEDTIME	Simulated time seconds since start of simulation [s]
✓	✓	LEFTSIDETRAFFIC	0 for right-side traffic , ≠ 0 for left-side traffic

R	W	Attribute	Description
✓	✓	PERIOD	Period in simulation seconds
✓	✓	RANDOMSEED	Random seed
✓	✓	RESOLUTION	Resolution in time steps per simulation second
✓	✓	RUNINDEX	Run index for evaluation with consecutive simulation runs
✓	✓	SEARCHPATHS	Flag to search new dynamic assignment paths
✓	✓	SPEED	Speed in simulation seconds per second. ≤ 0 maximal speed
✓	✓	STARTTIME	Start time as a string with the format “hh:mm:ss”
✓	✓	STATUSLINE	Enables/disables the status line when simulating.
✓	✓	STORECOSTS	Flag to store dynamic assignment costs to the BEW file
✓	✓	STOREPATHS	Flag to store dynamic assignment paths to the WEG file
✓	✓	THREADS	Number of threads for the simulation kernel. To be set previous to the simulation start.
✓		TIME	Simulation time [hh:mm:ss]

2.4.2 Methods of the ISimulation Interface

RunContinuous

Starts or continues (if coming changing from single step mode) a simulation run in continuous mode (see the BreakAt property for switching into single step mode at a predetermined simulation time). If a presentation is currently running it will be previously stopped.

Example

```
simulation.RunContinuous
```

RunSingleStep

Executes a single simulation time step (It starts a simulation run if no simulation is running). If a presentation is currently running it will be previously stopped.

Example

```
simulation.RunSingleStep
```

Stop

Stops a simulation run. If no simulation is running this command is ignored.

Example

```
simulation.Stop
```

LoadSnapshot ([in, defaultvalue("")] BSTR SnapshotPath)

Loads the simulation state saved of the given and starts a simulation in single step mode.

Parameters

```
[in] BSTR SnapshotPath : path + filename (*.snp) of the snapshot file to be read
```

Example

```
SET vissim = NEW Vissim
vissim. Simulation.LoadSnapshot "c:\vissim\data\example.snp"
```

SaveSnapshot ([in, defaultvalue("")] BSTR SnapshotPath)

Save the current simulation state into a snapshot file (*.snp).

Parameters

```
[in] BSTR SnapshotPath : path + filename (*.snp) of the snapshot file to be read
```

Example

```
SET vissim = NEW Vissim
vissim.Simulation.SaveSnapshot "c:\vissim\data\example.snp"
```


All simulation methods (RunContinuous included) are executed synchronously. No other command can be processed until the current one is done. In terms of a Visual Basic program it means that the client will wait until the VISSIM COM server has executed the command. See the property BreakAt to interrupt a simulation in continuous mode at a predetermined simulation time.

Examples

- Running the first 1000 simulation seconds in continuous mode and then one additional single time step:

```
simulation.BreakAt = 1000
simulation.RunContinuous
simulation.RunSingleStep
```

- Running a simulation with possible manual stop:

```
FOR i = 0 TO simulation.Period   'if resolution is 1 step per second
simulation.RunSingleStep

DOEVENTS   'allow Visual Basic to detect new events
IF stop_pushed THEN GOTO the_end
NEXT I

the_end:
simulation.Stop
```

- Running 10 simulations continuously with different run indices and random seeds:

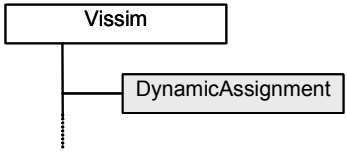
```
simulation.RunIndex = 0
FOR i = 0 to 10
simulation.RunContinuous

simulation.RunIndex = simulation.RunIndex + 1
simulation. RandomSeed = simulation.RandomSeed + 1
NEXT i
```


If you are using database evaluation start from index 0. Otherwise the database won't be created and an error will occur.

2.5 DynamicAssignment

The DynamicAssignment object belongs to Vissim and can be accessed through the property DynamicAssignment of the IVissim interface. It enables configuration and control of dynamic assignment options. A DynamicAssignment instance references always the current dynamic assignment options for the currently opened network.



Example

```
DIM vissim AS Vissim
DIM dynassig AS DynamicAssignment
SET vissim = NEW Vissim
SET dynassig = vissim.DynamicAssignment
vissim.LoadNet "example.inp"
dynassig.AttValue("STOREPATHS")
```

2.5.1 Properties of the IDynamicAssignment Interface

AttValue ([in] BSTR Attribute, [out, retval] VARIANT *pValue)

Returns a dynamic assignment attribute. Please get the language independent attribute tag from the table at the end of this section.

Parameters

```
[in] BSTR Attribute :            attribute name (see below)
[out, retval] VARIANT*pValue :   return of the attribute value
```

Example

```
period = dynassig.AttValue(„STOREPATHS ")
```

AttValue ([in] BSTR Attribute, [in] VARIANT Value)

Sets a dynamic assignment attribute. Please get the language independent attribute tag from the table at the end of this section.

Parameters

```
[in] BSTR Attribute :            Attribute name (see below)
[in] VARIANT Value :             Attribute value. (type according to attribute)
```

Example

```
dynassig.AttValue(„STOREPATHS") = TRUE
```

Attribute outline

R	W	Attribute	Description
✓		CONVERGEALL	True if all active convergence

R	W	Attribute	Description
			checks assert true
✓	✓	CONVERGENCETTEDGESCHECK	Flag to check edge travel times convergence
✓	✓	CONVERGENCETTEDGESFACTOR	Factor edge travel times convergence [%]
✓	✓	CONVERGENCETTEDGESMAX	Current maximal percentage difference between old and new edge travel times [%]
✓	✓	CONVERGENCETTPATHSCHECK	Flag to check path travel times convergence
✓	✓	CONVERGENCETTPATHSFACTOR	Factor path travel times convergence [%]
✓	✓	CONVERGENCETTPATHSMAX	Current maximal percentage difference between old and new path travel times [%]
✓	✓	CONVERGENCEVOLEDGESCHECK	Flag to check edge volumes convergence
✓	✓	CONVERGENCEVOLEDGESFACTOR	Factor edge volumes convergence [Veh]
✓	✓	CONVERGENCEVOLEDGESMAX	Current maximal difference between old and new edge volumes [%]
✓	✓	EVENTSMOOTHING	Enable/Disable the event EdgeSmoothing
✓	✓	EVENTROUTECHOICEDIST	Enable/Disable the event RouteChoiceDistribution
✓	✓	SEARCHPATHS	Flag to search new dynamic assignment paths
✓	✓	STORECOSTS	Flag to store costs to the BEW file
✓	✓	STOREPATHS	Flag to store paths to the WEG file
✓	✓	VOLUME	Scales the total volume [%]

As a complementary information to the convergence options of the dynamic assignment, the convergence of path travel times, edge travel times and edge volumes can be checked individually using the named attributes CONVERGETTEDGESMAX, CONVERGETTPATHSMAX and CONVERGEVOLEDGESMAX. These results are available during and after the simulation. But a result will be useful for comparision only at the end of every simulation starting from the second iteration.

2.5.2 Methods of the IDynamicAssignment Interface

2.5.3 Events of the DynamicAssignment Object

EdgeSmoothing ([in] double oldVal, [in] doule newVal, [out, retval] doule retVal)

Created event (if attribute EVENTSMOOTHING enabled) for every calculation of the smoothed edge travel times during the simulation. VISSIM will use the returned value as the new smoothed edge value.

Parameters

```
[in] double oldVal :            old smoothed value
[in] double newVal :            newly measured value
[out, retval] double retVal :   new computated smoothed value
```

Example

```
DIM vis AS Vissim
DIM sim AS Simulation
Dim WithEvents dyn as DynamicAssignment
Dim nIteration AS Long

' implementation of the DynamicAssignment event EdgeSmoothing
Private Function dyn_EdgeSmoothing(ByVal oldVal As Long, ByVal newVal As Long) As Double
' Method of successive Averages (MSA)
dyn_EdgeSmoothing = oldVal * (1# - 1# / nIteration)+ newVal  * 1# / nIteration
End Function

SET vis = NEW Vissim
SET sim = vis.Simulation
SET dyn = vis.DynamicAssignment

vis.LoadNet "example.inp")
dyn.AttValue("EVENTSMOOTHING") = TRUE
FOR nIteration = 1 TO 10
sim.RunContinuous
NEXT nIteration
```

RouteChoiceDistribution ([in] VARIANT Costs, [out] VARIANT *Distribution)

Created event (if the attribute EVENTROUTECHOICEDIST is enabled) for the calculation of the route choice distribution of a parking lot relation. VISSIM will use the returned relative percentages in Distribution to distribute the vehicles on the different routes.

Parameters

```
[in] VARIANT Costs :            current paths costs of the parking lot relation
[out] VARIANT *Distribution :   returned relative percentages
```

Example

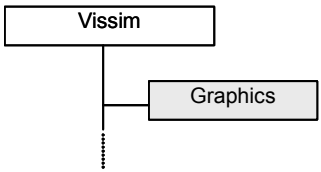
```
DIM vis AS Vissim
DIM sim AS Simulation
Dim WithEvents dyn as DynamicAssignment

' implementation of the DynamicAssignment event RouteChoiceDistribution
Private Sub dyn_RouteChoiceDistribution (ByVal Costs As VARIANT,
ByRef Distribution As VARIANT)
' Logit function:
DIM dist() AS Double
s = 0#
FOR i = LBOUND(Costs) To UBOUND(Costs)
s = s + EXP(0.2 / Costs(i))
NEXT i
FOR i = LBOUND(Costs) To UBOUND(Costs)
dist(i) = EXP(0.2 / Costs(i)) / s
NEXT i
Distribution = dist
END SUB

SET vis = NEW Vissim
SET sim = vis.Simulation
SET dyn = vis.DynamicAssignment

vis.LoadNet "example.inp")
dyn.AttValue("EVENTROUTECHOICEDIST") = TRUE
sim.RunContinuous
```

2.6 Graphics



The Graphics object belongs to Vissim and can be accessed through the property Graphics of the IVissim interface. It gives access to the network graphics options. These options are globally used for the Vissim instance and affect all networks and simulations during its existence. The VISSIM internal set of default graphic options is used for the creation of a new instance. Use the IVissim method LoadLayout to load a stored set of graphics.

Example

```
DIM vissim AS Vissim
DIM graphics AS Graphics
SET vissim = NEW Vissim
SET graphics = vissim.Graphics
```

2.6.1 Properties of the IGraphics Interface

AttValue ([in] BSTR Attribute, [out, retval] VARIANT *pValue)

Returns a graphic attribute. Please get the language independent attribute tag from the table at the end of this section.

Parameters

```
[in] BSTR Attribute :            attribute name (see table below)
[out, retval] VARIANT *pValue :  return of the attribute value
```

Example

```
type = graphics.AttValue(„VISUALIZATION")
```

AttValue ([in] BSTR Attribute, [in] VARIANT Value)

Sets a graphic attribute. Please get the language independent attribute tag from the table at the end of this section.

Parameters

```
[in] BSTR Attribute :    Attribute name (see table below)
[in] VARIANT Value :     Attribute value. (type according to attribute)
```

Example

```
graphics.AttValue(„VISUALIZATION") = 0 'disabled
```

Attribute outline

R	W	Attribute	Description
✓	✓	3D	0 = 2D mode, 1 = 3d mode

R	W	Attribute	Description
✓	✓	DISPLAY	0 = normal, 1 = center line, 2 = invisible, 3 = alternative
✓	✓	LAND	Land color as a long in RGB format
✓	✓	LINKS	Links color as a long in RGB format
✓	✓	SKY	Sky color as a long in RGB format
✓	✓	VISUALIZATION	Vehicles/aggregated values visualization on/off (true/false)

Note: The color values passed to or returned from VISSIM must be/are in RGB format. The valid range for a normal RGB color is 0 to 16,777,215 (&HFFFFFF). Each color setting (property or argument) is a 4-byte integer. The high byte of a number in this range equals 0. The lower 3 bytes, from least to most significant byte, determine the amount of red, green, and blue, respectively. The red, green, and blue components are each represented by a number between 0 and 255 (&HFF). In Visual Basic you can use the RGB(red, green, blue) function to obtain the corresponding integer.

If you are working with applications that require the byte order to be reversed you can use one of the following methods in VB:

```
color = CLng(blue + (green*&H100) + (red*&H10000))
rgb = RGB(&HFF And (color\&H10000), &HFF And (color\&H100), &HFF And color)
```

2.6.2 Methods of the IGraphics Interface

Redraw ()

Redraw the network repainting all elements with the current graphic options.

Example

```
Dim vissim As Vissim
SET vissim = NEW Vissim
vissim.LoadNet "c:\vissim\daten\example.inp"
vissim.Graphics.Redraw
```

GetWindow ([in] BSTR WinName, [out] VARIANT *Top, [out] VARIANT *Left, [out] VARIANT *Bottom, [out] VARIANT *Right, [in, defaultvalue(0)] long ID)

Retrieve the position of the window specified with the name attribute WinName and the optional ID number. The dimensions are given in screen coordinates that are relative to the upper-left corner of the screen.

Parameters

```
[in] BSTR Attribute :    Window name (see table below)
```

```
[out] VARIANT *Top :     the screen coordinate of the top edge
[out] VARIANT *Left :    the screen coordinate of the left edge
[out] VARIANT *Bottom :  the screen coordinate of the bottom edge
[out] VARIANT *Right :   the screen coordinate of the right edge
[in] long ID :           Optional window ID number (see table below)
```

Example

```
SET gr = vissim.Graphics
gr.GetWindow "SIGNALTT", top, left, bottom, right, 1
```

SetWindow ([in] BSTR WinName, [in] VARIANT Top, [in] VARIANT Left, [in] VARIANT Bottom, [in] VARIANT Right, ,m[in, defaultvalue(0)] long ID)

Set the position of the window specified with the name attribute WinName and the optional ID number. The dimensions are given in screen coordinates that are relative to the upper-left corner of the screen.

Parameters

```
[in] BSTR Attribute :    Window name (see table below)
[in] VARIANT Top :       the screen coordinate of the top edge
[in] VARIANT Left :      the screen coordinate of the left edge
[in] VARIANT Bottom :    the screen coordinate of the bottom edge
[in] VARIANT Right :     the screen coordinate of the right edge
[in] long ID :           Optional window ID number (see table below)
```

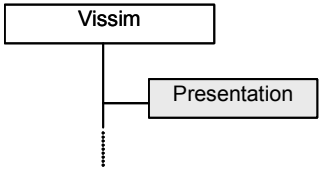
Example

```
SET gr = vissim.Graphics
gr.SetWindow "SIGNALTT", 800, 1000, 900, 1100, 1
```

WinName	ID	Description
NETWORK	-	Currently the main window
SCDETRECORD	SC Identifier number	Signal Control Detector Record window of the ID signal controller.
SIGNALTT	SC Identifier number	Signal Times Table window of the ID signal controller.

2.7 Presentation

The Presentation object belongs to Vissim and can be accessed through the property Presentation of the IVissim interface. It allows to record and control presentations.



Example

```
DIM vissim AS Vissim
DIM present AS Presentation
SET vissim = NEW Vissim
SET present = vissim.Presentation
```

2.7.1 Properties of the IPresentation Interface

AttValue ([in] BSTR Attribute, [out, retval] VARIANT *pValue)

Returns a presentation attribute. Please get the language independent attribute tag from the table at the end of this section.

Parameters

```
[in] BSTR Attribute :            attribute name (see below)
[out, retval] VARIANT *pValue :  return of the attribute value
```

Example

```
backwards = present.AttValue(„BACKWARDS“)
```

AttValue ([in] BSTR Attribute, [in] VARIANT Value)

Sets a presentation attribute. Please get the language independent attribute tag from the table at the end of this section.

Parameters

```
[in] BSTR Attribute :    Attribute name (see below)
[in] VARIANT Value :     Attribute value. (type according to attribute)
```

Example

```
present.AttValue(„RECORDING“) = TRUE
```

Attribute outline :

R	W	Attribute	Description
✓	✓	BACKWARDS	Backwards animation.
✓	✓	FORWARDS	Forwards animation.
✓		NINTERVALS	Return the number of recorded time intervals
✓	✓	RECORDING	Recording the animation during a simulation.

R	W	Attribute	Description
✓		TIMEFROM	Time intervals start [s]. Parameter: interval time order (1..n)
✓		TIMEUNTIL	Time intervals end [s]. Parameter: interval time order (1..n)

Currently the name for the animation file (*.ani) will be the same as the name of the*.inp file. When recording, if this exists it will be overwritten. Also, it is required to set the recording attribute previous to the simulation start. If enabled the animation of the whole network will be recorded until simulation stop.

2.7.2 Methods of the IPresentation Interface

RunContinuous

Start or continues (if coming changing from single step mode) a presentation in continuous mode using the *.ani file named as the *.inp file. If a simulation is currently running it will be previously stopped.

Example

```
present.RunContinuous
```

RunSingleStep

Execute a single presentation time step. (Start a presentation if no presentation is yet running). Uses the *.ani file named as the *.inp file. If a simulation is currently running it will be previously stopped.

Example

```
present.RunSingleStep
```

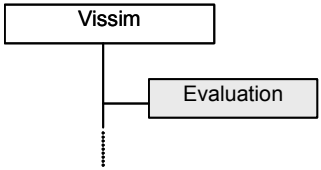
Stop

Stop a presentation. If no presentation is running this command is ignored.

Example

```
present.Stop
```

2.8 Evaluation



The Evaluation object belongs to Vissim and can be accessed through the property Evaluation of the IVissim interface. It gives access to the simulation evaluation options. These options are globally used for the Vissim instance and affect all networks and simulations during its existence. No set of default evaluation options is used for the creation of a new instance. Use the IVissim method LoadLayout to load a stored set of evaluations.

Example

```
DIM vissim AS Vissim
DIM eval AS Evaluation
SET vissim = NEW Vissim
SET eval = vissim.Evaluation
```

2.8.1 Properties of the IEvaluation Interface

LinkEvaluation ([out, retval] ILinkEvaluation **ppLinkEvaluation)

Creates an instance of a LinkEvaluation object (see page 174), that gives access to the individual configuration of the evaluation.

Parameters

```
[out, retval] ILinkEvaluation **ppLinkEvaluation : returned object
```

Example

```
DIM linkeval AS LinkEvaluation
SET linkeval = eval.LinkEvaluation
```

DataCollectionEvaluation ([out, retval] IDataCollectionEvaluation **ppEval)

Creates an instance of a DataCollectionEvaluation object (see page 187), that gives access to the individual configuration of the evaluation.

Parameters

```
[out, retval] IDataCollectionEvaluation **ppEval : returned object
```

Example

```
DIM dceval AS DataCollectionEvaluation
SET dceval = eval.DataCollectionEvaluation
```

QueueCounterEvaluation ([out, retval] IQueueCounterEvaluation **ppEval)

Creates an instance of a QueueCounterEvaluation object (see page 189), that gives access to the individual configuration of the evaluation.

Parameters

```
[out, retval] IQueueCounterEvaluation **ppEval : returned object
```

Example

```
DIM qceval AS QueueCounterEvaluation
SET qceval = eval.QueueCounterEvaluation
```

TravelTimeEvaluation ([out, retval] ITravelTimeEvaluation **ppEval)

Creates an instance of a TravelTimeEvaluation object (see page 191), that gives access to the individual configuration of the evaluation.

Parameters

```
[out, retval] ITravelTimeEvaluation **ppEval : returned object
```

Example

```
DIM tteval AS TravelTimeEvaluation
SET tteval = eval.TravelTimeEvaluation
```

DelayEvaluation ([out, retval] IDelayEvaluation **ppEval)

Creates an instance of a DelayEvaluation object (see page 193), that gives access to the individual configuration of the evaluation.

Parameters

```
[out, retval] IDelayEvaluation **ppEval : returned object
```

Example

```
DIM deval AS DelayEvaluation
SET deval = eval.DelayEvaluation
```

NodeEvaluation ([out, retval] INodeEvaluation **ppEval)

Creates an instance of a NodeEvaluation object (see page 195), that gives access to the individual configuration of the evaluation.

Parameters

```
[out, retval] INodeEvaluation **ppEval : returned object
```

Example

```
DIM nodeeval AS NodeEvaluation
SET nodeeval = eval.NodeEvaluation
```

AttValue ([in] BSTR Attribute, [out, retval] VARIANT *pValue)

Returns an evaluation attribute. Please get the language independent attribute tag from the table at the end of this section.

Parameters

```
[in] BSTR Attribute :           attribute name (see below)
[out, retval] VARIANT *pValue :  return of the attribute value
```

Example

```
enabled = eval.AttValue(„VEHICLERECORD")
```

AttValue ([in] BSTR Attribute, [in] VARIANT Value)

Sets an evaluation attribute. Please get the language independent attribute tag from the table at the end of this section.

Parameters

```
[in] BSTR Attribute :    Attribute name (see below)
[in] VARIANT Value :     Attribute value. (type according to attribute)
```

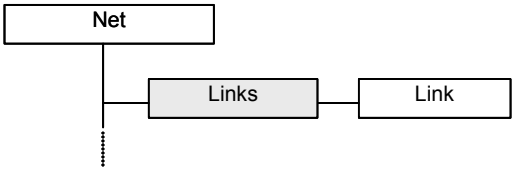
Example

```
eval.AttValue(„VEHICLERECORD") = True
```

Attribute outline

R	W	Attribute	Description
✓	✓	CONVERGENCE	Convergence evaluations (True/False)
✓	✓	DATACOLLECTION	Data collections (True/False)
✓	✓	DELAY	Delays (True/False)
✓	✓	EXPORT	Export evaluations (True/False)
✓	✓	GREENTIMES	Distance of Green times (True/False)
✓	✓	LANECHANGE	Lane change evaluations (True/False)
✓	✓	LINK	Link segment evaluations (True/False)
✓	✓	NETPERFORMANCE	Net performance evaluations (True/False)
✓	✓	NODE	Node evaluations (True/False)
✓	✓	PATHS	Dynamic assignment path evaluations (True/False)
✓	✓	PUBLICWAITTIME	Bus/Tram waiting time (True/False)
✓	✓	OBSERVER	Observer evaluations (True/False)
✓	✓	QUEUECOUNTER	Queue counter (True/False)
✓	✓	SCDETRECORD	Signal changes/Detector record (True/False)
✓	✓	SIGNALCHANGES	Signal changes protocol (True/False)
✓	✓	SPECIAL	Special evaluations (True/False)
✓	✓	TRAVELTIME	Travel times (True/False)
✓	✓	VEHICLEINPUT	Vehicle inputs (True/False)
✓	✓	VEHICLERECORD	Vehicle record (True/False)

2.9 Links



The Links object is a collection of Link objects (see page 53). It belongs to the Net object and can be accessed through the Links property of the INet interface.

It contains all links and connectors of the loaded network and allows the iteration through all of them as well as individual access (see also Link object).

Example

Instantiation of Links object and access to all its Link objects:

```
Dim vissim As Vissim
Dim links As Links
SET vissim = NEW Vissim
vissim.LoadNet "c:\vissim\daten\example.inp"
Set links = Vissim.Net.Links

FOR EACH link IN links  'access to the method _NewEnum to create an enumeration
  …
NEXT link

'or also:

FOR i = 1 TO links.Count
    SET link = links(i)  'or links.Item(i)
    …
NEXT i
```

2.9.1 Properties of the ILinks Interface

_NewEnum ([out, retval] LPUNKNOWN *ppEnum)

This property creates a collection (or enumeration) of all Link objects in the current network. Once a collection is created, individual members can be returned using the GetLinkByNumber method, while the entire collection can be iterated through using a FOR ...TO ... NEXT statement. In Visual Basic you can use the statement FOR EACH…NEXT without using _NewEnum, which VB calls internally (see Links example above).

Parameters

`[out, retval] LPUNKNOWN *ppEnum : returned Enumeration`

Item ([in] VARIANT Index, [out, retval] ILink **ppLink)

Returns a single Link of the collection selected by position. This is useful only when all links shall be accessed (see the Links example above). To select a link by its identifier number use GetLinkByNumber (see below). Since Item is the default property for a collection the following commands are equivalent:

```
SET link = links(1)
SET link = links.Item(1)
```

Parameters

```
[in] long Index:                index between 1 and Links.Count
[out, retval] ILink **ppLink:   returned Link object
```

Count ([out, retval] long *pCount)

Returns the number of Link objects in the collection. See the Links example above.

Parameters

```
[out, retval] long *pCount :    returned number of objects.
```

2.9.2 Methods of the ILinks Interface

GetLinkByNumber ([in] long Number, [out, retval] ILink **ppLink)

Returns the Link object with the identifier number Number.

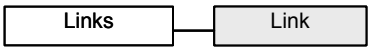
Parameters

```
[in] long Number:               link identifier
[out, retval] ILink **ppLink:   returned Link object
```

Example

```
SET link = links.GetLinkByNumber(1001)
```

2.10 Link



This object represents a link or connector element and belongs to the Links object. A Link object can be accessed through the Links object in two ways:

- access via iteration through the collection

```
DIM links As Links
DIM link As Link
SET links = Vissim.Net.Links
FOR EACH link IN links
List.AddItem link.ID
NEXT link
```

- individual access via identifier

```
DIM links As Links
DIM link As Link
SET links = Vissim.Net.Links
SET link = links. GetLinkByNumber(1000)
```

The Link object enables access to the properties of the link through the ILink interface.

2.10.1 Properties of the ILink Interface

ID ([out, retval] long *pID)

Returns the link identifier number. If the Link object doesn't refer to a valid VISSIM link element anymore the returned value is 0.

Parameters

```
[out, retval] long *pID : returned identifier (0 if reference no longer valid)
```

Example

```
DIM link AS Link
id = link.ID
```

Name ([out, retval] BSTR *pName)

Returns the link name.

Parameters

```
[out, retval] BSTR *pName :      returned name.
```

Example

```
name = link.Name
```

Name ([in] BSTR Name)

Sets the link name. Maximal 255 charatcters.

Parameters

```
[in] BSTR Name : new name.
```

Example

```
link.Name = „Banbury Road up"
```

AttValue ([in] BSTR Attribute, [out, retval] VARIANT *pValue)

Returns a link attribute. Please get the language independent attribute tag from the table below.

Parameters

```
[in] BSTR Attribute :            attribute name (see below)
[out, retval] VARIANT *pValue :  returned value of the attribute
```

Example

```
length = link.AttValue(„LENGTH")
polyline = link.AttValue(„POINTS")
FOR i = LBOUND(polyline) TO UBOUND(polyline)
        x = polyline (i).X
        y = polyline (i).Y
NEXT i
```

AttValue ([in] BSTR Attribute, [in] VARIANT Value)

Sets a link attribute. Please get the language independent attribute tag from the table below.

Parameters

```
[in] BSTR Attribute :    Attribute name (see below)
[in] VARIANT Value :     Attribute value. (type according to attribute)
```

Example

```
link.AttValue(„NAME") = "stumpfstr."
```

Attribute outline

R	W	Attribute	Description
✓		ID	Identifier number
✓	✓	NAME	Name
✓		CONNECTOR	True if the link is a connector
✓	✓	COST	Link cost per km
✓		FROMLANE	If it is a connector: the number of the right most lane of the origin link. Otherwise 0.
✓		FROMLINK	If it is a connector: the number of the origin link. Otherwise 0.
✓		GRADIENT	Gradient in %
✓		LANEWIDTH	Lane width in
✓		LENGTH	Length in the units of the current options
✓		NUMLANES	Lanes number
✓		POINTS	Reference line through the center of the link

R	W	Attribute	Description
			(array of VARIANTs).
✓	✓	SURCHARGE1	Weight sensitive surcharge
✓	✓	SURCHARGE2	Weight insensitive surcharge
✓		TOLANE	If it is a connector: the number of the right most lane of the destination link. Otherwise 0.
✓		TOLINK	If it is a connector: the number of the destination link. Otherwise 0.
✓	✓	TYPE	Link type

AttValue1 ([in] BSTR Attribute, [in] VARIANT Parameter, [out, retval] VARIANT *pValue)

Returns a link attribute with one parameter. Please get the language independent attribute tag from the table below.

Parameters

```
[in] BSTR Attribute :            attribute name (see below)
[in] VARIANT Parameter :         attribute dependent parameter (see below)
[out, retval] VARIANT *pValue :  returned value of the attribute
```

AttValue1 ([in] BSTR Attribute, [in] VARIANT Parameter, [in] VARIANT Value)

Sets a link attribute with one parameter. Please get the language independent attribute tag from the table below.

Parameters

```
[in] BSTR Attribute :    Attribute name (see below)
[in] VARIANT Parameter : attribute dependent parameter (see below)
[in] VARIANT Value :     Attribute value. (type according to attribute)
```

Attribute outline

This function is currently not used.

R	W	Attribute	Description
✓	✓	CLOSED	Connector closure. Parameter: vehicle class number

AttValue2 ([in] BSTR Attribute, [in] VARIANT Parameter1, [in] VARIANT Parameter2, [out, retval] VARIANT *pValue)

Returns a link attribute with two parameters. Please get the language independent attribute tag from the table below.

Parameters

```
[in] BSTR Attribute :            attribute name (see below)
```

```
[in] VARIANT Parameter1 :        first attribute dependent parameter (see below)
[in] VARIANT Parameter2 :        second attribute dependent parameter (see below)
[out, retval] VARIANT *pValue :  returned value of the attribute
```

Example

```
isclosed = link.AttValue2(„LANECLOSED", lanenr, vehclass)
```

AttValue2 ([in] BSTR Attribute, [in] VARIANT Parameter1, [in] VARIANT Parameter2, [in] VARIANT Value)

Sets a link attribute with two parameters. Please get the language independent attribute tag from the table below.

Parameters

```
[in] BSTR Attribute :            Attribute name (see below)
[in] VARIANT Parameter1 :        first attribute dependent parameter (see below)
[in] VARIANT Parameter2 :        second attribute dependent parameter (see below)
 [in] VARIANT Value :            Attribute value. (type according to attribute)
```

Example

```
link.AttValue2(„LANECLOSED", lanenr, vehclass) = TRUE
```

Attribute outline

R	W	Attribute	Description
✓	✓	LANECLOSED	Lane closure. Parameters: lane number and vehicle class number

2.10.2 Methods of the ILink Interface

GetSegmentResult ([in] BSTR Parameter, [in] long VehicleClass, [in] double XCoord, [in, defaultvalue(1)] int Lane, [in, defaultvalue(0)] BYTE Cumulative, [in, retval] VARIANT *pValue)

Returns the last collected result (see note below) of a previously configured segment. The link must have the segment evaluation flag set and a segment length defined. Use lane number 1 if evaluation data is not collected per lane (see ILinkEvaluation interface on page 174). Use a negative link coordinate if a two dimensional array of values for all segments is requested. In this case, the second dimension will be of size 2 and will contain the segment start coordinate and the result respectively. See the table below for a list of the possible parameter names.

Parameters

```
[in] BSTR Parameter :            parameter name (see below)
[in] long VehicleClass:          vehicle class number; 0 for all vehicle types
[in] double XCoord :             link coordinate (from 0.0 to link length; a negative
                                 value (i.e.-1.0) for a two dimensional array of values
                                 for each segment)
[in] int Lane :                  lane number (from 1 to n if "per lane" evaluations are
                                 collected, 1 otherwise)
```

```
[in] BYTE Cumulative:            cumulative evaluation flag (false as a default)
[out, retval] VARIANT *pValue :  returned value (real) or array of values (of reals)
```

Example

```
val = link.GetSegmentResult("SPEED", 0, 0.0)  ' first segment result for lane 1 or all
link if data is not being collected per lane
vals = link.GetSegmentResult("SPEED", 0, -1)  ' array of results for each segment.
FOR i = LBOUND(vals) TO UBOUND(vals)
  segindex = vals(i, 0)  ' segment index
  val = vals(i, 1)  'segment result
NEXT i
```

Parameters outline

Attribute	Description
DENSITY	Average density (current unit selection)
DELAY	Average relative lost time [s/s]
NVEHICLES	Number of vehicles (cumulative value of VOLUME). VOLUME must be activated within the evaluation configuration.
SPEED	Average speed (current unit selection)
VOLUME	Average volume [veh/h]

The results refer to the data collected during the last completed time interval. The data of a time interval is available immediately after the last time step of the interval has been simulated but not if this is the last time step of the simulation period as well.

To get results, a link evaluation configuration must be defined in VISSIM and stored in a *.sak file before requests can be done. The Offline Analysis option for Link Evaluation must be also enabled. Otherwise an error message will be returned ("The specified configuration is not defined within VISSIM").

GetVehicles ([out, retval] IVehicles** ppVehicles)

Returns a collection with all vehicles currently travelling on this link. This method is suposed to be called during a simulation. The returned collection actues as a reference to the dynamic list of vehicles on the link and is subjected to changes on each simulation step. (see note 92 for a more detailed explanation).

Parameters

```
[out, retval] IVehicles** ppVehicles :   returned Vehicles object.
```

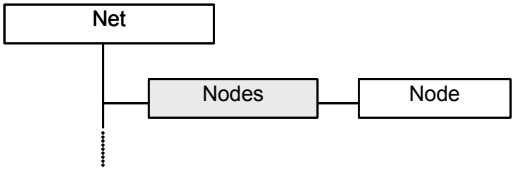
Example

```
vehicles = link.GetVehicles
FOR EACH v IN vehicles
  IF v.AttValue("SPEED") > 100.0 THEN
    v.AttValue("COLOR") = RGB(255, 0, 0)
```

```
    END IF
  NEXT v
```

2.11 Nodes

The Nodes object is a collection of Node objects. It belongs to the Net object and can be accessed through the Nodes property of the INet interface.



It contains all nodes of the loaded network and enables iteration through the collection or individual access to a Node object.

Example

Instantiation of Nodes object and access to all its Node objects:

```
Dim vissim As Vissim
Dim nodes As Nodes
SET vissim = NEW Vissim
vissim.LoadNet "c:\vissim\daten\example.inp"
Set nodes = Vissim.Net.Nodes

FOR EACH node IN nodes  'access to the _NewEnum to create an enumeration
  …
NEXT node

'or also:

FOR i = 1 TO nodes.Count
SET node = nodes(i)   'or nodes.Item(i)
…
NEXT i
```

2.11.1 Properties of the INodes Interface

_NewEnum ([out, retval] LPUNKNOWN *ppEnum)

This property creates a collection (or enumeration) of all Node objects. Once a collection is created, individual members can be returned using the GetNodeByNumber method, while the entire collection can be iterated using a FOR ...TO ... NEXT statement. In Visual Basic you can use the statement FOR EACH…NEXT without using the property _NewEnum, which VB calls internally. (see Nodes example above).

Parameters

`[out, retval] LPUNKNOWN *ppEnum : returned Enumeration`

Item ([in] VARIANT Index, [out, retval] INode **ppNode)

Returns a single Node of the collection selected by position. This is useful only when all nodes shall be accessed (see the Nodes example above). To

select a node by its identifier number use GetNodeByNumber (see below). Since Item is the default property for a collection the following commands are equivalent:

```
SET node = nodes(1)
SET node = nodes.Item(1)
```

Parameters

```
[in] long Index:                index between 1 and Nodes.Count
[out, retval] INode **ppNode:   returned Node object
```

Count ([out, retval] long *pCount)

Returns the number of Node objects in the collection. See the Nodes example above.

Parameters

```
[out, retval] long *pCount :    returned number of objects.
```

2.11.2 Methods of the INodes Interface

GetNodeByNumber ([in] long Number, [out, retval] INode **ppNode)

Returns the Node object with the identifier number Number.

Parameters

```
[in] long Number:               node identifier
[out, retval] INode **ppNode:   returned Node object
```

Example

```
DIM node AS Node
SET node =nodes.GetNodeByNumber(101)
```

2.11.3 Methods of the Interface INodes

GetResult ([in] double Time, [in] BSTR Parameter, [in] BSTR Function, [in] long VehicleClass, [out, retval] VARIANT *pValue)

This method returns the collected result for all nodes and for the requested named parameter (see parameters table below) and vehicle class. The returned value refers to the current collected result for the time interval enclosing the specified time point. The Function parameter has currently no meaning.

Parameters

```
[in] double Time :              time point in seconds
[in] BSTR Parameter :           parameter name (see below)
[in] BSTR Function :            not used
[in] long VehicleClass:         vehicle class number. 0 for all vehicle types
[out, retval] VARIANT *pValue:  returned value (real number)
```

Example

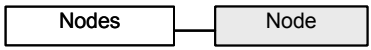
```
nv = nodes.GetResult(600, „NVEHICLES", "", 0) ' system total vehicle throughput
```

Parameters outline

Parameter	Description
DELAY	Average total delay per vehicle [s]
PERSONSDELAY	Average total delay per person [s]
NPERSONS	Persons throughput (Number of people)
NSTOPS	Average number of stops per vehicle [s]
NVEHICLES	Vehicles throughput (Number of vehicles)
QUEUELENGTHAVG	Average queue length (current unit selection)
QUEUELENGTHMAX	Maximum queue length (current unit selection)
STOPPEDDELAY	Average stand still time (stopped delay) per vehicle [s]

To get results, the Offline Analysis option for nodes must be enabled. Otherwise the result will be 0.0.

2.12 Node



A Node object represents a node element and belongs to the Nodes object. It can be accessed through the Nodes object in two ways:

- access via iteration through the collection

```
DIM nodes As Nodes
DIM node As Node
SET nodes = Vissim.Net.Nodes
FOR EACH node IN nodes
List.AddItem node.ID
NEXT node
```

- individual access via identifier

```
DIM nodes As Nodes
DIM node As Node
SET nodes = Vissim.Net.Nodes
SET node = nodes. GetNodeByNumber(101)
```

The Node object enables access to the properties of the node through the INode interface.

2.12.1 Properties of the INode Interface

ID ([out, retval] long *pID)

Returns the node identifier number. If the Node object doesn't refer to a valid VISSIM node element anymore the returned value is 0.

Parameters

```
[out, retval] long *pID :returned identifier.
```

Example

```
id = node.ID
```

Name ([out, retval] BSTR *pName)

Returns the node name.

Parameters

```
[out, retval] BSTR *pName :      returned name.
```

Example

```
name = node.Name
```

Name ([in] BSTR Name)

Sets the node name. Maximal 255 charatcters.

Parameters

```
[in] BSTR Name : new name.
```

Example

```
node.Name = „Durlachertor"
```

AttValue ([in] BSTR Attribute, [out, retval] VARIANT *pValue)

Returns a node attribute. Please get the language independent attribute tag from the table at the end of this section.

Parameters

```
[in] BSTR Attribute :            attribute name (see below)
[out, retval] VARIANT *pValue :  returned value of the attribute
```

Example

```
node = node.AttValue("NAME")
```

AttValue ([in] BSTR Attribute, [in] VARIANT Value)

Sets a node attribute. Please get the language independent attribute tag from the table at the end of this section.

Parameters

```
[in] BSTR Attribute :    Attribute name (see below)
[in] VARIANT Value :     Attribute value. (type according to attribute)
```

Example

```
node.AttValue(„NAME") = "Piccadilly Circus"
```

Attribute outline

R	W	Attribute	Description
✓		ID	Identifier number
	✓	NAME	Name

2.12.2 Methods of the Interface INode

GetResult ([in] double Time, [in] BSTR Parameter, [in] BSTR Function, [in] long VehicleClass, [out, retval] VARIANT *pValue)

This method returns collected result for the node total (all turn relations) and for the requested named parameter (see parameters table below) and vehicle class. The returned value refers to the current collected result for the time interval enclosing the specified time point. The Function parameter has currently no meaning.

Parameters

```
[in] double Time :               time point in seconds
[in] BSTR Parameter :            parameter name (see below)
[in] BSTR Function :             not used
[in] long VehicleClass:          vehicle class number. 0 for all vehicle types
[out, retval] VARIANT *pValue:   returned value (real number)
```

Example

```
nv = node.GetResult(600, „NVEHICLES“, “”, 0) ‘vehicle throughput
del = node.GetResult(600, “DELAY”, “”, 1) ‘average delay per vehicle of vehicle class 1
```

GetMovementResult ([in] double Time, [in] long FromLink, [in] long ToLink [in] BSTR Parameter, [in] BSTR Function, [in] long VehicleClass, [out, retval] VARIANT *pValue)

This method returns the collected result for the specified turn relation and for the requested named parameter (see parameters table below) and vehicle class. The returned value refers to the data collected up to this moment for the time interval enclosing the specified time point. The Function parameter has currently no meaning.

Parameters

```
[in] double Time :              time point in seconds
[in] long FromLink :            number of the link entering the node
[in] long ToLink :              number of the link leaving the node
[in] BSTR Parameter :           parameter name (see below)
[in] BSTR Function :            not used
[in] long VehicleClass:         vehicle class number. 0 for all vehicle types
[out, retval] VARIANT *pValue:  returned value (real number)
```

Example

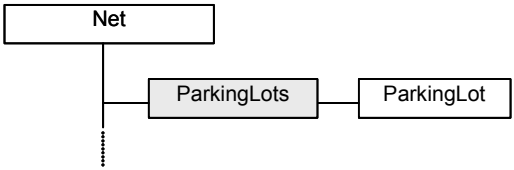
```
nv = n.GetResult(600,11,12, „NVEHICLES“, “”, 0) ‘vehicle throughput entering from link 11
and leaving trhough link 12
```

Parameters outline

Parameter	Description
DELAY	Average total delay per vehicle [s]
PERSONSDELAY	Average total delay per person [s]
NPERSONS	Persons throughput (Number of people)
NSTOPS	Average number of stops per vehicle [s]
NVEHICLES	Vehicles throughput (Number of vehicles)
QUEUELENGTHAVG	Average queue length (current unit selection)
QUEUELENGTHMAX	Maximum queue length (current unit selection)
STOPPEDDELAY	Average stand still time (stopped delay) per vehicle [s]

To get results, the Offline Analysis option for nodes must be enabled. Otherwise the result will be 0.0.

2.13 ParkingLots



The ParkingLots object is a collection of ParkingLot objects (see page 67). It belongs to the Net object and can be accessed through the ParkingLots property of the INet interface.

It contains all parking lots of the loaded network and allows the iteration through all of them as well as individual access (see also ParkingLot object).

Example

Instantiation of a ParkingLots object and access to all its ParkingLot objects:

```
DIM vissim As Vissim
DIM pls As ParkingLots
DIM pl As ParkingLot
SET vissim = NEW Vissim
vissim.LoadNet "c:\vissim\daten\example.inp"
Set pls = Vissim.Net.ParkingLots
FOR EACH pl IN pls  'access to _NewEnum to create an enumeration
  …
NEXT pl

'or also:

FOR i = 1 TO pls.Count
SET pl = pls(i)  'or pls.Item(i)
…
NEXT i
```

2.13.1 Properties of the IParkingLots Interface

_NewEnum ([out, retval] LPUNKOWN *ppEnum)

This property creates a collection (or enumeration) of all ParkingLot objects. Once a collection is created, individual members can be returned using the GetParkingLotByNumber method, while the entire collection can be iterated using a FOR ...TO ... NEXT statement. In Visual Basic you can use the statement FOR EACH…NEXT without using the property _NewEnum, which VB calls internally. (see ParkingLots example above).

Parameters

`[out, retval] LPUNKNOWN *ppEnum : returned Enumeration`

Item ([in] VARIANT Index, [out, retval] IParkingLot **ppParkingLot)

Returns a single ParkingLot of the collection selected by position. This is useful only when all parking lots shall be accessed (see the ParkingLots example above). To select a parking lot by its identifier number use GetParkingLotByNumber (see below). Since Item is the default property for a collection the following commands are equivalent:

```
SET pl = pls(1)
SET pl = pls.Item(1)
```

Parameters

[in] long Index :	index between 1 and Count
[out, retval] IParkingLot **ppParkingLots:	returned ParkingLot object

Count ([out, retval] long *pCount)

Returns the number of ParkingLot objects in the collection. See the ParkingLots example above.

Parameters

[out, retval] long *pCount :	returned number of objects.

2.13.2 Methods of the IParkingLots Interface

GetParkingLotByNumber ([in] long Number, [out, retval] IParkingLot **ppParkingLot)

Returns the ParkingLot object with the identifier number Number.

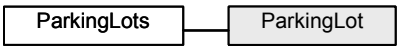
Parameters

[in] long Number :	identifier number
[out, retval] IParkingLot **ppParkingLot :	returned ParkingLot object

Example

```
SET pl = pls.GetParkingLotByNumber (2)
IF NOT (pl IS NOTHING) THEN
name = pl.AttValue("NAME")
END IF
```

2.14 ParkingLot



A ParkingLot object represents a parking lot element and belongs to the ParkingLots object. It can be accessed through the ParkingLots object to in two ways:

- access via iteration through the collection

```
DIM pl As ParkingLot
FOR EACH pls IN vissim.Net.ParkingLots
List.AddItem pl.ID
NEXT pl
```

- individual access via identifier

```
DIM pl As ParkingLot
SET pl = vissim.Net.ParkingLots.GetParkingLotByNumber (12)
```

The ParkingLot object enables access to the properties of the parking lot through the IParkingLot interface.

2.14.1 Properties of the Interface IParkingLot

ID ([out, retval] long *pID)

Returns the parking lot identifier number.

Parameters

`[out, retval] long *pID :returned identifier.`

Example

```
id = pl.ID
```

Name ([out, retval] BSTR *pName)

Returns the name of the parking lot.

Parameters

`[out, retval] BSTR *pName :  returned name.`

Example

```
name = pl.Name
```

Name ([in] BSTR Name)

Sets the name of the parking lot. Maximal 255 charatcters.

Parameters

`[in] BSTR Name : new name.`

Example

```
pl.Name = „XXX"
```

AttValue ([in] BSTR Attribute, [out, retval] VARIANT *pValue)

Returns a parking lot attribute. Please get the language independent attribute tag from the table at the end of this section.

Parameters

```
[in] BSTR Attribute :            attribute name (see table below)
[out, retval] VARIANT *pValue :  returned value of the attribute
```

Example

```
name = pl.AttValue(„NAME")
```

AttValue ([in] BSTR Attribute, [in] VARIANT Value)

Sets a parking lot attribute. Please get the language independent attribute tag from the table at the end of this section.

Parameters

```
[in] BSTR Attribute :    attribute name (see below)
[in] BSTR Value :        attribute value. (type according to attribute)
```

Example

```
pl.AttValue(„NAME")=„XXX"
```

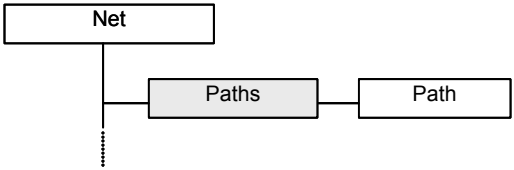
Attribute outline

R	W	Attribute	Description
✓		ID	Identifier number
✓	✓	NAME	Name
✓		LINK	Identifier number of the position link
✓		OCCUPANCY	Amount of currently parked vehicles (see note below)
✓		NVEHICLES	Total number of vehicles in the parking lot (see note below)

The named attribute NVEHICLES considers total loaded traffic demand independently of the user defined parking lot initial occupancy. On the other hand. The attribute OCCUPANCY doesn't consider the intial traffic load but the user defined parking lot initial occupancy together with the arrived and parked vehicles which are considered to determine if a parking lot is full.

2.15 Paths

The Paths object is a collection of Path objects. It belongs to the Net object and can be accessed through the Paths property of the INet interface.



It contains all paths inserted using this interface (not the dynamic assignment paths, see note at the end of this chapter) and enables iteration through the collection or individual access to a Path object.

Example

Instantiation of Paths object and access to all its Path objects (see page 73):

```
DIM vissim AS Vissim
DIM paths AS Paths
DIM path AS Path
SET vissim = NEW Vissim
vissim.LoadNet "c:\vissim\daten\example.inp"
SET paths = Vissim.Net.Paths
FOR EACH path IN paths  'access to the method _NewEnum to create an enumeration
  …
NEXT path

'or also:

FOR i = 1 TO paths.Count
SET path = paths(i)  'or paths.Item(i)
…
NEXT i
```

2.15.1 Properties of the IPaths Interface

_NewEnum ([out, retval] LPUNKNOWN *ppEnum)

This property creates a collection (or enumeration) of all Path objects. Once a collection is created, individual members can be returned using the GetPathByNumber method, while the entire collection can be iterated using a FOR ...TO ... NEXT statement. In Visual Basic you can use the statement FOR EACH…NEXT without using the property _NewEnum, which VB calls internally. (see Paths example above).

Parameters

```
[out, retval] LPUNKNOWN *ppEnum : returned Enumeration
```

Item ([in] VARIANT Index, [out, retval] IPath **ppPath)

Returns a single Path of the collection selected by position. This is useful only when all paths shall be accessed (see the Paths example above). To

select a path by its identifier number use GetPathByNumber (see below). Since Item is the default property for a collection the following commands are equivalent:

```
SET path = paths(1)
SET path = paths.Item(1)
```

Parameters

```
[in] long Index :                index between 1 and Paths.Count
[out, retval] IPath **ppPath :   returned Path object
```

Count ([out, retval] long *pCount)

Returns the number of Path objects in the collection. See the Paths example above.

Parameters

```
[out, retval] long *pCount :     returned number of objects.
```

2.15.2 Methods of the IPaths Interface

GetPathByNumber ([in] long Number, [out, retval] IPath **ppPath)

Returns the Path object with the identifier number Number.

Parameters

```
[in] long Number :               identifier number
[out, retval] IPath **ppPath :   returned Path object
```

Example

```
DIM path AS Path
SET path = paths.GetPathByNumber (11)
```

AddPathAsNodeSequence ([in] long Number, [in] VARIANT Nodes, [out, retval] IPath **ppPath)

This method creates and inserts a new path based on the given sequence of nodes in the array Nodes. If 0 is passed as the identifier number the method will set the path number automatically. A non valid node sequence will produce an error message (see the possible error messages below and the page 204 for error handling). An origin parking lot must be available between the first and second node and a destination parking lot in-between the two last nodes.

Parameters

```
[in] long Number :               identifier number or 0
[in] VARIANT Nodes :             array of node identifier numbers
[out, retval] IPath **ppPath :   returned Path object
```

Example

```
DIM path AS Path
path = paths.AddPathAsNodeSequence (1, Array(10, 20, 30, 40))
```

RemovePath ([in] long Number)

This method removes the Path object identified with Number from the collection of paths inserted with the COM interface. Nothing is done if the path doesn't exist in this collection. If the path is used for VISSIM's dynamic assignment, it belongs to the dynamic assignment path collection (*.weg file) and it will not be removed from this collection. A path can be removed during a simulation; in this case it won't be possible to assign to the path to any further vehicle after the method is called and it will be physically removed from the network as soon as no vehicles are using it.

Parameters

```
[in] long Number :      identifier number
```

Example

```
paths.RemovePath (11)
```

2.15.3 Useful to Know

Path network element

Although paths are static network elements, it is possible to insert and use paths during the simulation at any simulation step using the IPaths interface.

Paths and dynamic assignment

Paths inserted using the IPaths interface are not collected to the path collection of the dynamic assignment (*.WEG file) if they are not found by the short path search algorithm (if they are found they are treated as normal dynamic assignment paths).

Path insertion, using the IPaths interface, doesn't consider any type of closure (neither link, edge nor route closures).

Paths inserted using the IPaths interface are never considered long detours.

Only the paths inserted before the simulation start will affect the option for high cost path rejection (Path Search – Reject paths with total cost higher by % than total cost of the best path).

The path numbers used by the IPath interface don't necessarily correspond to the path numbers used by the path evaluation file (*.WGA).

2.15.4 Possible Errors

“An object with the same identifier number already exists.”

This message will appear when using an identifier number assigned to an existing path to add a path. Correct the number or use automatic identifier assignment (0).

“The object with specified identifier number doesn’t exists.”

This message will appear when using non existing node identifier numbers.

"No origin parking lot between the first two nodes."

A path must start on a parking lot. The existence of a parking lot in-between the first two nodes must be guaranteed.

"No destination parking lot between the last two nodes."

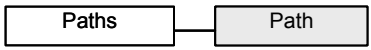
A path must end up on a parking lot. The existence of a parking lot in-between the last two nodes must be guaranteed.

“No valid node sequence.”

This message will occur when:

1. The node sequence can’t meet any valid path.
2. The node sequence is valid for more that one path. Use intermediate nodes in-between to solve the ambiguity.

2.16 Path



A Path object represents a path element and belongs to the Paths object. It can be accessed through the Paths object in two ways:

- access via iteration through the collection

```
DIM paths AS Paths
DIM path AS Path
SET paths = Vissim.Net.Paths
FOR EACH path IN paths
List.AddItem path.ID
NEXT path
```

- individual access via identifier

```
DIM paths AS Paths
DIM path AS Path
SET paths = Vissim.Net.Paths
SET path = paths.GetPathByNumber (11)
```

The Path object enables access to the properties of the path through the IPath interface.

As a difference to other network elements, the paths inserted through the IPaths interface are not strictly the same as the dynamic assignment paths (see the “Useful to Know” section on page 71 for more information)

2.16.1 Properties of the IPath Interface

ID ([out, retval] long *pID)

Returns the path identifier number. If the Path object doesn’t refer to a valid VISSIM path element anymore the returned value is 0.

Parameters

```
[out, retval] long *pID :returned identifier.
```

Example

```
id = path.ID
```

Name ([out, retval] BSTR *pName)

Returns the path name.

Parameters

```
[out, retval] BSTR *pName :      returned name.
```

Example

```
name = path.Name
```

Name ([in] BSTR Name)

Sets the path name.

Parameters

[in] BSTR Name : new name.

Example

path.Name = „Markplatz_Stumpftsr"

AttValue ([in] BSTR Attribute, [out, retval] VARIANT *pValue)

Returns a path attribute. Please get the language independent attribute tag from the table at the end of this section.

Parameters

[in] BSTR Attribute : attribute name (see below)
[out, retval] VARIANT *pValue : returned value of the attribute

Example

length = path.AttValue(„LENGTH")

AttValue ([in] BSTR Attribute, [in] VARIANT Value)

Sets a path attribute. Please get the language independent attribute tag from the table at the end of this section.

Parameters

[in] BSTR Attribute : attribute name (see below)
[in] BSTR Value : attribute value (type, according to the attribute)

Example

path.AttValue(„NAME") = „Markplatz_Stumpfstrasse"

Attribute outline

R	W	Attribute	Description
✓		ID	Identifier number
✓	✓	NAME	Name
✓		LENGTH	Length in the units of the current options

2.17 DrivingBehaviorParSets

The DrivingBehaviorParSets object is a collection of DrivingBehaviorParSet objects (see page 77).

It belongs to the Net object and can be accessed through the DrivingBehaviorParSets property of the INet interface. It contains all driving behavior parameter sets of the loaded network and allows the iteration through all of them as well as individual access (see also DrivingBehaviorParSet object).

Example

Instantiation of a DrivingBehaviorParSets object and access to all its DrivingBehaviorParSet objects:

```
DIM vissim As Vissim
DIM dbpss As DrivingBehaviorParSets
DIM dbps As DrivingBehaviorParSet
SET vissim = NEW Vissim
vissim.LoadNet "c:\vissim\daten\example.inp"
Set dbpss = Vissim.Net.DrivingBehaviorParSets
FOR EACH dbps IN dbpss  'access to _NewEnum to create an enumeration
  …
NEXT dbps

'or also:

FOR i = 1 TO dbpss.Count
SET dbps = dbpss(i)  'or dbpss.Item(i)
…
NEXT i
```

2.17.1 Properties of the IDrivingBehaviorParSets Interface

_NewEnum ([out, retval] LPUNKOWN *ppEnum)

This property creates a collection (or enumeration) of all DrivingBehaviorParSet objects. Once a collection is created, individual members can be returned using the GetDrivingBehaviorParSetByNumber method, while the entire collection can be iterated using a FOR ...TO ... NEXT statement. In Visual Basic you can use the statement FOR EACH…NEXT without using the property _NewEnum, which VB calls internally. (see DrivingBehaviorParSets example above).

Parameters

`[out, retval] LPUNKNOWN *ppEnum : returned Enumeration`

Item ([in] VARIANT Index, [out, retval] IDrivingBehaviorParSet **ppDBPS)

Returns a single DrivingBehaviorParSet of the collection selected by position. This is useful only when all driving behavior parameter sets shall be accessed (see the DrivingBehaviorParSets example above). To select a driving behavior parameter set by its identifier number use GetDrivingBehaviorParSetByNumber (see below). Since Item is the default property for a collection the following commands are equivalent:

```
SET dbps = dbpss(1)
SET dbps = dbpss.Item(1)
```

Parameters

[in] long Index :	index between 1 and Count
[out, retval] IDrivingBehaviorParSet **ppDBPS:	returned DrivingBehaviorParSet object

Count ([out, retval] long *pCount)

Returns the number of DrivingBehaviorParSet objects in the collection. See the DrivingBehaviorParSets example above.

Parameters

[out, retval] long *pCount :	returned number of objects.

2.17.2 Methods of the IDrivingBehaviorParSets Interface

GetDrivingBehaviorParSetByNumber ([in] long Number, [out, retval] IDrivingBehaviorParSet **ppDBPS)

Returns the DrivingBehaviorParSet object with the identifier number Number.

Parameters

[in] long Number :	identifier number
[out, retval] IDrivingBehaviorParSet **ppDBPS :	returned DrivingBehaviorParSet object

Example

```
SET dbps = dbpss.GetDrivingBehaviorParSetByNumber(2)
IF NOT (dbps IS NOTHING) THEN
name = dbps.AttValue("NAME")
END IF
```

2.18 DrivingBehaviorParSet

A DrivingBehaviorParSet object represents a driving behavior parameter set element and belongs to the DrivingBehaviorParSets object.

It can be accessed through the DrivingBehaviorParSets object to in two ways:

- access via iteration through the collection

```
DIM dbps As DrivingBehaviorParSet
FOR EACH dbps IN vissim.Net.DrivingBehaviorParSets
List.AddItem dbps.ID
NEXT dbps
```

- individual access via identifier

```
DIM dbps As DrivingBehaviorParSet
SET dbps = vissim.Net.DrivingBehaviorParSets.GetDrivingBehaviorParSetByNumber(2)
```

The DrivingBehaviorParSet object enables access to the properties of the driving behavior parameter set through the IDrivingBehaviorParSet interface.

2.18.1 Properties of the Interface IDrivingBehaviorParSet

ID ([out, retval] long *pID)

Returns the identifier number of the driving behavior parameter set.

Parameters

```
[out, retval] long *pID :returned identifier.
```

Example

```
id = dbps.ID
```

Name ([out, retval] BSTR *pName)

Returns the name of the driving behavior parameter set.

Parameters

```
[out, retval] BSTR *pName :      returned name.
```

Example

```
name = dbps.Name
```

Name ([in] BSTR Name)

Sets the name of the driving behavior parameter set. Maximal 255 characters.

Parameters

```
[in] BSTR Name : new name.
```

Example

```
dbps.Name = „XXX"
```

AttValue ([in] BSTR Attribute, [out, retval] VARIANT *pValue)

Returns a driving behavior parameter set attribute. Please get the language independent attribute tag from the table at the end of this section.

Parameters

```
[in] BSTR Attribute :            attribute name (see table below)
[out, retval] VARIANT *pValue :  returned value of the attribute
```

Example

```
name = dbps.AttValue(„NAME")
```

AttValue ([in] BSTR Attribute, [in] VARIANT Value)

Sets a driving behavior parameter set attribute. Please get the language independent attribute tag from the table at the end of this section.

Parameters

```
[in] BSTR Attribute :    attribute name (see below)
[in] BSTR Value :        attribute value. (type according to attribute)
```

Example

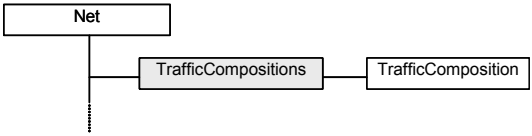
```
dbps.AttValue(„NAME")=„XXX"
```

Attribute outline

R	W	Attribute	Description
✓		ID	Identifier number
✓	✓	NAME	Name
✓	✓	CC0	Standstill Distance (current distance units set)
✓	✓	CC1	Headway Time [s]
✓	✓	CC2	'Following' Variation (current distance units set)
✓	✓	CC3	Threshold of entering 'Following' [-]
✓	✓	CC4	Negative 'Following' Threshold [-]
✓	✓	CC5	Positive 'Following' Threshold [-]
✓	✓	CC6	Speed dependeny of Oscillation
✓	✓	CC7	Oscillation Acceleration [m/s^2]
✓	✓	CC8	Standstill Acceleration [m/s^2]
✓	✓	CC9	Acceleration at 80 km/h [m/s^2]

2.19 TrafficCompositions

The TrafficCompositions object is a collection of TrafficComposition objects (see page 81).



It belongs to the Net object and can be accessed through the TrafficCompositions property of the INet interface. It contains all traffic compositions of the loaded network and allows the iteration through all of them as well as individual access (see also TrafficComposition object).

Example

Instantiation of a TrafficCompositions object and access to all its TrafficComposition objects:

```
DIM vissim As Vissim
DIM tcs As TrafficCompositions
DIM tc As TrafficComposition
SET vissim = NEW Vissim
vissim.LoadNet "c:\vissim\daten\example.inp"
Set tcs = Vissim.Net.TrafficCompositions
FOR EACH tc IN tcs  'access to _NewEnum to create an enumeration
  …
NEXT tc

'or also:

FOR i = 1 TO tcs.Count
SET tc = tcs(i)  'or tcs.Item(i)
…
NEXT i
```

2.19.1 Properties of the ITrafficCompositions Interface

_NewEnum ([out, retval] LPUNKOWN *ppEnum)

This property creates a collection (or enumeration) of all TrafficComposition objects. Once a collection is created, individual members can be returned using the GetTrafficCompositionByNumber method, while the entire collection can be iterated using a FOR ...TO ... NEXT statement. In Visual Basic you can use the statement FOR EACH…NEXT without using the property _NewEnum, which VB calls internally. (see TrafficCompositions example above).

Parameters

```
[out, retval] LPUNKNOWN *ppEnum : returned Enumeration
```

Item ([in] VARIANT Index, [out, retval] ITrafficComposition **ppTC)

Returns a single TrafficComposition of the collection selected by position. This is useful only when all traffic compositions shall be accessed (see the TrafficCompositions example above). To select a traffic composition by its identifier number use GetTrafficCompositionByNumber (see below). Since Item is the default property for a collection the following commands are equivalent:

```
SET tc = tcs(1)
SET tc = tcs.Item(1)
```

Parameters

[in] long Index :	index between 1 and Count
[out, retval] ITrafficComposition **ppTC:	returned TrafficComposition object

Count ([out, retval] long *pCount)

Returns the number of TrafficComposition objects in the collection. See the TrafficCompositions example above.

Parameters

[out, retval] long *pCount :	returned number of objects.

2.19.2 Methods of the ITrafficCompositions Interface

GetTrafficCompositionByNumber ([in] long Number, [out, retval] ITrafficComposition **ppTC)

Returns the TrafficComposition object with the identifier number Number.

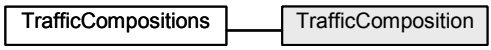
Parameters

[in] long Number :	identifier number
[out, retval] ITrafficComposition **ppTC :	returned TrafficComposition object

Example

```
SET tc = tcs.GetTrafficCompositionByNumber(2)
IF NOT (tc IS NOTHING) THEN
name = tc.AttValue("NAME")
END IF
```

2.20 TrafficComposition



A TrafficComposition object represents a traffic composition element and belongs to the TrafficCompositions object.

It can be accessed through the TrafficCompositions object to in two ways:

- access via iteration through the collection

```
DIM tc As TrafficComposition
FOR EACH tc IN vissim.Net.TrafficCompositions
List.AddItem tc.ID
NEXT tc
```

- individual access via identifier

```
DIM tc As TrafficComposition
SET tc = vissim.Net.TrafficCompositions.GetTrafficCompositionByNumber(2)
```

The TrafficComposition object enables access to the properties of the traffic composition through the ITrafficComposition interface.

2.20.1 Properties of the Interface ITrafficComposition

ID ([out, retval] long *pID)

Returns the identifier number of the traffic composition.

Parameters

```
[out, retval] long *pID :returned identifier.
```

Example

```
id = tc.ID
```

Name ([out, retval] BSTR *pName)

Returns the name of the traffic composition.

Parameters

```
[out, retval] BSTR *pName :      returned name.
```

Example

```
name = tc.Name
```

Name ([in] BSTR Name)

Sets the name of the traffic composition. Maximal 255 charatcters.

Parameters

```
[in] BSTR Name : new name.
```

Example

```
tc.Name = „XXX"
```

AttValue ([in] BSTR Attribute, [out, retval] VARIANT *pValue)

Returns a traffic composition attribute. Please get the language independent attribute tag from the table at the end of this section.

Parameters

```
[in] BSTR Attribute :            attribute name (see table below)
[out, retval] VARIANT *pValue :  returned value of the attribute
```

Example

```
name = tc.AttValue(„NAME")
```

AttValue ([in] BSTR Attribute, [in] VARIANT Value)

Sets a traffic composition attribute. Please get the language independent attribute tag from the table at the end of this section.

Parameters

```
[in] BSTR Attribute :    attribute name (see below)
[in] BSTR Value :        attribute value. (type according to attribute)
```

Example

```
tc.AttValue(„NAME")=„XXX"
```

Attribute outline

R	W	Attribute	Description
✓		ID	Identifier number
✓	✓	NAME	Name

AttValue1 ([in] BSTR Attribute, [in] VARIANT Parameter, [out, retval] VARIANT *pValue)

Returns a traffic composition attribute with one parameter. Please get the language independent attribute tag from the table below.

Parameters

```
[in] BSTR Attribute :            attribute name (see below)
[in] VARIANT Parameter :         attribute dependent parameter (see below)
[out, retval] VARIANT *pValue :  returned value of the attribute
```

AttValue1 ([in] BSTR Attribute, [in] VARIANT Parameter, [in] VARIANT Value)

Sets a traffic composition attribute with one parameter. Please get the language independent attribute tag from the table below.

Parameters

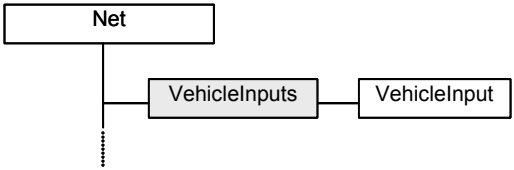
```
[in] BSTR Attribute :            Attribute name (see below)
[in] VARIANT Parameter :         attribute dependent parameter (see below)
[in] VARIANT Value :             Attribute value. (type according to attribute)
```

Attribute outline

R	W	Attribute	Description
✓	✓	RELATIVEFLOW	Relative flow for the specified vehicle type. Parameter: vehicle type number.

Changes will affect immediately if called during a simulation. For example, changing the relative flows will affect the vehicle types composition of each vehicle input that use it.

2.21 VehicleInputs



The VehicleInputs object is a collection of VehicleInput objects (see page 86). It belongs to the Net object and can be accessed through the VehicleInputs property of the INet interface.

It contains all vehicle inputs of the loaded network and allows the iteration through all of them as well as individual access (see also VehicleInput object).

Example

Instantiation of a VehicleInputs object and access to all its VehicleInput objects:

```
DIM vissim As Vissim
DIM vehins As VehicleInputs
DIM vehin As VehicleInput
SET vissim = NEW Vissim
vissim.LoadNet "c:\vissim\daten\example.inp"
Set inputs = Vissim.Net.VehicleInputs
FOR EACH vehin IN vehins 'access to _NewEnum to create an enumeration
  …
NEXT vehin

'or also:

FOR i = 1 TO vehins.Count
SET vehin = vehins(i)   'or vehins.Item(i)
…
NEXT i
```

2.21.1 Properties of the IVehicleInputs Interface

_NewEnum ([out, retval] LPUNKOWN *ppEnum)

This property creates a collection (or enumeration) of all VehicleInput objects. Once a collection is created, individual members can be returned using the GetVehicleInputByNumber method, while the entire collection can be iterated using a FOR ...TO ... NEXT statement. In Visual Basic you can use the statement FOR EACH…NEXT without using the property _NewEnum, which VB calls internally. (see VehilcleInputs example above).

Parameters

`[out, retval] LPUNKNOWN *ppEnum : returned Enumeration`

Item ([in] VARIANT Index, [out, retval] IVehicleInput **ppVehicleInput)

It returns a single VehicleInput of the collection selected by position. This is useful only when all vehicle inputs shall be accessed (see the VehicleInputs example above). To select a vehicle input by its identifier number use GetVehicleInputByNumber (see below). Since Item is the default property for a collection the following commands are equivalent:

```
SET vehin = inputs(1)
SET vehin = inputs.Item(1)
```

Parameters

[in] long Index :	index between 1 and Count
[out, retval] IVehicleInput **ppVehicleInputs:	returned VehicleInput object

Count ([out, retval] long *pCount)

Returns the number of VehicleInput objects in the collection. See the VehicleInputs example above.

Parameters

[out, retval] long *pCount :	returned number of objects.

2.21.2 Methods of the IVehicleInputs Interface

GetVehicleInputByNumber ([in] long Number, [out, retval] IVehicleInput **ppVehicleInput)

Returns the VehicleInput object with the identifier number Number.

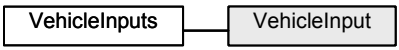
Parameters

[in] long Number :	identifier number
[out, retval] IVehicleInput **ppVehicleInput :	returned VehicleInput object

Example

```
SET vehin = vehins.GetVehicleInputByNumber (2)
IF NOT (vehin IS NOTHING) THEN
name = vehin.AttValue("NAME")
END IF
```

2.22 VehicleInput



A VehicleInput object represents a vehicle input element and belongs to the VehicleInputs object. It can be accessed through the VehicleInputs object to in two ways:

- access via iteration through the collection

```
DIM vehin As VehicleInput
FOR EACH vehin IN vissim.Net.VehicleInputs
List.AddItem vehin.ID
NEXT vehin
```

- individual access via identifier

```
DIM vehin As VehicleInput
SET vehin = vissim.Net.VehicleInputs.GetVehicleInputByNumber (12)
```

The VehicleInput object enables access to the properties of the vehicle input through the IVehicleInput interface.

2.22.1 Properties of the Interface IVehicleInput

ID ([out, retval] long *pID)

Returns the vehicle input identifier number.

Parameters

```
[out, retval] long *pID :returned identifier.
```

Example

```
id = vehin.ID
```

Name ([out, retval] BSTR *pName)

Returns the vehicle input name.

Parameters

```
[out, retval] BSTR *pName :      returned name.
```

Example

```
name = vehin.Name
```

Name ([in] BSTR Name)

Sets the input's name. Maximal 255 charatcters.

Parameters

```
[in] BSTR Name : new name.
```

Example

```
vehin.Name = „XXX"
```

AttValue ([in] BSTR Attribute, [out, retval] VARIANT *pValue)

Returns a vehicle input attribute. Please get the language independent attribute tag from the table at the end of this section.

Parameters

```
[in] BSTR Attribute :            attribute name (see table below)
[out, retval] VARIANT *pValue :  returned value of the attribute
```

Example

```
name = vehin.AttValue(„NAME“)
```

AttValue ([in] BSTR Attribute, [in] VARIANT Value)

Sets a vehicle input attribute. Please get the language independent attribute tag from the table at the end of this section.

Parameters

```
[in] BSTR Attribute :    attribute name (see below)
[in] BSTR Value :        attribute value. (type according to attribute)
```

Example

```
vehin.AttValue(„NAME“)=„XXX“
```

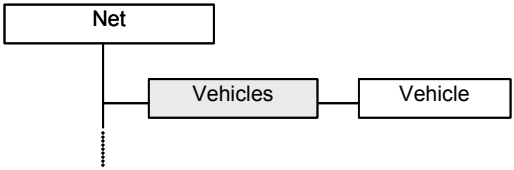
Attribute outline

R	W	Attribute	Description
✓		ID	Identifier number
✓	✓	NAME	Name
✓	✓	TIMEFROM	Time interval start [s]
✓	✓	TIMEUNTIL	Time interval end [s]
✓	✓	VOLUME	Volume [Veh/h]

Changes will affect immediately if called during a simulation. If the option “Generate exact number of vehicles” is checked, changing the time interval (using the attributes TIMEFROM or TIMEUNTIL) will force to regenerate the sequence of vehicles with the current attributes.

It is required that TIMEUNTIL is greater or equal than TIMEFROM when setting the interval. If the current interval is [100, 1000] and you want to set it to [1100, 2000] you are forced to set first TIMEUNTIL to 2000 and afterwards TIMEFROM to 1100, otherwise you will get an error message.

2.23 Vehicles



The Vehicles object is a collection of Vehicle objects (see page 94). It belongs to the Net object and can be accessed through the Vehicles property of the INet interface.

It contains all vehicles currently traveling on the network during a simulation, including the parked ones. It enables iteration through the collection or individual access to a Vehicle object.

Example

Instantiation of a Vehicles object and access to all its Vehicle objects:

```
DIM vissim As Vissim
DIM vehicles As Vehicles
DIM vehicle As Vehicle
SET vissim = NEW Vissim
vissim.LoadNet "c:\vissim\daten\example.inp"
Set vehicles = Vissim.Net.Vehicles
FOR EACH vehicle IN vehicles  'access to _NewEnum to create an enumeration
  …
NEXT vehicle

'or also:

FOR i = 1 TO vehicles.Count
SET vehicle = vehicles(i)  'or vehicles.Item(i)
…
NEXT i
```

2.23.1 Properties of the IVehicles Interface

_NewEnum ([out, retval] LPUNKOWN *ppEnum)

This property creates a collection (or enumeration) of all Vehicle objects. Once a collection is created, individual members can be returned using the GetVehicleByNumber method, while the entire collection can be iterated using a FOR ...TO ... NEXT statement. In Visual Basic you can use the statement FOR EACH…NEXT without using the property _NewEnum, which VB calls internally. (see Vehicles example above).

Parameters

`[out, retval] LPUNKNOWN *ppEnum : returned Enumeration`

Item ([in] VARIANT Index, [out, retval] IVehicle **ppVehicle)

Returns a single Vehicle of the collection selected by position. This is useful only when all vehicles shall be accessed (see the Vehicles example above). To select a vehicle by its identifier number use GetVehicleByNumber (see below). Since Item is the default property for a collection the following commands are equivalent:

```
SET vehicle = vehicles(1)
SET vehicle = vehicles.Item(1)
```

Parameters

[in] long Index :	index between 1 and vehicles.Count
[out, retval] IVehicle **ppVehicles:	returned Vehicle object

Count ([out, retval] long *pCount)

Returns the number of Vehicle objects in the collection. See the Vehicles example above.

Parameters

[out, retval] long *pCount :	returned number of objects.

IDs ([in, defaultvalue(0)] BSTR Attribute, [in, defaultvalue(0)] VARIANT Value, [out, retval] VARIANT *pIDs)

Returns an array of vehicle IDs complying the condition “Attribute = Value” if a vehicle attribute an a value (see the attribute table on page 95) are given, otherwise an array with the IDs of all vehicles in the collection is returned.

Parameters

[in] BSTR Attribute :	attribute name (see IVehicle attribute table)
[in] VARIANT Value :	attribute value or array of values (type according to attribute)
[out, retval] VARIANT *pIDs :	returned array of IDs.

Example

```
ids = vehicles.IDs(„TYPE“, 1) ‘array with the IDs of all vehicles of type 1
```

2.23.2 Methods of the IVehicles Interface

GetVehicleByNumber ([in] long Number, [out, retval] IVehicle **ppVehicle)

Returns the Vehicle object with the identifier number Number.

Parameters

[in] long Number :	identifier number
[out, retval] IVehicle **ppVehicle :	returned Vehicle object

Example

```
DIM vehicle AS Vehicle
SET vehicle = vehicles.GetVehicleByNumber (11)
IF NOT (vehicle IS NOTHING) THEN
speed = vehicle.AttValue(“SPEED”)
```

```
END IF
```

GetMultiAttValues ([in] VARIANT IDs, [in] BSTR Attribute, [out] VARIANT *pValues)

Fills the array pValues with the respective attribute values for the requested IDs (use the Variant Empty to get it for all vehicles in the collection). Please get the language independent attribute tag from the table on page 95.

Parameters

```
[in] VARIANT IDs :       array of vehicle IDs or vehicle ID
[in] BSTR Attribute :    attribute name (see IVehicle attribute table)
[out] VARIANT *pValues : returned array of values
```

Example

```
vehs.GetMultiAttValues Array(1, 2, 3, 4, 5), "SPEED", values)
For i = LBound(values) To UBound(values)
  val = values (i)
Next i
```

SetMultiAttValues ([in] VARIANT IDs, [in] BSTR Attribute, [in] VARIANT Values)

Sets the attribute values of every required vehicle in the passed array of IDs with their respective new values (use the Variant Empty to set it for all vehicles in the collection). If a value, instead of an array of values, is passed the same value will be used for every vehicle. Please get the language independent attribute tag from the table on page 95.

Parameters

```
[in] VARIANT IDs :       array of vehicle IDs or vehicle ID
[in] BSTR Attribute :    attribute name (see IVehicle attribute table)
[in] VARIANT Values :    array of values or value (type according to attribute)
```

Example

```
vehs.SetMultiAttValues Empty, "COLOR", RGB(255, 0, 0)
```

AddVehicleInParkingLot ([in] long Type, [in] long ParkingID, [out, retval] IVehicle **ppVehicle)

This method places a new vehicle of type Type on the parking lot identified with the number ParkingID. If a non valid type or parking lot number is passed an error will be returned and no vehicle will be created. The vehicle gets the current simulation time as its departure time (0 if no simulation is running) and will depart as soon as it has been assigned a path (see attribute “PATH” of IVehicle on page 95).

Parameters

```
[in] long Type :                    vehicle type according to the current network
[in] long ParkingID :               parking lot identifier number
[out, retval] IVehicle **ppVehicle : returned Vehicle object
```

Example

```
DIM vehicle AS Vehicle
vehicle = vehicles.AddVehicleInParkingLot (1, 10)
```

AddVehicleAtLinkCoordinate ([in] long Type, [in] double DesiredSpeed, [in] long Link, [in] int Lane, [in] double XCoord, [in, defaultvalue(1)] BYTE Interaction, [out, retval, defaultvalue(0)] IVehicle **ppVeh)

This method places a new vehicle of type Type on the lane Lane at the link coordinate XCoord of link Link. The standard mode of this method will let the placed vehicle interact with their neighbors and set its speed according to the current situation and to the DesiredSpeed. It is also possible to set the flag Interaction to false (0) and let the vehicle ignore the traffic flow conditions, traveling directly at its desired speed. As a difference to the method AddVehicleInParkingLot, the vehicles placed this way don't travel with a specific path and they will depart on the next simulation second. Therefore calling this method with the simulation not running will cause an error.

Parameters

```
[in] long Type :                 vehicle type according to the current network
[in] double DesiredSpeed:        desired speed (speed if interaction is disabled)
[in] long Link :                 link identifier number
[in] int Lane :                  lane number (from 1 to n according to the link)
[in] double XCoord :             link coordinate (from 0.0 to link length)
[in] BYTE Interaction:           interaction flag (true as a default)
[out, retval] IVehicle **ppVeh : returned Vehicle object
```

Example

```
DIM vehicle AS Vehicle
vehicle = vehicles.AddVehicleAtLinkCoordinate (1, 40.0, 1000, 1, 20.0)
```

RemoveVehicle ([in] long Number)

Removes the vehicle with the number Number from the network, whether it is parked or not.

Parameters

```
[in] long Number :       vehicle identifier number
```

Example

```
vehicles.RemoveVehicle (112)
```

GetQueued ([out, retval] IVehicles **ppVehicles)

This method returns a collection of Vehicle objects containing all vehicles that are in a queued condition (according to the current queue configuration).

Parameters

```
[out, retval] IVehicles **ppVehicles :    returned Vehicles object
```

Example

```
DIM queued_vehicles AS Vehicles
queued_vehicles = vehicles.GetQueued
```

GetArrived ([out, retval] IVehicles **ppVehicles)

This method returns a collection of vehicle objects containing all vehicles that have reached their destination parking lot (according to the assigned path) in the last simulation step. Vehicles created through OD-matrices

disappear when reaching their parking lot destination and thus can not be accessed with this method.

Parameters

```
[out, retval] IVehicles **ppVehicles :    returned Vehicles object
```

Example

```
DIM arrived_vehicles AS Vehicles
arrived_vehicles = vehicles.GetArrived
```

GetParked ([out, retval] IVehicles **ppVehicles)

This method returns a collection of vehicle objects containing all vehicles that are parked in parking lots. (Vehicles defined by a matrix file are visible only in their origin parking lot, vehicles defined by a trip chain file or set into the network through the COM interface are visible in their destination parking lot as well.)

Parameters

```
[out, retval] IVehicles **ppVehicles :    returned Vehicles object
```

Example

```
DIM parked_vehicles AS Vehicles
parked_vehicles = vehicles.GetParked
```

2.23.3 Useful to know

Attributes

The named attributes for the AttValue methods are the same as the ones used in the IVehicle interface (see attribute table on page 95)

Vehicles collections

Vehicles are dynamic network elements (i.e. the are created, removed and changed during a simulation run). Therefore, vehicle collections are also dynamic and change their internal elements (vehicles) and order of the elements during the simulation. In a client programming context this means that a collection variable refers to the current network status. In the following example the vehicles collection variable queued_vehicles may not contain the same vehicles after the simulation step (i.e. n_before and n_after could differ):

```
DIM queued_vehicles AS Vehicles
queued_vehicles = vehicles.GetQueued
n_before = queued_vehicles.Count
sim.RunSimulationStep
n_after = queued_vehicles.Count
```

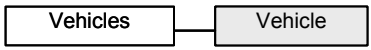
AddVehicleAtLinkCoordinate

When adding a vehicle to a link coordinate the vehicle will be appear at that coordinate with the next simulation step. Use the method Redraw of IGraphics to see the vehicle.

Virtual memory option

The virtual memory check box for parked vehicles within the dynamic assignment options, has no effect on vehicles inserted with the method AddVehicleInParkingLot().

2.24 Vehicle



A Vehicle object represents a single vehicle and belongs to the Vehicles object. It can be accessed through the Vehicles object in two ways:

- access via iteration through the collection

```
DIM vehicle As Vehicle
FOR EACH vehicle IN vissim.Net.Vehicles
List.AddItem vehicle.ID
NEXT vehicle
```

- individual access via identifier number

```
DIM vehicle As Vehicle
SET vehicle = vissim.Net.Vehicles.GetVehicleByNumber (101)
```

The Vehicle object enables access to the properties of the vehicle through the IVehicle interface. It is not guaranteed that an instantiated Vehicle object refers to a valid vehicle in VISSIM after the execution of a simulation step (please refer to the note at the end of this section about the dynamic aspect of vehicles). Use the property ID (see below) for a validity check.

2.24.1 Properties of the IVehicle Interface

ID ([out, retval] long *pID)

Returns the vehicle's identifier number. . If the Vehicle object doesn't refer to a valid VISSIM vehicle anymore the returned value is 0.

Parameters

`[out, retval] long *pID :` returned identifier (0 if reference no longer valid)

Example

```
id = vehicle.ID
```

Name ([out, retval] BSTR *pName)

Returns the vehicle's name.

Parameters

`[out, retval] BSTR *pName :` returned name.

Example

```
name = vehicle.Name
```

Name ([in] BSTR Name)

Sets the vehicle's name.

Parameters

`[in] BSTR Name :` new name.

Example

```
vehicle.Name = „KA-LK 240"
```

AttValue ([in] BSTR Attribute, [out, retval] VARIANT *pValue)

Returns a vehicle attribute. Please get the language independent attribute tag from the table at the end of this section.

Parameters

```
[in] BSTR Attribute :            attribute name (see below)
[out, retval] VARIANT *pValue :  returned value of the attribute
```

Example

```
speed = vehicle.AttValue(„SPEED")
```

AttValue ([in] BSTR Attribute, [in] VARIANT Value)

Sets a vehicle attribute. Please get the language independent attribute tag from the table at the end of this section.

Parameters

```
[in] BSTR Attribute :    attribute name (see below)
[in] BSTR Value :        attribute value (type according to attribute)
```

Example

```
vehicle.AttValue(„PATH")= 1
```

Attribute outline

R	W	Attribute	Description
✓		ID	Identifier number
✓	✓	NAME	Name
✓	✓	BOARDEDPASS	Number of boarding passengers
✓	✓	COLOR	Color in RGB format
✓	✓	ALIGHTEDPASS	Number of alighting passengers
✓		DESLANECHANGE	Desired lane change (1=left, 0=none, -1=right)
✓	✓	DESIREDSPEED	Desired speed in the units of the current options
✓	✓	DESSPEEDFRACTIL	Random selected fractil (0,1) for desired speed selection
✓		DESTPARKLOT	Destination parking lot number (0 if none)
✓		DESTZONE	Destination zone number (0 if none)
✓		ELAPSEDTIME	Total time in network [s]
✓	✓	INTERACTION	Flag (true/false) if vehicle interacts with neighborhood
✓	✓	LANE	Current lane on which the vehicle is positioned
✓		LANECHANGE	Current lane change (1=left, 0=none, -

R	W	Attribute	Description
			1=right)
✓		LASTLANECHANGE	Time since last lane change started [s]
✓		LASTNODE	Previous node number to be traveled (0 if none)
✓	✓	LENGTH	Real vehicle length (current units set)
✓	✓	LINK	Current link where the vehicle is positioned
✓	✓	LINKCOORD	Current x coordinate of the current link
✓		NEXTNODE	Next node number to be traveled (0 if none)
✓		ORIGPARKLOT	Origin parking lot number (0 if none)
✓		ORIGZONE	Origin zone number (0 if none)
✓	✓	PARKLOT	Parking lot number where the vehicle is parked (0 if none)
✓	✓	PASSENGERS	Number of passengers in the vehicles
✓	✓	PATH	Current used path number (0 if none)
✓		POINT	World coordinates of vehicle's position (see note below)
✓		PRECEDING	Number of the next (not necessarily relevant) vehicle downstream
✓		QUEUECOUNTER	Number of queue encounters
✓	✓	SPEED	Current speed in the units of the current options
✓		TOTALDISTANCE	Total distance traveled in the network (current units set)
✓		TRAILING	Number of the next (not necessarily relevant) vehicle upstream
✓	✓	TYPE	Vehicle type number
✓	✓	3DMODELSTATE	3D model state (vehicles with pedestrian categorie are excluded)
✓	✓	WEIGHT	Vehicle total weight [mt]

See page 44 for a note on the RGB color format

The coordinates of a world point are available through the interface IWorldPoint described in page 197

2.24.2 Methods of the IVehicle Interface

MoveToLinkCoordinate ([in] long Link, [in] int Lane, [in] double XCoord)

This method places a vehicle on the lane Lane at the link coordinate XCoord of link Link. It is only applicable to vehicles traveling on the network. If the vehicle is parked the method will not have any effect.

Parameters

```
[in] long Link :        link identifier number
[in] int Lane :         lane number (from 1 to n according to the link)
[in] double XCoord :    link coordinate (from 0.0 to link length)
```

Example

```
DIM vehicle AS Vehicle
SET vehicle = vehicles.GetVehicleByNumber (12)
vehicle.MoveToLinkCoordinate (1000, 1, 20.0)
```

2.24.3 Useful to Know

Attributes

LINK: writing on this attribute has the same effect as the method MoveToLinkCoordinate applied to the assigned link number and to the current lane and coordinate.

PATH: it is possible to use this attribute for re-routing overwriting the current assigned path with a new overlapping path and when the vehicle is driving on the overlapping section.

LINKCOORD: writing on this attribute has the same effect as the method MoveToLinkCoordinate applied to the current link and lane but changing the coordinate to the assigned value.

PARKLOT: writing on this attribute moves a parked vehicle to the assigned parking lot number. If the vehicle is not parked the assignment will no have any effect.

INTERACTION: turning the interaction off will let the vehicle traveling at constant speed without paying attention to vehicles ahead.

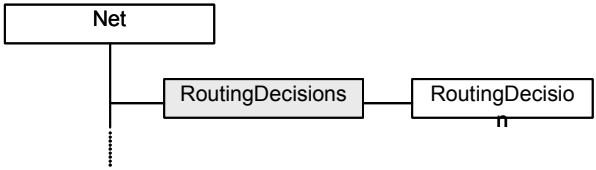
Vehicle is a dynamic network element

Vehicles are dynamic network elements (i.e. the are created, removed and changed during a simulation run). In a client programming context this means that a vehicle variable refers to the current network status. In the following example the speed variable may not be the same value after the simulation step (i.e. speed_before and speed_after could differ):

```
DIM veh AS Vehicle
SET veh = vehicles.GetVehicleByNumber(1)
speed_before = veh.AttValue("SPEED")
sim.RunSimulationStep
speed_after = veh.AttValue("SPEED")
```

2.25 RoutingDecisions

The RoutingDecisions object is a collection of RoutingDecision objects (see page 102).



It belongs to the Net object and can be accessed through the RoutingDecisions property of the INet interface. It contains all routing decisions of the loaded network and allows the iteration through all of them as well as individual access (see also RoutingDecision object).

Example

Instantiation of a RoutingDecisions object and access to all its RoutingDecision objects:

```
DIM vissim As Vissim
DIM decisions As RoutingDecisions
DIM decision As RoutingDecision
SET vissim = NEW Vissim
vissim.LoadNet "c:\vissim\daten\example.inp"
SET decisions = Vissim.Net.RoutingDecisions
FOR EACH decision IN decisions  'access to _NewEnum to create an enumeration
  …
NEXT decision

'or also:

FOR i = 1 TO decisions.Count
SET decision = decisions(i)  'or decisions.Item(i)
…
NEXT i
```

2.25.1 Properties of the IRoutingDecisions Interface

_NewEnum ([out, retval] LPUNKOWN *ppEnum)

This property creates a collection (or enumeration) of all RoutingDecision objects. Once a collection is created, individual members can be returned using the GetRoutingDecisionByNumber method, while the entire collection can be iterated using a FOR ...TO ... NEXT statement. In Visual Basic you can use the statement FOR EACH…NEXT without using the property _NewEnum, which VB calls internally. (see RoutingDecisions example above).

Parameters

```
[out, retval] LPUNKNOWN *ppEnum : returned Enumeration
```

Item ([in] VARIANT Index, [out, retval] IRoutingDecision **ppDecision)

Returns a single RoutingDecision of the collection selected by position. This is useful only when all routing decisions shall be accessed (see the RoutingDecisions example above). To select a routing decision by its identifier number use GetRoutingDecisionByNumber (see below). Since Item is the default property for a collection the following commands are equivalent:

```
SET decision = decisions(1)
SET decision = decisions.Item(1)
```

Parameters

[in] long Index :	index between 1 and Count
[out, retval] IRoutingDecision **ppDecision:	returned RoutingDecision object

Count ([out, retval] long *pCount)

Returns the number of RoutingDecision objects in the collection. See the RoutingDecisions example above.

Parameters

[out, retval] long *pCount :	returned number of objects.

2.25.2 Methods of the IRoutingDecisions Interface

GetRoutingDecisionByNumber ([in] long Number, [out, retval] IRoutingDecision **ppDecision)

Returns the RoutingDecision object with the identifier number Number.

Parameters

[in] long Number :	identifier number
[out, retval] IRoutingDecision **ppDecision :	returned RoutingDecision object

Example

```
SET decision = decisions.GetRoutingDecisionByNumber (2)
IF NOT (decision IS NOTHING) THEN
name = decision.AttValue("NAME")
END IF
```

AddStaticRoutingDecision ([in] long Link, [in] double Xcoord, [out, retval] long *pNumber)

Adds a new static routing decision at the specified link coordinate and with the default time interval 0-99999. If successful, it returns the assigned number, otherwise 0.

Parameters

[in] long Link :	Link identifier number
[in] double XCoord :	Link coordinate
[out, retval] long *pNumber :	returned routing decision identifier number

Example

```
id = decisions.AddStaticRoutingDecision(1, 100.0)
SET decision = decisions.GetRoutingDecisionByNumber (id)
IF NOT (decision IS NOTHING) THEN
decision.AddRoute(2, 200.0)
END IF
```

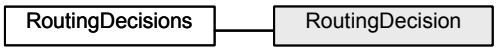
RemoveRoutingDecision ([in] long Number)

Removes the routing decision with the specified identifier number.

Parameters

```
[in] long Number :       Identifier number
```

2.26 RoutingDecision



A RoutingDecision object represents a routing decision element and belongs to the RoutingDecisions object.

It can be accessed through the RoutingDecisions object to in two ways:

- access via iteration through the collection

```
DIM decision As RoutingDecision
FOR EACH decision IN vissim.Net.RoutingDecisions
List.AddItem decision.ID
NEXT decision
```

- individual access via identifier

```
DIM decision As RoutingDecision
SET decision = vissim.Net.RoutingDecisions.GetRoutingDecisionByNumber (12)
```

The RoutingDecision object enables access to the properties of the routing decision through the IRoutingDecision interface.

2.26.1 Properties of the Interface IRoutingDecision

ID ([out, retval] long *pID)

Returns the routing decision identifier number.

Parameters

```
[out, retval] long *pID :returned identifier.
```

Example

```
id = decision.ID
```

Name ([out, retval] BSTR *pName)

Returns the routing decision name.

Parameters

```
[out, retval] BSTR *pName :      returned name.
```

Example

```
name = decision.Name
```

Name ([in] BSTR Name)

Sets the decision's name. Maximal 255 charatcters.

Parameters

```
[in] BSTR Name : new name.
```

Example

```
decision.Name = „XXX"
```

AttValue ([in] BSTR Attribute, [out, retval] VARIANT *pValue)

Returns a routing decision attribute. Please get the language independent attribute tag from the table at the end of this section.

Parameters

```
[in] BSTR Attribute :            attribute name (see table below)
[out, retval] VARIANT *pValue :  returned value of the attribute
```

Example

```
name = decision.AttValue(„NAME“)
```

AttValue ([in] BSTR Attribute, [in] VARIANT Value)

Sets a routing decision attribute. Please get the language independent attribute tag from the table at the end of this section.

Parameters

```
[in] BSTR Attribute :    attribute name (see below)
[in] BSTR Value :        attribute value. (type according to attribute)
```

Example

```
decision.AttValue(„NAME“)=„XXX“
```

Attribute outline

R	W	Attribute	Description
✓		ID	Identifier number
✓	✓	NAME	Name
✓		VEHICLECLASSES	Affected vehicle classes (Array of numbers)
✓		VEHICLETYPES	Affected vehicle types (Array of numbers)

AttValue1 ([in] BSTR Attribute, [in] VARIANT Parameter, [out, retval] VARIANT *pValue)

Returns a routing decision attribute with one parameter. Please get the language independent attribute tag from the table below.

Parameters

```
[in] BSTR Attribute :            attribute name (see below)
[in] VARIANT Parameter :         attribute dependent parameter (see below)
[out, retval] VARIANT *pValue :  returned value of the attribute
```

AttValue1 ([in] BSTR Attribute, [in] VARIANT Parameter, [in] VARIANT Value)

Sets a routing decision attribute with one parameter. Please get the language independent attribute tag from the table below.

Parameters

```
[in] BSTR Attribute :    Attribute name (see below)
[in] VARIANT Parameter : attribute dependent parameter (see below)
[in] VARIANT Value :     Attribute value. (type according to attribute)
```

Attribute outline

R	W	Attribute	Description
✓	✓	RELATIVEFLOW	Relative flow for the curring time intervall (the first one if no simulation is running). Parameter: routing number.
✓	✓	TIMEFROM	Time interval start [s]. Parameter: interval time order (1..n)
✓	✓	TIMEUNTIL	Time interval end [s]. Parameter: interval time order (1..n)
✓	✓	VEHICLECLASS	TRUE if the vehicle class is affected. Parameter: vehicle class number (0 for all vehicle types)
		VEHICLETYPE	TRUE if the vehicle class is affected. Parameter: vehicle type number.

Changes will affect immediately if called during a simulation.

It is required that TIMEUNTIL is greater or equal than TIMEFROM when setting the interval. If the current interval is [100, 1000] and you want to set it to [1100, 2000] you are forced to set first TIMEUNTIL to 2000 and afterwards TIMEFROM to 1100, otherwise you will get an error message.

AttValue2 ([in] BSTR Attribute, [in] VARIANT Parameter1, [in] VARIANT Parameter2, [out, retval] VARIANT *pValue)

Returns a routing decision attribute with two parameters. Please get the language independent attribute tag from the table below.

Parameters

```
[in] BSTR Attribute :            attribute name (see table below)
[in] VARIANT Parameter1 :        first attribute dependent parameter
[in] VARIANT Parameter2 :        second attribute dependent parameter
[out, retval] VARIANT *pValue :  returned value of the attribute
```

AttValue2 ([in] BSTR Attribute, [in] VARIANT Parameter1, [in] VARIANT Parameter2, [in] VARIANT Value)

Sets a routing decision attribute with two parameters. Please get the language independent attribute tag from the table below.

Parameters

```
[in] BSTR Attribute :            Attribute name (see table below)
[in] VARIANT Parameter1 :        first attribute dependent parameter
[in] VARIANT Parameter2 :        second attribute dependent parameter
 [in] VARIANT Value :            Attribute value. (type according to attribute)
```

Attribute outline

R	W	Attribute	Description
✓	✓	RELATIVEFLOW	Relative flow for time intervall and routing number. Parameter1: routing number. Parameter2: a time point of the requested time interval.

Changes will affect immediately if called during a simulation.

Routes ([out, retval] IRoutes **ppRoutes)

Creates an instance of a Routes object (see page 106), that gives individual access to the routes defined in the network for this routing decision.

Parameters

```
[out, retval] IRoutes **ppRoutes : returned Routes object
```

Example

```
DIM routes AS Routes
DIM decision As RoutingDecision
SET decision = vissim.Net.RoutingDecisions.GetRoutingDecisionByNumber (10)
SET routes = decision.Routes
```

2.26.2 Methods of the IRoutingDecision Interface

AddTimeInterval ([in] double From, [in] double To, [out, retval] long *pIndex)

Adds a new time interval. If successful, it returns the assigned list postion index (1..n), otherwise 0.

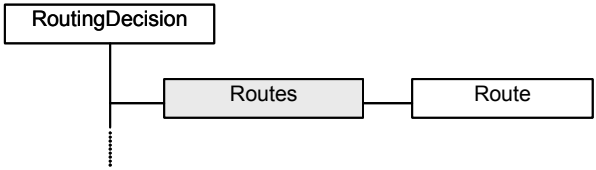
Parameters

```
[in] double From :              Time interval start
[in] double To :                Time interval end
[out, retval] long *pIndex :    returned list position index
```

Example

```
id = decisions.AddStaticRoutingDecision(1, 100.0)
SET decision = decisions.GetRoutingDecisionByNumber (id)
IF NOT (decision IS NOTHING) THEN
decision.AttValue1("TIMEUNTIL", 1) = 1799 'sets default time interval to [0-1799]
index = decision.AddTimeInterval(1800, 3600)
END IF
```

2.27 Routes



The Routes object is a collection of Route objects (see page 109). It belongs to the RoutingDecision object and can be accessed through the Routes property of the IRoutingDecision interface.

It contains all routes of the routing decsion and allows the iteration through all of them as well as individual access (see also Route object).

Example

Instantiation of a Routes object and access to all its Route objects:

```
DIM vissim As Vissim
DIM routes As Routes
DIM route As Route
SET vissim = NEW Vissim
vissim.LoadNet "c:\vissim\daten\example.inp"
SET routes = Vissim.Net.RouteDecisions(1).Routes
FOR EACH route INroutes  'access to _NewEnum to create an enumeration
  …
NEXT route

'or also:

FOR i = 1 TO routes.Count
SET route = routes (i)  'or routes.Item(i)
…
NEXT i
```

2.27.1 Properties of the IRoutes Interface

_NewEnum ([out, retval] LPUNKOWN *ppEnum)

This property creates a collection (or enumeration) of all Route objects. Once a collection is created, individual members can be returned using the GetRouteByNumber method, while the entire collection can be iterated using a FOR ...TO ... NEXT statement. In Visual Basic you can use the statement FOR EACH…NEXT without using the property _NewEnum, which VB calls internally (see Routes example above).

Parameters

`[out, retval] LPUNKNOWN *ppEnum : returned Enumeration`

Item ([in] VARIANT Index, [out, retval] IRoute **ppRoute)

Returns a single Route of the collection selected by position. This is useful only when all routes shall be accessed (see the Routes example above). To select a route by its identifier number use GetRouteByNumber (see below). Since Item is the default property for a collection the following commands are equivalent:

```
SET route = routes(1)
SET route = routes.Item(1)
```

Parameters

```
[in] long Index :                 index between 1 and Count
[out, retval] IRoute **ppRoutes: returned Route object
```

Count ([out, retval] long *pCount)

Returns the number of Route objects in the collection. See the Routes example above.

Parameters

```
[out, retval] long *pCount :      returned number of objects.
```

2.27.2 Methods of the IRoutes Interface

GetRouteByNumber ([in] long Number, [out, retval] IRoute **ppRoute)

Returns the Route object with the identifier number Number.

Parameters

```
[in] long Number :                identifier number
[out, retval] IRoute **ppRoute : returned Route object
```

Example

```
SET route = routes.GetRouteByNumber(2)
IF NOT (route IS NOTHING) THEN
rf = route.AttValue("RELATIVEFLOW")
END IF
```

AddRoute ([in] long Link, [in] double Xcoord, [out, retval] long *pNumber)

Adds a new route with the specified specified link coordinate destination and with a default relative flow 1.0 for each time interval. If successful, it returns the assigned number, otherwise 0.

Parameters

```
[in] long Link :                  Destination link identifier number
[in] double XCoord :              Destination link coordinate
[out, retval] long *pNumber :     returned route identifier number
```

Example

```
id = decisions.AddStaticRoutingDecision(1, 100.0)
SET decision = decisions.GetRoutingDecisionByNumber (id)
IF NOT (decision IS NOTHING) THEN
decision.AddRoute(2, 200.0)
END IF
```

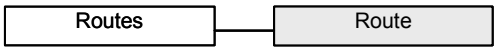
RemoveRoute ([in] long Number)

Removes the route with the specified identifier number.

Parameters

```
[in] long Number :       Identifier number
```

2.28 Route



A Route object represents a route element of a routing decision and belongs to the Routes object. It can be accessed through the Routes object to in two ways:

- access via iteration through the collection

```
DIM routes As Routes
DIM route As Route
SET routes = Vissim.Net.RouteDecisions(1).Routes
FOR EACH route IN routes
List.AddItem route.ID
NEXT route
```

- individual access via identifier

```
DIM route As Route
SET route = vissim.Net.RouteDecisions(1).Routes.GetRouteByNumber (12)
```

The Route object enables access to the properties of the route through the IRoute interface.

2.28.1 Properties of the Interface IRoute

ID ([out, retval] long *pID)

Returns the route identifier number.

Parameters

```
[out, retval] long *pID :returned identifier.
```

Example

```
id = route.ID
```

Name ([out, retval] BSTR *pName)

Returns the route name.

Parameters

```
[out, retval] BSTR *pName :      returned name.
```

Example

```
name = route.Name
```

Name ([in] BSTR Name)

Sets the route's name.

Parameters

```
[in] BSTR Name : new name.
```

Example

```
route.Name = „XXX"
```

AttValue ([in] BSTR Attribute, [out, retval] VARIANT *pValue)

Returns a route attribute. Please get the language independent attribute tag from the table at the end of this section.

Parameters

```
[in] BSTR Attribute :            attribute name (see table below)
[out, retval] VARIANT *pValue :  returned value of the attribute
```

Example

```
name = route.AttValue(„NAME")
```

AttValue ([in] BSTR Attribute, [in] VARIANT Value)

Sets a route attribute. Please get the language independent attribute tag from the table at the end of this section.

Parameters

```
[in] BSTR Attribute :    attribute name (see below)
[in] BSTR Value :        attribute value. (type according to attribute)
```

Example

```
route.AttValue(„NAME")=„XXX"
```

Attribute outline

R	W	Attribute	Description
✓		ID	Identifier number
	✓	NAME	Name

AttValue1 ([in] BSTR Attribute, [in] VARIANT Parameter, [out, retval] VARIANT *pValue)

Returns a route attribute with one parameter. Please get the language independent attribute tag from the table below.

Parameters

```
[in] BSTR Attribute :            attribute name (see below)
[in] VARIANT Parameter :         attribute dependent parameter (see below)
[out, retval] VARIANT *pValue :  returned value of the attribute
```

AttValue1 ([in] BSTR Attribute, [in] VARIANT Parameter, [in] VARIANT Value)

Sets a route attribute with one parameter. Please get the language independent attribute tag from the table below.

Parameters

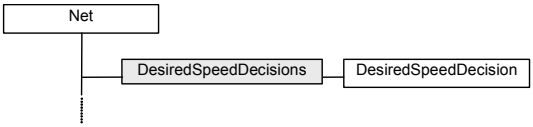
```
[in] BSTR Attribute :    Attribute name (see below)
[in] VARIANT Parameter : attribute dependent parameter (see below)
[in] VARIANT Value :     Attribute value. (type according to attribute)
```

Attribute outline

R	W	Attribute	Description
✓	✓	RELATIVEFLOW	Relative flow of time intervall. Parameter: time interval index (1..n).

Changes will affect immediately if called during a simulation.

2.29 DesiredSpeedDecisions



The Desired Speed Decisions object is a collection of Desired Speed Decision objects (see page 114).

It belongs to the Net object and can be accessed through the DesiredSpeedDecisions property of the INet interface. It contains all desired speed decisions of the loaded network and allows the iteration through all of them as well as individual access (see also DesiredSpeedDecision object).

Example

Instantiation of a DesiredSpeedDecisions object and access to all its DesiredSpeedDecision objects:

```
DIM vissim As Vissim
DIM decisions As DesiredSpeedDecisions
DIM decision As DesiredSpeedDecision
SET vissim = NEW Vissim
vissim.LoadNet "c:\vissim\daten\example.inp"
SET decisions = Vissim.Net.DesiredSpeedDecisions
FOR EACH decision IN decisions  'access to _NewEnum to create an enumeration
  …
NEXT decision

'or also:

FOR i = 1 TO decisions.Count
SET decision = decisions(i)  'or decisions.Item(i)
…
NEXT i
```

2.29.1 Properties of the IDesiredSpeedDecisions Interface

_NewEnum ([out, retval] LPUNKOWN *ppEnum)

This property creates a collection (or enumeration) of all DesiredSpeedDecision objects. Once a collection is created, individual members can be returned using the GetDesiredSpeedDecisionByNumber method, while the entire collection can be iterated using a FOR ...TO ... NEXT statement. In Visual Basic you can use the statement FOR EACH…NEXT without using the property _NewEnum, which VB calls internally. (see DesiredSpeedDecisions example above).

Parameters

[out, retval] LPUNKNOWN *ppEnum : returned Enumeration

Item ([in] VARIANT Index, [out, retval] IDesiredSpeedDecision **ppDecision)

Returns a single DesiredSpeedDecision of the collection selected by position. This is useful only when all desired speed decisions shall be accessed (see the DesiredSpeedDecisions example above). To select a desired speed decision by its identifier number use GetDesiredSpeedDecisionByNumber (see below). Since Item is the default property for a collection the following commands are equivalent:

```
SET decision = decisions(1)
SET decision = decisions.Item(1)
```

Parameters

```
[in] long Index:                                  index between 1 and Count
[out, retval] IDesiredSpeedDecision **ppDecision: returned DesiredSpeedDecision
```

Count ([out, retval] long *pCount)

Returns the number of DesiredSpeedDecision objects in the collection. See the DesiredSpeedDecisions example above.

Parameters

```
[out, retval] long *pCount :     returned number of objects.
```

2.29.2 Methods of the IDesiredSpeedDecisions Interface

GetDesiredSpeedDecisionByNumber ([in] long Number, [out, retval] IDesiredSpeedDecision **ppDecision)

Returns the DesiredSpeedDecision object with the identifier number Number.

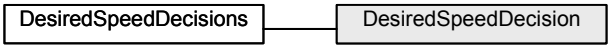
Parameters

```
[in] long Number :                                identifier number
[out, retval] IDesiredSpeedDecision **ppDecision :returned DesiredSpeedDecision
```

Example

```
SET decision = decisions.GetDesiredSpeedDecisionByNumber (2)
IF NOT (decision IS NOTHING) THEN
name = decision.AttValue("NAME")
END IF
```

2.30 DesiredSpeedDecision



A DesiredSpeedDecision object represents a desired speed decision element and belongs to the DesiredSpeedDecisions object.

It can be accessed through the DesiredSpeedDecisions object to in two ways:

- access via iteration through the collection

```
DIM decision As DesiredSpeedDecision
FOR EACH decision IN vissim.Net.DesiredSpeedDecisions
List.AddItem decision.ID
NEXT decision
```

- individual access via identifier

```
DIM decision As DesiredSpeedDecision
SET decision = vissim.Net.DesiredSpeedDecisions.
GetDesiredSpeedDecisionByNumber (12)
```

The DesiredSpeedDecision object enables access to the properties of the desired speed decision through the IDesiredSpeedDecision interface.

2.30.1 Properties of the Interface IDesiredSpeedDecision

ID ([out, retval] long *pID)

Returns the desired speed decision identifier number.

Parameters

```
[out, retval] long *pID :returned identifier.
```

Example

```
id = decision.ID
```

Name ([out, retval] BSTR *pName)

Returns the name of the desired speed decision.

Parameters

```
[out, retval] BSTR *pName :      returned name.
```

Example

```
name = decision.Name
```

Name ([in] BSTR Name)

Sets the decision's name. Maximal 255 charatcters.

Parameters

[in] BSTR Name : new name.

Example

decision.Name = „XXX"

AttValue ([in] BSTR Attribute, [out, retval] VARIANT *pValue)

Returns a desired speed decision attribute. Please get the language independent attribute tag from the table at the end of this section.

Parameters

[in] BSTR Attribute : attribute name (see table below)
[out, retval] VARIANT *pValue : returned value of the attribute

Example

name = decision.AttValue(„NAME")

AttValue ([in] BSTR Attribute, [in] VARIANT Value)

Sets a desired speed decision attribute. Please get the language independent attribute tag from the table at the end of this section.

Parameters

[in] BSTR Attribute : attribute name (see below)
[in] BSTR Value : attribute value. (type according to attribute)

Example

decision.AttValue(„NAME")=„XXX"

Attribute outline

R	W	Attribute	Description
✓		ID	Identifier number
✓	✓	NAME	Name
✓	✓	TIMEFROM	Time interval start of activation[s].
✓	✓	TIMETO	Time interval end of activation[s].
✓	✓	DESIREDSPEED	Desired speed distribution number of the first vehicle class.
✓		VEHICLECLASSES	Affected vehicle classes (Array of numbers)
✓		VEHICLETYPES	Affected vehicle types (Array of numbers)

Note: It is required that TIMEUNTIL is greater or equal than TIMEFROM when setting the interval. If the current interval is [100, 1000] and you want to set it to [1100, 2000] you are forced to set first TIMEUNTIL to 2000 and afterwards TIMEFROM to 1100, otherwise you will get an error message.

AttValue1 ([in] BSTR Attribute, [in] VARIANT Parameter, [out, retval] VARIANT *pValue)

Returns a desired speed decision attribute with one parameter. Please get the language independent attribute tag from the table below.

Parameters

```
[in] BSTR Attribute :            attribute name (see below)
[in] VARIANT Parameter :         attribute dependent parameter (see below)
[out, retval] VARIANT *pValue :  returned value of the attribute
```

AttValue1 ([in] BSTR Attribute, [in] VARIANT Parameter, [in] VARIANT Value)

Sets a desired speed decision attribute with one parameter. Please get the language independent attribute tag from the table below.

Parameters

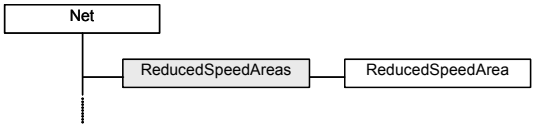
```
[in] BSTR Attribute :    Attribute name (see below)
[in] VARIANT Parameter : attribute dependent parameter (see below)
[in] VARIANT Value :     Attribute value. (type according to attribute)
```

Attribute outline

R	W	Attribute	Description
✓	✓	DESIREDSPEED	Desired speed distribution number. Parameter: vehicle class.
✓	✓	VEHICLECLASS	TRUE if the vehicle class is affected. Parameter: vehicle class number.
✓		VEHICLETYPE	TRUE if the vehicle class is affected. Parameter: vehicle type number.

Changes will affect immediately if called during a simulation.

2.31 ReducedSpeedAreas



The ReducedSpeedAreas object is a collection of ReducedSpeedArea objects (see page 119). It belongs to the Net object and can be accessed through the ReducedSpeedAreas property of the INet interface.

It contains all reduced speed areas of the loaded network and allows the iteration through all of them as well as individual access (see also ReducedSpeedArea object).

Example

Instantiation of a ReducedSpeedAreas object and access to all its ReducedSpeedArea objects:

```
DIM vissim As Vissim
DIM speedareas As ReducedSpeedAreas
DIM speedarea As ReducedSpeedArea
SET vissim = NEW Vissim
vissim.LoadNet "c:\vissim\daten\example.inp"
SET speedareas = Vissim.Net.ReducedSpeedAreas
FOR EACH speedarea IN speedareas  'access to create an enumeration
  …
NEXT speedarea

'or also:

FOR i = 1 TO speedareas.Count
SET speedarea = speedareas(i)  'or speedareas.Item(i)
…
NEXT i
```

2.31.1 Properties of the IReducedSpeedAreas Interface

_NewEnum ([out, retval] LPUNKOWN *ppEnum)

This property creates a collection (or enumeration) of all ReducedSpeedArea objects. Once a collection is created, individual members can be returned using the GetReducedSpeedAreaByNumber method, while the entire collection can be iterated using a FOR ...TO ... NEXT statement. In Visual Basic you can use the statement FOR EACH…NEXT without using the property _NewEnum, which VB calls internally. (see ReducedSpeedAreas example above).

Parameters

```
[out, retval] LPUNKNOWN *ppEnum : returned Enumeration
```

Item ([in] VARIANT Index, [out, retval] IReducedSpeedArea **ppSpeedArea)

Returns a single ReducedSpeedArea of the collection selected by position. This is useful only when all reduced speed areas shall be accessed (see the ReducedSpeedAreas example above). To select a reduced speed area by its identifier number use GetReducedSpeedAreaByNumber (see below). Since Item is the default property for a collection the following commands are equivalent:

```
SET speedarea = speedareas(1)
SET speedarea = speedareas.Item(1)
```

Parameters

```
[in] long Index:                                index between 1 and Count
[out, retval] IReducedSpeedArea **ppSpeedArea:  returned ReducedSpeedArea
```

Count ([out, retval] long *pCount)

Returns the number of ReducedSpeedArea objects in the collection. See the ReducedSpeedAreas example above.

Parameters

```
[out, retval] long *pCount :     returned number of objects.
```

2.31.2 Methods of the IReducedSpeedAreas Interface

GetReducedSpeedAreaByNumber ([in] long Number, [out, retval] IReducedSpeedArea ** ppSpeedArea)

Returns the ReducedSpeedArea object with the identifier number Number.

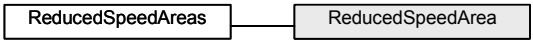
Parameters

```
[in] long Number :                              identifier number
[out, retval] IReducedSpeedArea ** ppSpeedArea: returned ReducedSpeedArea
```

Example

```
SET speedarea = speedareas.GetReducedSpeedAreaByNumber (2)
IF NOT (speedarea IS NOTHING) THEN
name = speedarea.AttValue("NAME")
END IF
```

2.32 ReducedSpeedArea



A ReducedSpeedArea object represents a reduced speed area element and belongs to the ReducedSpeedAreas object. It can be accessed through the ReducedSpeedAreas object to in two ways:

- access via iteration through the collection

```
DIM speedarea As ReducedSpeedArea
FOR EACH speedarea IN vissim.Net.ReducedSpeedAreas
List.AddItem speedarea.ID
NEXT speedarea
```

- individual access via identifier

```
DIM speedarea As ReducedSpeedArea
SET speedarea = vissim.Net.ReducedSpeedAreas.
GetReducedSpeedAreaByNumber (12)
```

The ReducedSpeedArea object enables access to the properties of the reduced speed area through the IReducedSpeedArea interface.

2.32.1 Properties of the Interface IReducedSpeedArea

ID ([out, retval] long *pID)

Returns the reduced speed area identifier number.

Parameters

```
[out, retval] long *pID :returned identifier.
```

Example

```
id = speedarea.ID
```

Name ([out, retval] BSTR *pName)

Returns the name of the reduced speed area.

Parameters

```
[out, retval] BSTR *pName :      returned name.
```

Example

```
name = speedarea.Name
```

Name ([in] BSTR Name)

Sets the name of the reduced speed area. Maximal 255 charatcters.

Parameters

```
[in] BSTR Name : new name.
```

Example

```
speedarea.Name = „XXX"
```

AttValue ([in] BSTR Attribute, [out, retval] VARIANT *pValue)

Returns a reduced speed area attribute. Please get the language independent attribute tag from the table at the end of this section.

Parameters

```
[in] BSTR Attribute :            attribute name (see table below)
[out, retval] VARIANT *pValue :  returned value of the attribute
```

Example

```
name = speedarea.AttValue(„NAME")
```

AttValue ([in] BSTR Attribute, [in] VARIANT Value)

Sets a reduced speed area attribute. Please get the language independent attribute tag from the table at the end of this section.

Parameters

```
[in] BSTR Attribute :    attribute name (see below)
[in] BSTR Value :        attribute value. (type according to attribute)
```

Example

```
speedarea.AttValue(„NAME")=„XXX"
```

Attribute outline

R	W	Attribute	Description
✓		ID	Identifier number
✓	✓	NAME	Name
✓	✓	TIMEFROM	Time interval start of activation[s].
✓	✓	TIMETO	Time interval end of activation[s].
✓	✓	DESIREDSPEED	Desired speed distribution number of the first vehicle class.
✓		VEHICLECLASSES	Affected vehicle classes (Array of numbers)
✓		VEHICLETYPES	Affected vehicle types (Array of numbers)

i It is required that TIMEUNTIL is greater or equal than TIMEFROM when setting the interval. If the current interval is [100, 1000] and you want to set it to [1100, 2000] you are forced to set first TIMEUNTIL to 2000 and afterwards TIMEFROM to 1100, otherwise you will get an error message.

AttValue1 ([in] BSTR Attribute, [in] VARIANT Parameter, [out, retval] VARIANT *pValue)

Returns a reduced speed area attribute with one parameter. Please get the language independent attribute tag from the table below.

Parameters

```
[in] BSTR Attribute :            attribute name (see below)
[in] VARIANT Parameter :         attribute dependent parameter (see below)
[out, retval] VARIANT *pValue :  returned value of the attribute
```

AttValue1 ([in] BSTR Attribute, [in] VARIANT Parameter, [in] VARIANT Value)

Sets a reduced speed area attribute with one parameter. Please get the language independent attribute tag from the table below.

Parameters

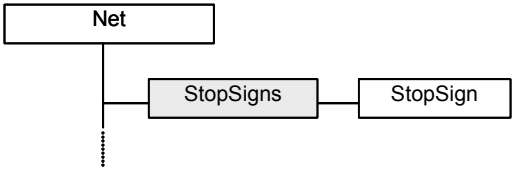
```
[in] BSTR Attribute :    Attribute name (see below)
[in] VARIANT Parameter : attribute dependent parameter (see below)
[in] VARIANT Value :     Attribute value. (type according to attribute)
```

Attribute outline

R	W	Attribute	Description
✓	✓	DESIREDSPEED	Desired speed distribution number. Parameter: vehicle class.
✓	✓	VEHICLECLASS	TRUE if the vehicle class is affected. Parameter: vehicle class number.
✓		VEHICLETYPE	TRUE if the vehicle class is affected. Parameter: vehicle type number.

Changes will affect immediately if called during a simulation.

2.33 StopSigns



The StopSigns object is a collection of StopSign objects (see page 124). It belongs to the Net object and can be accessed through the StopSigns property of the INet interface.

It contains all stop signs (not transit stops) of the loaded network and allows the iteration through all of them as well as individual access (see also StopSign object).

Example

Instantiation of a StopSigns object and access to all its StopSign objects:

```
DIM vissim As Vissim
DIM stops As StopSigns
DIM stop As StopSign
SET vissim = NEW Vissim
vissim.LoadNet "c:\vissim\daten\example.inp"
Set stops = Vissim.Net.StopSigns
FOR EACH stop IN stops 'access to _NewEnum to create an enumeration
  …
NEXT stop

'or also:

FOR i = 1 TO stops.Count
SET stop = stops(i)   'or stops.Item(i)
…
NEXT i
```

2.33.1 Properties of the IStopSigns Interface

_NewEnum ([out, retval] LPUNKOWN *ppEnum)

This property creates a collection (or enumeration) of all StopSign objects. Once a collection is created, individual members can be returned using the GetStopSignByNumber method, while the entire collection can be iterated using a FOR ...TO ... NEXT statement. In Visual Basic you can use the statement FOR EACH…NEXT without using the property _NewEnum, which VB calls internally (see the StopSigns example above).

Parameters

```
[out, retval] LPUNKNOWN *ppEnum : returned Enumeration
```

Item ([in] VARIANT Index, [out, retval] IStopSign **ppStopSign)

Returns a single StopSign of the collection selected by position. This is useful only when all stop signs shall be accessed (see the StopSigns example above). To select a stop sign by its identifier number use GetStopSignByNumber (see below). Since Item is the default property for a collection the following commands are equivalent:

```
SET stop = stops(1)
SET stop = stops.Item(1)
```

Parameters

```
[in] long Index :                        index between 1 and Count
[out, retval] IStopSign **ppStopSigns:   returned StopSign object
```

Count ([out, retval] long *pCount)

Returns the number of StopSign objects in the collection. See the StopSigns example above.

Parameters

```
[out, retval] long *pCount :     returned number of objects.
```

2.33.2 Methods of the IStopSigns Interface

GetStopSignByNumber ([in] long Number, [out, retval] IStopSign **ppStopSign)

Returns the StopSign object with the identifier number Number.

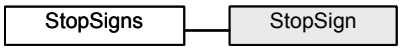
Parameters

```
[in] long Number :                       identifier number
[out, retval] IStopSign **ppStopSign :   returned StopSign object
```

Example

```
SET stop = stops.GetStopSignByNumber (2)
IF NOT (stop IS NOTHING) THEN
name = stop.AttValue("NAME")
END IF
```

2.34 StopSign



A StopSign object represents a stop sign element and belongs to the StopSigns object. It can be accessed through the StopSigns object to in two ways:

- access via iteration through the collection

```
DIM stop As StopSign
FOR EACH stop IN vissim.Net.StopSigns
List.AddItem stop.ID
NEXT stop
```

- individual access via identifier

```
DIM stop As StopSign
SET stop = vissim.Net.StopSigns.GetStopSignByNumber (12)
```

The StopSign object enables access to the properties of the stop sign through the IStopSign interface.

2.34.1 Properties of the Interface IStopSign

ID ([out, retval] long *pID)

Returns the stop sign identifier number.

Parameters

```
[out, retval] long *pID :returned identifier.
```

Example

```
id = stop.ID
```

Name ([out, retval] BSTR *pName)

Returns the name of the stop sign.

Parameters

```
[out, retval] BSTR *pName :      returned name.
```

Example

```
name = stop.Name
```

Name ([in] BSTR Name)

Sets the name of the stop sign. Maximal 255 charatcters.

Parameters

```
[in] BSTR Name : new name.
```

Example

```
stop.Name = „XXX"
```

AttValue ([in] BSTR Attribute, [out, retval] VARIANT *pValue)

Returns a stop sign attribute. Please get the language independent attribute tag from the table below.

Parameters

```
[in] BSTR Attribute :            attribute name (see below)
[out, retval] VARIANT *pValue :  returned value of the attribute
```

Example

```
name = stop.AttValue(„NAME“)
```

AttValue ([in] BSTR Attribute, [in] VARIANT Value)

Sets a named attribute. Please get the language independent attribute tag from the table below.

Parameters

```
[in] BSTR Attribute :    Attribute name (see below)
[in] VARIANT Value :     Attribute value. (type according to attribute)
```

Example

```
stop.AttValue(„NAME“) = “cashbox1”
```

Attribute outline

R	W	Attribute	Description
✓		ID	Identifier number
✓	✓	NAME	Name

AttValue1 ([in] BSTR Attribute, [in] VARIANT Parameter, [out, retval] VARIANT *pValue)

Returns a stop sign attribute with one parameter. Please get the language independent attribute tag from the table at the end of this section.

Parameters

```
[in] BSTR Attribute :            Attribute name (see below)
[in] VARIANT Parameter :         Attribute dependent parameter (see below)
[out, retval] VARIANT *pValue :  returned value of the attribute
```

AttValue1 ([in] BSTR Attribute, [in] VARIANT Parameter, [in] VARIANT Value)

Sets a stop sign attribute with one parameter. Please get the language independent attribute tag from the table below.

Parameters

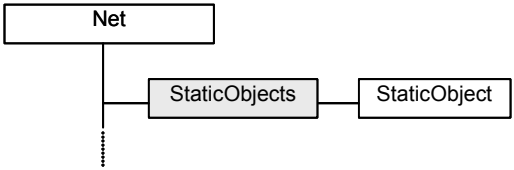
```
[in] BSTR Attribute :    Attribute name (see below)
[in] VARIANT Parameter : Attribute dependent parameter (see below)
[in] VARIANT Value :     Attribute value. (type according to the attribute)
```

Attribute outline

This function is currently not used.

R	W	Attribute	Description
✓	✓	TIMEDIST	Time distribution number for dwell time (Writeable during simulation time). Parameter: vehicle class number. Assign 0 to remove an assigned time distribution to a vehicle class.

2.35 StaticObjects



The StaticObjects object is a collection of StaticObject objects (see page 130). It belongs to the Net object and can be accessed through the StaticObjects property of the INet interface.

It contains all 3D static obejcts of the loaded network and allows the iteration through all of them as well as individual access (see also StaticObject object).

Example

Instantiation of a StaticObjects object and access to all its StaticObject objects:

```
DIM vissim As Vissim
DIM stobjs As StaticObjects
DIM stobj As StaticObject
SET vissim = NEW Vissim
vissim.LoadNet "c:\vissim\daten\example.inp"
SET stobjs = Vissim.Net.StaticObjects
FOR EACH stobj IN stobjs 'access to _NewEnum to create an enumeration
  …
NEXT stobj

'or also:

FOR i = 1 TO stobjs.Count
SET stobj = stobjs(i)  'or stobjs.Item(i)
…
NEXT i
```

2.35.1 Properties of the IStaticObjects Interface

_NewEnum ([out, retval] LPUNKOWN *ppEnum)

This property creates a collection (or enumeration) of all StaticObject objects. Once a collection is created, individual members can be returned using the GetStaticObjectByNumber method, while the entire collection can be iterated using a FOR ...TO ... NEXT statement. In Visual Basic you can use the statement FOR EACH…NEXT without using the property _NewEnum, which VB calls internally (see the StaticObjects example above).

Parameters

`[out, retval] LPUNKNOWN *ppEnum : returned Enumeration`

Item ([in] VARIANT Index, [out, retval] IStaticObject **ppStaticObject)

Returns a single StaticObject of the collection selected by position. This is useful only when all static obejcts shall be accessed (see the StaticObjects example above). To select a static object by its identifier number use GetStaticObjectByNumber (see below). Since Item is the default property for a collection the following commands are equivalent:

```
SET stobj = stobjs(1)
SET stobj = stobjs.Item(1)
```

Parameters

[in] long Index :	index between 1 and Count
[out, retval] IStaticObject **ppStaticObjects:	returned StaticObject object

Count ([out, retval] long *pCount)

Returns the number of StaticObject objects in the collection. See the StaticObjects example above.

Parameters

[out, retval] long *pCount :	returned number of objects.

2.35.2 Methods of the IStaticObjects interface

GetStaticObjectByName ([in] BSTR Name, [in, defaultvalue(0)] IWorldPoint *pWorldPoint, [out, retval] IStaticObject **ppStaticObject)

Returns the StaticObject object with the name Name and world coordinates pWorldPoint. The name of a static object is the file name of its 3D model (*.v3d). If no world coordinates are specified the first static object encountered with the name Name will be returned.

Parameters

[in] BSTR Name :	Name (Name of the *.v3d file)
[in] IWorldPoint *pWorldPoint :	Optional position world coordinates
[out, retval] IStaticObject **ppStaticObject :	returned StaticObject object

Example

```
SET stobj = stobjs.GetStaticObjectByName("Biz-Man_suit.v3d")
IF NOT (stobj IS NOTHING) THEN
state = stobj.AttValue("STATE")
END IF
```

GetStaticObjectByCoord ([in] IWorldPoint *pWorldPoint, [in, defaultvalue("")]BSTR Name, [out, retval] IStaticObject **ppStaticObject)

Returns the StaticObject object within the position with world coordinates pWorldPoint and the name Name. The name of a static object is the file name of its 3D model (*.v3d). If no name is specified the first static object encountered on the position pWorldPoint will be returned.

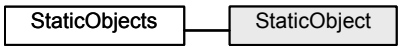
Parameters

[in] IWorldPoint *pWorldPoint :	Position world coordinates
[in] BSTR Name :	Optional name (Name of the *.v3d file)
[out, retval] IStaticObject **ppStaticObject :	returned StaticObject object

Example

```
DIM wp AS WorldPoint
SET wp = vissim.NewWorldPoint(100, 100, 100)
SET stobj = stobjs.GetStaticObjectByCoord(wp)
IF NOT (stobj IS NOTHING) THEN
state = stobj.AttValue("STATE")
END IF
```

2.36 StaticObject



A StaticObject object represents a static object element and belongs to the StaticObjects object. It can be accessed through the StaticObjects object to in two ways:

- access via iteration through the collection

```
DIM stobj As StaticObject
FOR EACH stobj IN vissim.Net.StaticObjects
List.AddItem stobj.ID
NEXT stobj
```

- individual access via identifier

```
DIM stobj As StaticObject
SET stobj = vissim.Net.StaticObjects.GetStaticObjectByName ("Biz-Man_suit.v3d")
```

The StaticObject object enables access to the properties of the static object through the IStaticObject interface.

2.36.1 Properties of the Interface IStaticObject

ID ([out, retval] long *pID)

Returns the static object identifier number. The identifier number is resolved internally and may change when the network is modified.

Parameters

```
[out, retval] long *pID :returned identifier.
```

Example

```
id = stobj.ID
```

Name ([out, retval] BSTR *pName)

Returns the name of the static object. The name of the static object is the name of ist *.v3d file.

Parameters

```
[out, retval] BSTR *pName :      returned name.
```

Example

```
name = stobj.Name
```

AttValue ([in] BSTR Attribute, [out, retval] VARIANT *pValue)

Returns a static object attribute. Please get the language independent attribute tag from the table below.

Parameters

```
[in] BSTR Attribute :            attribute name (see below)
[out, retval] VARIANT *pValue :  returned value of the attribute
```

Example

```
state = stobj.AttValue(„STATE“)
```

AttValue ([in] BSTR Attribute, [in] VARIANT Value)

Sets a named attribute. Please get the language independent attribute tag from the table below.

Parameters

```
[in] BSTR Attribute :    Attribute name (see below)
[in] VARIANT Value :     Attribute value. (type according to attribute)
```

Example

```
stobj.AttValue(„STATE“) = 2
```

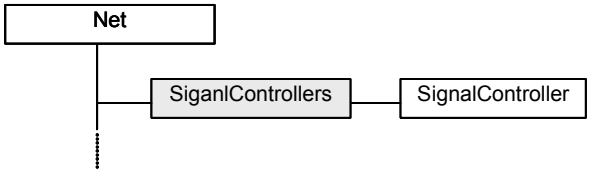
Attribute outline

R	W	Attribute	Description
✓		ID	Identifier number
✓		NAME	Name
✓		NSTATES	Number of possible 3D model states of the object.
✓		POSITION	World coordinates of the object position on the network
✓	✓	STATE	Current object 3D model state.

i Changes will affect immediately if called during a simulation. Example:

```
DIM sim AS Simulation
DIM stobj AS StaticObject
SET sim = vissim.Simulation
SET stobj = vissim.Net. StaticObjects.GetStaticObjectByName (“Biz-Man_suit.v3d”)
n_states = stobj.AttValue(„NSTATES“)
FOR i = 1 TO sim.Period * sim.Resolution
sim.RunSingleStep
stobj.AttValue(„STATE“) = (stobj.AttValue(„STATE“) + 1) MOD n_states
NEXT si
```

2.37 SignalControllers



The SignalControllers object is a collection of Signal objects (see page 134). It belongs to the Net object and can be accessed through the SignalControllers property of the INet interface.

It contains all signal controllers of the loaded network and enables iteration through the collection or individual access to a SignalController object.

Example

Instantiation of a SignalControllers object and access to all its SignalController objects:

```
DIM vissim AS Vissim
DIM controllers AS SignalControllers
DIM controller AS SignalController
SET vissim = NEW Vissim
vissim.LoadNet "c:\vissim\daten\example.inp"
SET controllers = Vissim.Net.SignalControllers
FOR EACH controller IN controllers  'access to _NewEnum to create an enumeration
…
NEXT controller

'or also:

FOR i = 1 TO controllers.Count
SET controller = controllers(i)  'or controllers.Item(i)
…
NEXT i
```

2.37.1 Properties of the ISignalControllers Interface

_NewEnum ([out, retval] LPUNKNOWN *ppEnum)

This property creates a collection (or enumeration) of all SignalController objects. Once a collection is created, individual members can be obtained using the GetSignalControllerByNumber method, while the entire collection can be iterated using a FOR ...TO ... NEXT statement. In Visual Basic you can use the statement FOR EACH…NEXT without using the property _NewEnum, which VB calls internally. (see SignalControllers example above).

Parameters

```
[out, retval] LPUNKNOWN *ppEnum : returned Enumeration
```

Item ([in] VARIANT Index, [out, retval] ISignalController **ppSC)

Returns a single SignalController of the collection selected by position. This is useful only when all signal controllers shall be accessed (see the SignalControllers example above). To select a signal controller by its identifier number use GetSignalControllerByNumber (see below). Since Item is the default property for a collection the following commands are equivalent:

```
SET sc = controllers(1)
SET sc = controllers.Item(1)
```

Parameters

```
[in] long Index :                        index between 1 and controllers.Count
[out, retval] ISignalController **ppSC : returned SignalController object
```

Count ([out, retval] long *pCount)

Returns the number of SignalController objects in the collection. See the SignalControllers example above.

Parameters

```
[out, retval] long *pCount :     returned number of objects.
```

2.37.2 Methods of the ISignalControllers Interface

GetSignalControllerByNumber ([in] long Number,[out, retval] ISignalController **ppSC)

Returns the SignalController object with the identifier number Number.

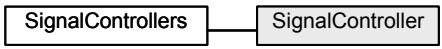
Parameters

```
[in] long Number :                       identifier number
[out, retval] ISignalController *ppSC :  returned SignalController object
```

Example

```
DIM sc AS SignalController
SET sc = controllers.GetSignalControllerByNumber (11)
```

2.38 SignalController



A SignalController object belongs to the network's SignalControllers object. Through this a signal controller can be accessed in two ways:

- access via iteration through the collection

```
DIM controllers As SignalControllers
DIM controller As SignalController
SET controllers = Vissim.Net.SignalControllers
FOR EACH controller IN controllers
List.AddItem controller.ID
NEXT controller
```

- individual access via identifier

```
DIM controllers As SignalControllers
DIM controller As SignalController
SET controllers = Vissim.Net.SignalControllers
SET controller = controllers.GetSignalControllerByNumber(1000)
```

The SignalController object enables access to the properties of the signal controller through the ISignalController interface.

2.38.1 Properties of the ISignalController Interface

ID ([out, retval] long *pID)

Returns the identifier of the signal controller.

Parameters

```
[out, retval] long *pID :returned identifier.
```

Example

```
id = sc.ID
```

Name ([out, retval] BSTR *pName)

Returns the name of the signal controller.

Parameters

```
[out, retval] BSTR *pName :      returned name.
```

Example

```
name = sc.Name
```

Name ([in] BSTR Name)

Sets the name of the signal controller.

Parameters

```
[in] BSTR Name : new name.
```

Example

```
sc.Name = „CanPanegre“
```

AttValue ([in] BSTR Attribute, [out, retval] VARIANT *pValue)

Returns a signal controller attribute. Please get the language independent attribute tag from the table at the end of this section.

Parameters

```
[in] BSTR Attribute :            attribute name (see below)
[out, retval] VARIANT *pValue :  returned value of the attribute
```

Example

```
cycle = sc.AttValue(„CYCLETIME“)
```

AttValue ([in] BSTR Attribute, [in] VARIANT Value)

Sets a signal controller attribute. Please get the language independent attribute tag from the table at the end of this section.

Parameters

```
[in] BSTR Attribute :    Attribute name (see below)
[in] BSTR Value :        Attribute value. (type according to attribute)
```

Example

```
sc.AttValue(„TYPE“) = 1  'fixed time
```

Attribute outline

R	W	Attribute	Description
✓		ID	Identifier number
✓	✓	NAME	Name
✓	✓	CYCLETIME	Cylcle time [s]
✓	✓	LOGFILE	Filename for the record file (*.ldp)
✓	✓	PROGRAM	VAP program number
✓	✓	OFFSET	Offset time [s]
✓	✓	TYPE	Type (1: fixed, 2: SDM, 3: VS_PLUS, 4: TRENDS, 5: VAP, 6: TL, 7: VOS, 8: NEMA, 9: External)

Detectors ([out, retval] IDetectors **ppDetectors)

Creates an instance of a Detectors object (see page 137), that gives individual access to the detector elements of the network.

Parameters

```
[out, retval] IDetectors**ppDetectors :   returned Detectors object
```

Example

```
DIM dets AS Detectors
SET dets = sc.Detectors
```

SignalGroups ([out, retval] ISignalGroups **ppSignalGroups)

Creates an instance of a SignalGroups object (see page 142), that gives individual access to the signal group elements of the network.

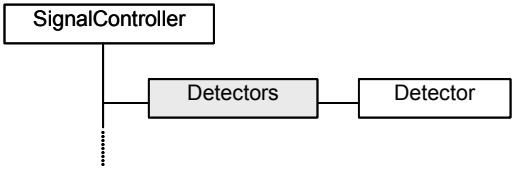
Parameters

```
[out, retval] ISignalGroups **ppSignalGroups :     returned SignalGroups object
```

Example

```
DIM sgs AS SignalGroups
SET sgs = sc.SignalGroups
```

2.39 Detectors



The Detectors object is a collection of Detector objects (see page 139). It belongs to a SignalController object and can be accessed through the Detectors property of the ISignalController interface.

It contains all detectors of the refered signal controller and enables iteration through the collection or individual access to a Detector object.

Example

Instantiation of a Detectors object and access to all its Detector objects:

```
DIM vissim AS Vissim
DIM dets AS Detectors
DIM det AS Detector
SET vissim = NEW Vissim
vissim.LoadNet "c:\vissim\daten\example.inp"
SET dets = Vissim.Net.SignalControllers(1).Detectors
FOR EACH det IN dets  'access to _NewEnum to create an enumeration
…
NEXT det

'or also:

FOR i = 1 TO dets.Count
SET det = dets(i)  'or dets.Item(i)
…
NEXT i
```

2.39.1 Properties of the IDetectors Interface

_NewEnum ([out, retval] LPUNKNOWN *ppEnum)

This property creates a collection (or enumeration) of all Detector objects. Once a collection is created, individual members can be returned using the GetDetectorByNumber method, while the entire collection can be iterated using a FOR ...TO ... NEXT statement. In Visual Basic you can use the statement FOR EACH…NEXT without using the property _NewEnum, which VB calls internally. (see Detectors example above).

Parameters

`[out, retval] LPUNKNOWN *ppEnum : returned Enumeration`

Item ([in] VARIANT Index, [out, retval] IDetector **ppDetector)

Returns a single Detector of the collection selected by position. This is useful only when all detectors shall be accessed (see the Detectors example above). To select a detector by its identifier number use GetDetectorByNumber (see below). Since Item is the default property for a collection the following commands are equivalent:

```
SET det = dets(1)
SET det = dets.Item(1)
```

Parameters

```
[in] long Index :                        index between 1 and dets.Count
[out, retval] IDetector **ppDetector :   returned Detector object
```

Count ([out, retval] long *pCount)

Returns the number of Detector objects in the collection. See the Detectors example above.

Parameters

```
[out, retval] long *pCount :     returned number of objects.
```

2.39.2 Methods of the IDetectors Interface

GetDetectorByNumber ([in] long Number, [out, retval] IDetector **ppDetector)

Returns the Detector object with the identifier number Number.

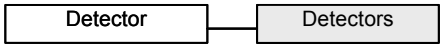
Parameters

```
[in] long Number :                       identifier number
[out, retval] IDetector **ppDetector:    returned Detector object
```

Example

```
DIM det AS Detector
SET det = dets.GetDetectorByNumber(1)
```

2.40 Detector



A Detector object belongs to a controller's Detectors object. Through this a detector can be accessed in two ways:

- access via iteration through the collection

```
DIM dets As Detectors
DIM det As Detector
SET dets = Vissim.Net.SignalControllers(1).Detectors
FOR EACH det IN dets
List.AddItem det.ID
NEXT det
```

- individual access via identifier

```
DIM dets As Detectors
DIM det As Detector
SET dets = Vissim.Net.SignalControllers(1).Detectors
SET det = dets.GetDetectorByNumber(10)
```

The Detector object enables access to the properties of the detector through the IDetector interface.

2.40.1 Properties of the IDetector interface

ID ([out, retval] long *pID)

Returns the identifier of the detector.

Parameters

```
[out, retval] long pID : returned identifier.
```

Example

```
id = det.ID
```

Name ([out, retval] BSTR *pName)

Returns the name of the detector.

Parameters

```
[out, retval] BSTR pName :       returned name.
```

Example

```
name = det.Name
```

Name ([in] BSTR Name)

Sets the name of the detector. Maximal 255 charatcters.

Parameters

```
[in] BSTR Name : new name.
```

Example

```
det.Name = „detectomat"
```

AttValue ([in] BSTR Attribute, [out, retval] VARIANT *pValue)

Returns a detector attribute. Please get the language independent attribute tag from the table at the end of this section.

Parameters

```
[in] BSTR Attribute :            Attribute name (see below)
[out, retval] VARIANT *pValue :  returned value of the attribute
```

Example

```
cycle = det.AttValue(„CONTROLLER")
```

AttValue ([in] BSTR Attribute, [in] VARIANT Value)

Sets a detector attribute. Please get the language independent attribute tag from the table at the end of this section.

Parameters

```
[in] BSTR Attribute :    Attribute name (see below)
[in] BSTR Value :        Attribute value. (type according to attribute)
```

Example

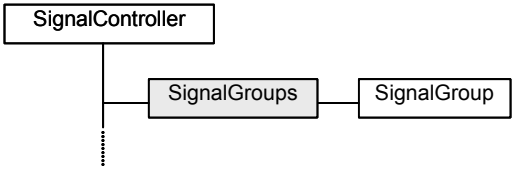
```
det.AttValue(„CONTROLLER") = 1
```

Attribute outline

R	W	Attribute	Description
✓		ID	Identifier number
✓	✓	NAME	Name
✓		CONTROLLER	Signal controller identifier number
✓	✓	DETECTION	Reading: 1 if there is or was a vehicle on the detector since the previous signal controller time step, else 0. Writing: will set an impulse like a vehicle that arrives in the current time step on the detector (i.e. a front end is detected) but does not leave it again during the current time step.
✓		HEADWAY	Time gap in seconds. 0 if the detector is occupied at the end of the current signal controller time step
✓		IMPULSE	Returns 1 if the impulse memory is 1 (i.e. a vehicle front end has been detected in the current signal controller time step). Otherwise it returns 0
✓		LINK	Link identifier number
✓		LANE	Lane number

R	W	Attribute	Description
✓		OCCUPANCY	Time in seconds since the arrival of a vehicle. 0 if no vehicle was detected at the end of the current signal controller time step 0
✓		OCCUPANCYRATE	Smoothed occupancy rate
✓		PRESENCE	1 if a vehicle has been on the detector at the end of the last simulation time step, else 0. (Note: This is different from the VAP function presence()!)
✓		VEHICLEID	If a vehicle front end has been detected in the current simulation time step it returns the ID number of this vehicle. Otherwise 0. If more than one vehicle front end has been detected the result is unspecified.
✓		VEHICLELENGTH LENGTH	Length (in the units of the current options) of the last vehicle that was detected within the current signal controller time step. If no vehicle was detected 0
✓		VELOCITY SPEED VEHICLESPEED	Speed (in the units of the current options) of the last vehicle that was detected within the current signal controller time step. If no vehicle was detected 0

2.41 SignalGroups



The SignalGroups object is a collection of SignalGroup objects (see page 144). It belongs to a SignalController object and can be accessed through the SignalGroups property of the ISignalController interface.

It contains all signal groups of the refered signal controller and enables iteration through the collection or individual access to a SignalGroup object.

Example

Instantiation of a SignalGroups object and access to all its SignalGroup objects:

```
DIM vissim AS Vissim
DIM groups AS SignalGroups
DIM group AS SignalGroup
SET vissim = NEW Vissim
vissim.LoadNet "c:\vissim\daten\example.inp"
SET groups = Vissim.Net.SignalControllers(1).SignalGroups
FOR EACH group IN groups  'access to _NewEnum to create an enumeration
…
NEXT group

'or also:

FOR i = 1 TO groups.Count
SET group = groups (i)  'or groups.Item(i)
…
NEXT i
```

2.41.1 Properties of the ISignalGroups Interface

_NewEnum ([out, retval] LPUNKNOWN *ppEnum)

This property creates a collection (or enumeration) of all SignalGroup objects. Once a collection is created, individual members can be returned using the GetSignalGroupByNumber method, while the entire collection can be iterated using a FOR ...TO ... NEXT statement. In Visual Basic you can use the statement FOR EACH…NEXT without using the property _NewEnum, which VB calls internally. (see SignalGroups example above).

Parameters

`[out, retval] LPUNKNOWN *ppEnum : returned Enumeration`

Item ([in] VARIANT Index, [out, retval] ISignalGroup **ppSG)

Returns a single SignalGroup of the collection selected by position. This is useful only when all signal groups shall be accessed (see the SignalGroups example above). To select a signal group by its identifier number use GetSignalGroupByNumber (see below). Since Item is the default property for a collection the following commands are equivalent:

```
SET sg = groups(1)
SET sg = groups.Item(1)
```

Parameters

```
[in] long Index :                       index between 1 and groups.Count
[out, retval] ISignalGroup **ppSG :     returned SignalGroup object
```

Count ([out, retval] long *pCount)

Returns the number of SignalGroup objects in the collection. See the SignalGroups example above.

Parameters

```
[out, retval] long *pCount :    returned number of objects.
```

2.41.2 Methods of the ISignalGroups Interface

GetSignalGroupByNumber ([in] long Number, [out, retval] ISignalGroup **ppSG)

Returns the SignalGroup object with the identifier number Number.

Parameters

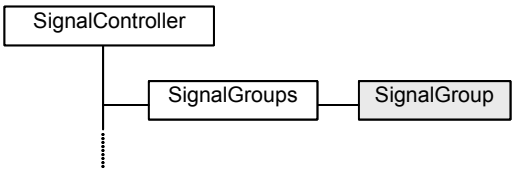
```
[in] long Number :                      identifier number
[out, retval] ISignalGroup **ppSG :     returned SignalGroup object
```

Example

```
DIM sg AS SignalGroup
SET sg = groups.GetSignalGroupByNumber (1)
```

2.42 SignalGroup

A SignalGroup object belongs to a controller's SignalGroups object. Through this a signal group can be accessed in two ways:



- access via iteration through the collection

```
DIM groups As SignalGroups
DIM group As SignalGroup
SET groups = Vissim.Net.SignalControllers(1).SignalGroups
FOR EACH group IN groups
List.AddItem group.ID
NEXT group
```

- individual access via identifier

```
DIM groups As SignalGroups
DIM group As SignalGroup
SET groups = Vissim.Net.SignalControllers(1).SignalGroups
SET group = groups.GetSignalGroupByNumber(10)
```

The SignalGroup object enables access to the properties of the signal group through the ISignalGroup interface.

2.42.1 Properties of the ISignalGroup Interface

ID ([out, retval] long *pID)

Returns the identifier of the signal group.

Parameters

`[out, retval] long pID : returned identifier.`

Example

```
id = group.ID
```

Name ([out, retval] BSTR *pName)

Returns the name of the signal group.

Parameters

`[out, retval] BSTR pName : returned name.`

Example

```
name = group.Name
```

Name ([in] BSTR Name)

Sets the name of the signal group. Maximal 255 charatcters.

Parameters

`[in] BSTR Name : new name.`

Example

```
group.Name = „phaseone"
```

AttValue ([in] BSTR Attribute, [out, retval] VARIANT *pValue)

Returns a signal group attribute. Please get the language independent attribute tag from the table at the end of this section.

Parameters

```
[in] BSTR Attribute :            Attribute name (see below)
[out, retval] VARIANT *pValue :  returned value of the attribute
```

Example

```
cycle = group.AttValue(„TYPE")
```

AttValue ([in] BSTR Attribute, [in] VARIANT Value)

Sets a signal group attribute. Please get the language independent attribute tag from the table at the end of this section.

Parameters

```
[in] BSTR Attribute :    Attribute name (see below)
[in] BSTR Value :        Attribute value. (type according to attribute)
```

Example

```
group.AttValue(„TYPE") = 1
```

Attribute outline

R	W	Attribute	Description
✓		ID	Identifier number
✓	✓	NAME	Name
✓		AMBER	Amber time [s] (0 for switching off)
✓	✓	CONTROLLER	Signal controller identifier number
✓	✓	GREENEND	Green end [s]
✓	✓	GREENEND2	Additional green end [s] (negative or 0 when non used)
✓	✓	REDAMBER	Red/Amber time [s] (0 for switching off)
✓	✓	REDEND	Red end [s]
✓	✓	REDEND2	Additional red end [s] (negative or 0 when non used)
✓		STATE	1 = Red, 2 = Redamber, 3 = Green, 4 = Amber, 5 = Off, 6 = Undefined, 7 = Flashing Amber, 8 = Flashing Red, 9 = Flashing Green, 10 = Flashing Redgreen, 11 = Greenamber, 12 = Off_red
✓	✓	TYPE	1 = cycle, 2 = Permanent green, 3 =

R	W	Attribute	Description
			Permanent Red

State ([out, retval] SignalState *pState)

Returns the current state of the signal group according to the following *SignalState* types table:

Attribute	Description
Red	Red
Redamber	Red and amber
Green	Green
Amber	Amber
Off	Off, meaning no light
Undefined	State not defined
Amber_f	Flashing amber
Red_f	Flashing red
Green_f	Flashing green
Redgreen_f	Flashing red and green
Greenamber	Green and amber
Off_red	Off, meaning red

Parameters

```
[out, retval] SignalSate *pState :        returned state
```

Example

```
DIM state AS SignalState
state = group.State
```

SignalHeads ([out, retval] ISignalHeads **ppSignalHeads)

Creates an instance of a SignalHeads object (see page 147), that gives individual access to the signal group elements of the network.

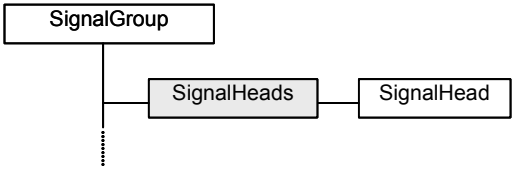
Parameters

```
[out, retval] ISignalHeads **ppSignalHeads :      returned SignalHeads object
```

Example

```
DIM shs AS SignalHeads
SET shs = group.SignalHeads
```

2.43 SignalHeads



The SignalHeads object is a collection of SignalHead objects (see page 149). It belongs to a SignalGroup object and can be accessed through the SignalHeads property of the ISignalGroup interface.

It contains all signal heads of the refered signal group and enables iteration through the collection or individual access to a SignalHead object.

Example

Instantiation of a SignalHeads object and access to all its SignalHead objects:

```
DIM vissim AS Vissim
DIM heads AS SignalHeads
DIM head AS SignalHead
SET vissim = NEW Vissim
vissim.LoadNet "c:\vissim\daten\example.inp"
SET heads = Vissim.Net.SignalControllers(1).SignalGroups(1).SignalHeads
FOR EACH head IN heads    'access to _NewEnum to create an enumeration
…
NEXT head

'or also:

FOR i = 1 TO heads.Count
SET head = heads(i)   'or heads.Item(i)
…
NEXT i
```

2.43.1 Properties of the ISignalHeads Interface

_NewEnum ([out, retval] LPUNKNOWN *ppEnum)

This property creates a collection (or enumeration) of all SignalHead objects. Once a collection is created, individual members can be returned using the GetSignalHeadByNumber method, while the entire collection can be iterated using a FOR ...TO ... NEXT statement. In Visual Basic you can use the statement FOR EACH…NEXT without using the property _NewEnum, which VB calls internally. (see SignalHeads example above).

Parameters

`[out, retval] LPUNKNOWN *ppEnum : returned Enumeration`

Item ([in] VARIANT Index, [out, retval] ISignalHead **ppSH)

Returns a single SignalHead of the collection selected by position. This is useful only when all signal heads shall be accessed (see the SignalHeads example above). To select a signal head by its identifier number use GetSignalHeadByNumber (see below). Since Item is the default property for a collection the following commands are equivalent:

```
SET sh = heads(1)
SET sh = heads.Item(1)
```

Parameters

[in] long Index :	index between 1 and heads.Count
[out, retval] ISignalController *ppSH :	returned SignalHead object

Count ([out, retval] long *pCount)

Returns the number of SignalHead objects in the collection. See the SignalHeads example above.

Parameters

[out, retval] long *pCount :	returned number of objects.

2.43.2 Methods of the ISignalHeads Interface

GetSignalHeadByNumber ([in] long Number, [out, retval] ISignalHead **ppSH)

Returns the SignalHead object with the identifier number Number.

Parameters

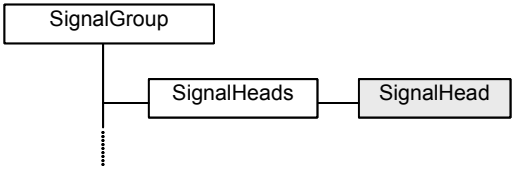
[in] long Number :	identifier number
[out, retval] ISignalHead **ppSH :	returned SignalHead object

Example

```
DIM sh AS SignalHead
SET sh = heads.GetSignalHeadByNumber (1)
```

2.44 SignalHead

A SignalHead object belongs to a signal group's SignalHeads object. Through this a signal head can be accessed in two ways:



- access via iteration through the collection

```
DIM heads As SignalHeads
DIM head As SignalHead
SET heads = Vissim.Net.SignalControllers(1).SignalGroups(1).SignalHeads
FOR EACH head IN heads
List.AddItem head.ID
NEXT heads
```

- individual access via identifier

```
DIM heads As SignalHeads
DIM head As SignalHeads
SET heads = Vissim.Net.SignalControllers(1).SignalGroups(1).SignalHeads
SET head = heads.GetSignalHeadByNumber(10)
```

The SignaHead object enables access to the properties of the signal head through the ISignalHead interface.

2.44.1 Properties of the ISignalHead Interface

ID ([out, retval] long *pID)

Returns the identifier of the signal head.

Parameters

```
[out, retval] long *pID :returned identifier.
```

Example

```
id = head.ID
```

Name ([out, retval] BSTR *pName)

Returns the name of the signal head.

Parameters

```
[out, retval] BSTR *pName :      returned name.
```

Example

```
name = head.Name
```

Name ([in] BSTR Name)

Sets the name of the signal head. Maximal 255 charatcters.

Parameters

[in] BSTR Name : new name.

Example

head.Name = „turnright"

AttValue ([in] BSTR Attribute, [out, retval] VARIANT *pValue)

Returns a signal head attribute. Please get the language independent attribute tag from the table at the end of this section.

Parameters

[in] BSTR Attribute : Attribute name (see below)
[out, retval] VARIANT *pValue : returned value of the attribute

Example

or = head.AttValue(„ORSIGNALGROUP")

AttValue ([in] BSTR Attribute, [in] VARIANT Value)

Sets a signal head attribute. Please get the language independent attribute tag from the table at the end of this section.

Parameters

[in] BSTR Attribute : Attribute name (see below)
[in] BSTR Value : Attribute value. (type according to attribute)

Example

head.AttValue(„ORSIGNALGROUP ") = 5

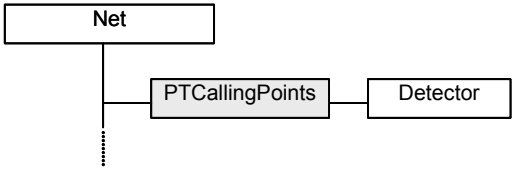
Attribute outline

R	W	Attribute	Description
✓		ID	Identifier number
✓	✓	NAME	Name
✓	✓	CONTROLLER	Signal controller number
✓	✓	GROUP	Signal group number
✓	✓	LINK	Link identifier number
✓	✓	LANE	Lane number
✓	✓	ORGROUP	Alternative signal group
✓	✓	LINKCOORD	Link coordinate (in current units)

Changes will affect immediately if called during a simulation.

It is required that CONTROLLER and GROUP are used consistently. Changing the signal controller requires that the current group number exists within the new assigned controller and, the other way around, assigning a new signal group requires its existence within the current signal controller.

2.45 PTCallingPoints



The PTCallingPoints object is a collection of Detector objects, defined as public transport calling point, (see page 139). It belongs to a Net object and can be accessed through the PTCallingPoints property of the INet interface.

It contains all PT calling points (special type of detector) of the the loaded network and enables iteration through the collection or individual access to a Detector object.

Example

Instantiation of a PTCallingPoints object and access to all its Detector objects defined as public transport calling point:

```
DIM vissim AS Vissim
DIM ptcps AS PTCallingPoints
DIM det AS Detector
SET vissim = NEW Vissim
vissim.LoadNet "c:\vissim\daten\example.inp"
SET dets = Vissim.Net.PTCallingPoints
FOR EACH det IN ptcps  'access to _NewEnum to create an enumeration
…
NEXT det

'or also:

FOR i = 1 TO ptcps.Count
SET det = ptcps  (i)  'or ptcps.Item(i)
…
NEXT i
```

2.45.1 Properties of the IPTCallingPoints Interface

_NewEnum ([out, retval] LPUNKNOWN *ppEnum)

This property creates a collection (or enumeration) of all Detector objects defined as public transport calling point. Once a collection is created, individual members can be returned using the GetDetectorByNumber method, while the entire collection can be iterated using a FOR ...TO ... NEXT statement. In Visual Basic you can use the statement FOR EACH…NEXT without using the property _NewEnum, which VB calls internally. (see Detectors example above).

Parameters

`[out, retval] LPUNKNOWN *ppEnum : returned Enumeration`

Item ([in] VARIANT Index, [out, retval] IDetector **ppPTCallingPoint)

Returns a single Detector of the collection selected by position. This is useful only when all detectors, defined as public transport calling point, shall be accessed (see the Detectors example above). To select a detector by its identifier number use GetPTCallingPointByNumber (see below). Since Item is the default property for a collection the following commands are equivalent:

```
SET det = ptcps(1)
SET det = ptcps.Item(1)
```

Parameters

[in] long Index :	index between 1 and ptcps.Count
[out, retval] IDetector **ppPTCallingPoint :	returned Detector object

Count ([out, retval] long *pCount)

Returns the number of Detector objects, defined as public transport calling point, in the collection. See the Detectors example above.

Parameters

[out, retval] long *pCount :	returned number of objects.

2.45.2 Methods of the IDetectors interface

GetPTCallingPointByNumber ([in] long Number, [out, retval] IDetector ** ppPTCallingPoint)

Returns the Detector object, defined as public transport calling point, with the identifier number Number.

Parameters

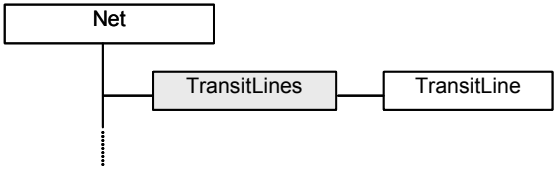
[in] long Number :	identifier number
[out, retval] IDetector ** ppPTCallingPoint:	returned Detector object

Example

```
DIM det AS Detector
SET det = ptcps.GetPTCallingPointrByNumber(1)
```

2.46 TransitLines

The Transit Lines object is a collection of Transit Line objects (see page 155).



It belongs to the Net object and can be accessed through the TransitLines property of the INet interface. It contains all transit lines of the loaded network and allows the iteration through all of them as well as individual access (see also TransitLine object).

Example

Instantiation of a TransitLines object and access to all its TransitLine objects:

```
DIM vissim As Vissim
DIM transitlines As TransitLines
DIM transitline As TransitLine
SET vissim = NEW Vissim
vissim.LoadNet "c:\vissim\daten\example.inp"
SET transitlines = Vissim.Net.TransitLines
FOR EACH transitline IN transitlines  'access to _NewEnum to create an enumeration
  …
NEXT transitline

'or also:

FOR i = 1 TO transitlines.Count
SET transitline = transitlines(i)  'or transitlines.Item(i)
…
NEXT i
```

2.46.1 Properties of the ITransitLines Interface

_NewEnum ([out, retval] LPUNKOWN *ppEnum)

This property creates a collection (or enumeration) of all TransitLine objects. Once a collection is created, individual members can be returned using the GetTransitLineByNumber method, while the entire collection can be iterated using a FOR ...TO ... NEXT statement. In Visual Basic you can use the statement FOR EACH…NEXT without using the property _NewEnum, which VB calls internally. (see TransitLines example above).

Parameters

```
[out, retval] LPUNKNOWN *ppEnum : returned Enumeration
```

Item ([in] VARIANT Index, [out, retval] ITransitLine **ppTransitLine)

Returns a single TransitLine of the collection selected by position. This is useful only when all transit lines shall be accessed (see the TransitLines example above). To select a transit line by its identifier number use GetTransitLineByNumber (see below). Since Item is the default property for a collection the following commands are equivalent:

```
SET transitline = transitlines(1)
SET transitline = transitlines.Item(1)
```

Parameters

```
[in] long Index:                                  index between 1 and Count
[out, retval] ITransitLine **ppTransitLine: returned TransitLine
```

Count ([out, retval] long *pCount)

Returns the number of TransitLine objects in the collection. See the TransitLines example above.

Parameters

```
[out, retval] long *pCount :      returned number of objects.
```

2.46.2 Methods of the ITransitLines Interface

GetTransitLineByNumber ([in] long Number, [out, retval] ITransitLine **ppTransitLine)

Returns the TransitLine object with the identifier number Number.

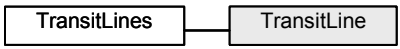
Parameters

```
[in] long Number :                                identifier number
[out, retval] ITransitLine **ppTransitLine :      returned TransitLine
```

Example

```
SET transitline = transitlines.GetTransitLineByNumber (2)
IF NOT (transitline IS NOTHING) THEN
name = transitline.AttValue("NAME")
END IF
```

2.47 TransitLine



A TransitLine object represents a transit line element and belongs to the TransitLines object.

It can be accessed through the TransitLines object to in two ways:

- access via iteration through the collection

```
DIM transitline As TransitLine
FOR EACH transitline IN vissim.Net.TransitLines
List.AddItem transitline.ID
NEXT transitline
```

- individual access via identifier

```
DIM transitline As TransitLine
SET transitline = vissim.Net.TransitLines.
GetTransitLineByNumber (12)
```

The TransitLine object enables access to the properties of the transit line through the ITransitLine interface.

2.47.1 Properties of the Interface ITransitLine

ID ([out, retval] long *pID)

Returns the transit line identifier number.

Parameters

`[out, retval] long *pID :returned identifier.`

Example

```
id = transitline.ID
```

Name ([out, retval] BSTR *pName)

Returns the name of the transit line.

Parameters

`[out, retval] BSTR *pName :  returned name.`

Example

```
name = transitline.Name
```

Name ([in] BSTR Name)

Sets the name of the transit line. Maximal 255 charatcters.

Parameters

`[in] BSTR Name : new name.`

Example

```
transitline.Name = „XXX“
```

AttValue ([in] BSTR Attribute, [out, retval] VARIANT *pValue)

Returns a transit line attribute. Please get the language independent attribute tag from the table at the end of this section.

Parameters

```
[in] BSTR Attribute :            attribute name (see table below)
[out, retval] VARIANT *pValue :  returned value of the attribute
```

Example

```
name = transitline.AttValue(„NAME“)
```

AttValue ([in] BSTR Attribute, [in] VARIANT Value)

Sets a transit line attribute. Please get the language independent attribute tag from the table at the end of this section.

Parameters

```
[in] BSTR Attribute :    attribute name (see below)
[in] BSTR Value :        attribute value. (type according to attribute)
```

Example

```
transitline.AttValue(„NAME“)=„XXX“
```

Attribute outline

R	W	Attribute	Description
✓		ID	Identifier number
✓	✓	NAME	Name

AttValue1 ([in] BSTR Attribute, [in] VARIANT Parameter, [out, retval] VARIANT *pValue)

Returns a transit line attribute with one parameter. Please get the language independent attribute tag from the table below.

Parameters

```
[in] BSTR Attribute :            attribute name (see below)
[in] VARIANT Parameter :         attribute dependent parameter (see below)
[out, retval] VARIANT *pValue :  returned value of the attribute
```

AttValue1 ([in] BSTR Attribute, [in] VARIANT Parameter, [in] VARIANT Value)

Sets a transit line attribute with one parameter. Please get the language independent attribute tag from the table below.

Parameters

```
[in] BSTR Attribute :    attribute name (see below)
[in] VARIANT Parameter : attribute dependent parameter (see below)
[in] VARIANT Value :     attribute value. (type according to attribute)
```

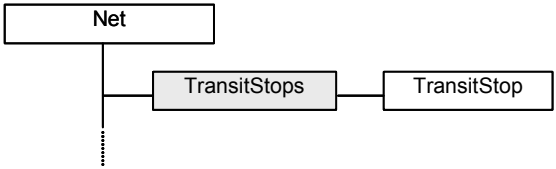
Attribute outline

R	W	Attribute	Description
✓	✓	ALIGHTINGPROB	Percentage of passengers that alight at that stop.
✓	✓	DWELLTIME	Dwell time distribution number to be used to determine the stop time

Changes will affect immediately if called during a simulation.

2.48 TransitStops

The Transit Stops object is a collection of Transit Stop objects (see page 160).



It belongs to the Net object and can be accessed through the TransitStops property of the INet interface. It contains all transit stops of the loaded network and allows the iteration through all of them as well as individual access (see also TransitStop object).

Example

Instantiation of a TransitStops object and access to all its TransitStop objects:

```
DIM vissim As Vissim
DIM transitstops As TransitStops
DIM transitstop As TransitStop
SET vissim = NEW Vissim
vissim.LoadNet "c:\vissim\daten\example.inp"
SET transitstops = Vissim.Net.TransitStops
FOR EACH transitstop IN transitstops  'access to _NewEnum to create an enumeration
  …
NEXT transitstop

'or also:

FOR i = 1 TO transitstops.Count
SET transitstop = transitstops(i)  'or transitstops.Item(i)
…
NEXT i
```

2.48.1 Properties of the ITransitStops Interface

_NewEnum ([out, retval] LPUNKOWN *ppEnum)

This property creates a collection (or enumeration) of all TransitStop objects. Once a collection is created, individual members can be returned using the GetTransitStopByNumber method, while the entire collection can be iterated using a FOR ...TO ... NEXT statement. In Visual Basic you can use the statement FOR EACH…NEXT without using the property _NewEnum, which VB calls internally. (see TransitStops example above).

Parameters

`[out, retval] LPUNKNOWN *ppEnum : returned Enumeration`

Item ([in] VARIANT Index, [out, retval] ITransitStop **ppTransitStop)

Returns a single TransitStop of the collection selected by position. This is useful only when all transit stops shall be accessed (see the TransitStops example above). To select a transit stop by its identifier number use GetTransitStopByNumber (see below). Since Item is the default property for a collection the following commands are equivalent:

```
SET transitstop = transitstops(1)
SET transitstop = transitstops.Item(1)
```

Parameters

[in] long Index:	index between 1 and Count
[out, retval] ITransitStop **ppTransitStop:	returned TransitStop

Count ([out, retval] long *pCount)

Returns the number of TransitStop objects in the collection. See the TransitStops example above.

Parameters

[out, retval] long *pCount :	returned number of objects.

2.48.2 Methods of the ITransitStops Interface

GetTransitStopByNumber ([in] long Number, [out, retval] ITransitStop **ppTransitStop)

Returns the TransitStop object with the identifier number Number.

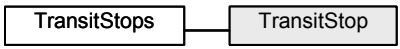
Parameters

[in] long Number :	identifier number
[out, retval] ITransitStop **ppTransitStop :	returned TransitStop

Example

```
SET transitstop = transitstops.GetTransitStopByNumber (2)
IF NOT (transitstop IS NOTHING) THEN
name = transitstop.AttValue("NAME")
END IF
```

2.49 TransitStop



A TransitStop object represents a transit stop element and belongs to the TransitStops object.

It can be accessed through the TransitStops object to in two ways:

- access via iteration through the collection

```
DIM transitstop As TransitStop
FOR EACH transitstop IN vissim.Net.TransitStops
List.AddItem transitstop.ID
NEXT transitstop
```

- individual access via identifier

```
DIM transitstop As TransitStop
SET transitstop = vissim.Net.TransitStops.
GetTransitStopByNumber (12)
```

The TransitStop object enables access to the properties of the transit stop through the ITransitStop interface.

2.49.1 Properties of the Interface ITransitStop

ID ([out, retval] long *pID)

Returns the transit stop identifier number.

Parameters

```
[out, retval] long *pID :returned identifier.
```

Example

```
id = transitstop.ID
```

Name ([out, retval] BSTR *pName)

Returns the name of the transit stop.

Parameters

```
[out, retval] BSTR *pName :      returned name.
```

Example

```
name = transitstop.Name
```

Name ([in] BSTR Name)

Sets the name of the transitstop. Maximal 255 charatcters.

Parameters

```
[in] BSTR Name : new name.
```

Example

```
transitstop.Name = „XXX"
```

AttValue ([in] BSTR Attribute, [out, retval] VARIANT *pValue)

Returns a transit stop attribute. Please get the language independent attribute tag from the table at the end of this section.

Parameters

```
[in] BSTR Attribute :            attribute name (see table below)
[out, retval] VARIANT *pValue :  returned value of the attribute
```

Example

```
name = transitstop.AttValue(„NAME“)
```

AttValue ([in] BSTR Attribute, [in] VARIANT Value)

Sets a transit stop attribute. Please get the language independent attribute tag from the table at the end of this section.

Parameters

```
[in] BSTR Attribute :    attribute name (see below)
[in] BSTR Value :        attribute value. (type according to attribute)
```

Example

```
transitstop.AttValue(„NAME“)=„XXX“
```

Attribute outline

R	W	Attribute	Description
✓		ID	Identifier number
✓	✓	NAME	Name

2.49.2 Methods of the ITransitStop Interface

GetVehicle ([out, retval] IVehicle **ppVehicle)

It returns the first vehicle serving on the transit stop.

Parameters

```
[out, retval] IVehicle **ppVehicle :      returned Vehicle object
```

Example

```
DIM veh AS Vehicle
SET veh = transitstop.GetVehicle()
passengers = veh.AttValue(“PASSENGERS”)
```

2.50 DataCollections

The DataCollections object is a collection of DataCollection objects (see page 165). It belongs to the Net object and can be accessed through the DataCollections property of the INet interface.

It contains all currently defined data collection measurements of the loaded network and enables iteration through the collection or individual access to a DataCollection object.

Example

Instantiation of a DataCollections object and access to all its DataCollection objects:

```
Dim vissim As Vissim
Dim datacollections As DataCollections
Dim datacollection As DataCollection
Set vissim = NEW Vissim
vissim.LoadNet "c:\vissim\daten\example.inp"
Set datacollections = Vissim.Net.DataCollections
FOR EACH datacollection IN datacollections   'access to _NewEnum
…
NEXT datacollection

'or also:

FOR i = 1 TO datacollections.Count
SET datacollection = datacollections(i)  'or datacollections.Item(i)
…
NEXT i
```

2.50.1 Properties of the IDataCollections Interface

_NewEnum ([out, retval] LPUNKNOWN *ppEnum)

This property creates a collection (or enumeration) of all DataCollection objects. Once a collection is created, individual members can be returned using the GetDataCollectionByNumber method, while the entire collection can be iterated using a FOR ...TO ... NEXT statement. In Visual Basic you can use the statement FOR EACH…NEXT without using the property _NewEnum, which VB calls internally (see DataCollections example above).

Parameters

```
[out, retval] LPUNKNOWN *ppEnum : returned Enumeration
```

Item ([in] VARIANT Index, [out, retval] IDataCollection **ppDC)

Returns a single DataCollection of the collection selected by position. This is useful only when all data collections shall be accessed (see the DataCollections example above). To select a data collection by its identifier number use GetDataCollectionByNumber (see below). Since Item is the default property for a collection the following commands are equivalent:

```
SET dc = DataCollections(1)
SET dc = DataCollections.Item(1)
```

Parameters

```
[in] long Index :                          index between 1 and DataCollections.Count
[out, retval] IDataCollection **ppDC :     returned DataCollection object
```

Count ([out, retval] long *pCount)

Returns the number of DataCollection objects in the collection. See the DataCollections example above.

Parameters

```
[out, retval] long *pCount :      returned number of objects.
```

2.50.2 Methods of the IDataCollections Interface

GetDataCollectionByNumber ([in] long Number, [out, retval] IDataCollection **ppDC)

Returns the DataCollection object with the identifier number Number.

Parameters

```
[in] long Number :                         identifier number
[out, retval] IDataCollection **ppDC :     returned DataCollection object
```

Example

```
DIM dc AS DataCollection
SET dc = DataCollections.GetDataCollectionByNumber(101)
```

GetIDs ([out] VARIANT *pIDs, [in, defaultvalue("")] BSTR Attribute, [in, optional] VARIANT Value, [in, defaultvalue(0)] short Compare)

Returns an array of the data collection identifier numbers that assert the condition “DataCollection(Attribute) <Compare> Value”, where Compare can get three different values: -1 for “less than”, 0 for “equal to” and 1 for “greater than”. If an empty attribute or no attribute is given an array with all identifier numbers of the collection will be returned.

Parameters

```
[out] VARIANT *pIDs: returned array of IDs (array of VARIANTs)
[in] BSTR Attribute: attribute to be compared. If empty the condition will always be true.
[in] VARIANT Value : value to be compared.
[in] short Compare : -1, 0 or 1 for “less than”, “equal to” or “greater that” respectively
```

Example

```
DIM ids() AS LONG
dcs.GetIDs ids ' ids will contain all IDs
```

GetMultiAttValues ([in] VARIANT IDs, [in] BSTR Attribute, [out] VARIANT *pValues)

Returns an array of values for the passed attribute tag and the requested data collections in the identifier numbers array IDs and in the same order. If no identifier number is requested (IDs is Empty) then the returned array will contain the values of all data collections in an unspecified order. If IDs is a identifier number but not an array then the method will return the value of the specified attribute for the passed ID.

Parameters

```
[in] VARIANT IDs :                    requested identifiers (array of VARIANTs)
[in] BSTR Attribute :    attribute name (see IDataCollection attribute table)
[out] VARIANT *pValues :         returned array of values.
```

Example

```
dcs.GetMultiAttValues Empty, "NAME", vals  ' vals will contain all names
```

SetMultiAttValues ([in] VARIANT IDs, [in] BSTR Attribute, [in] VARIANT Values)

Updates the attribute with the passed array of values for the requested data collections in the identifier numbers array IDs and in the same order. If not enough values are given the last value will be taken until complete the update. Exceeding values will not be taken in account. If no identifier number is requested (IDs is Empty) then all data collections will be considered. It is possible to pass a value instead of an array of values; in this case the same value will be used for the update.

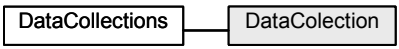
Parameters

```
[in] VARIANT IDs :       requested identifiers (array of VARIANTs)
[in] BSTR Attribute :    attribute name (see IDataCollection attribute table)
[in] VARIANT Values :    value or array of values.
```

Example

```
dcs.SetMultiAttValues Empty, "NAME", ""  ' remove all names
```

2.51 DataCollection



A DataCollection object represents a data collection measurement (defined through the configuration of Data collection for Offline Analysis) and belongs to the DataCollections object.

It can be accessed through the DataCollections object to in two ways:

- access via iteration through the collection

```
DIM datacollection As DataCollection
FOR EACH datacollection IN vissim.Net.DataCollections
List.AddItem datacollection.ID
NEXT datacollection
```

- individual access via identifier

```
DIM datacollection As DataCollection
SET datacollection = vissim.Net.DataCollections.GetDataCollectionByNumber (101)
```

The DataCollection object enables access to the properties of the data collection through the IDataCollection interface.

2.51.1 Properties of the Interface IDataCollection

ID ([out, retval] long *pID)

Returns the data collection identifier number. If the DataCollection object doesn't refer to a valid VISSIM data collection element anymore the returned value is 0.

Parameters

```
[out, retval] long *pID :returned identifier.
```

Example

```
id = datacollection.ID
```

Name ([out, retval] BSTR *pName)

Returns the data collection name. If no name was given the name of the first data collection point for this data collection will be used.

Parameters

```
[out, retval] BSTR *pName :      returned name.
```

Example

```
name = datacollection.Name
```

Name ([in] BSTR Name)

Sets the data collection's name.

Parameters

```
[in] BSTR Name : new name.
```

Example

```
datacollection.Name = „Friedhofeingang“
```

AttValue ([in] BSTR Attribute, [out, retval] VARIANT *pValue)

Returns a data collection attribute. Please get the language independent attribute tag from the table at the end of this section.

Parameters

```
[in] BSTR Attribute :            attribute name (see table below)
[out, retval] VARIANT *pValue :  returned value of the attribute
```

Example

```
name = datacollection.AttValue(„NAME“)
```

AttValue ([in] BSTR Attribute, [in] VARIANT Value)

Sets a data collection attribute. Please get the language independent attribute tag from the table at the end of this section.

Parameters

```
[in] BSTR Attribute :    attribute name (see below)
[in] BSTR Value :        attribute value. (type according to attribute)
```

Example

```
datacollection.AttValue(„NAME“)=„Friedhofeingang“
```

Attribute outline

R	W	Attribute	Description
✓		ID	Identifier number
✓	✓	NAME	Name
✓		VEHICLEIDS	Array of the current reported vehicle IDs within the interval.

VEHICLEIDS refers to the data collected during the last completed time interval.

2.51.2 Methods of the Interface IDataCollection

GetResult ([in] BSTR Parameter, [in] BSTR Function, [in] long VehicleClass [out, retval] VARIANT *pValue)

This method returns the last collected result (see notes below) of the previously configured parameter with its function and vehicle class. The returned value will be an 2d-array if the function “FREQUENCIES” is used. In this case, the second dimension will be of size 3 and will contain the lower and upper class limits and the result itself respectively.

Parameters

```
[in] BSTR Parameter :            parameter name (see below)
[in] BSTR Function :             function name (see below)
[in] long VehicleClass:          vehicle class number. 0 for all vehicle types
[out, retval] VARIANT *pValue :  returned value (real) or array of values (of reals)
```

Example

```
maxspeed = datacollection.GetResult(„SPEED", „MAX")
minspeed = datacollection.GetResult(„SPEED", „MIN")
freqs = datacollection.GetResult(„SPEED", „FREQUENCIES", 0)
FOR i = LBOUND(freqs) TO UPBOUND(freqs)
List.AddItem "From "+ CStr(freqs(i,0)) + ", To "+ CStr(freqs(i,1)) + " : "+
CStr(freqs(i,2))
NEXT i
```

Parameters outline

Parameter	Description
ACCELERATION	Acceleration [m/s2] [ft/s2]. MIN, MAX, MEAN, FREQUENCIES
LENGTH	Vehicle length [m] [ft]. MIN, MAX, MEAN, FREQUENCIES
MOTOTEMPERATURE	Cooling water temperature [°C]. MIN, MAX, MEAN, FREQUENCIES
NVEHICLES	Number of vehicles. SUM
NPERSONS	Number of people. MIN, MAX, MEAN, SUM, FREQUENCIES
OCCUPANCYRATE	Occupancy rate [%]. SUM
QUEUEDELTIME	Total queue delay time [s]. MIN, MAX, MEAN, SUM, FREQUENCIES
SPEED	Speed [km/h] [mph]. MIN, MAX, MEAN, FREQUENCIES
TACHO	Total distance traveled in the network [m] [ft]. MIN, MAX, MEAN, FREQUENCIES

Functions outline

Function	Description
MIN	Maximum value
MAX	Minimum value
MEAN	Mean value
SUM	Total sum
FREQUENCIES	All configured frequencies in an array

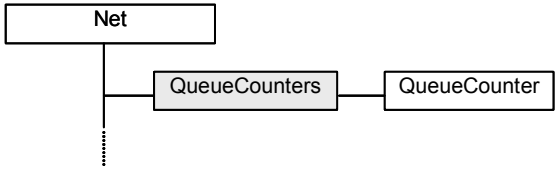
Note: The parameters “NVEHICLES” and “OCCUPANCYRATE” can be used with the function “SUM” only. For example:

```
result = datacollection.GetResult(„NVEHICLES“, „SUM“)
```

Note: The results refer to the data collected during the last completed time interval. The data of a time interval is available immediately after the last time step of the interval has been simulated but not if this is the last time step of the simulation period as well

Note: To receive results from collected data, a configuration must be defined in VISSIM and stored in a *.qmk file before requests can be done. The Offline Analysis option for data collections must be also enabled. Otherwise an error message will be returned (“The specified configuration is not defined within VISSIM”).

2.52 QueueCounters



The QueueCounters object is a collection of QueueCounter objects (see page 171). It belongs to the Net object and can be accessed through the QueueCounters property of the INet interface.

It contains all currently defined queue counters of the loaded network (during a simulation run including the temporary ones created for node evaluation) and enables iteration through the collection or individual access to a QueueCounter object.

Example

Instantiation of a QueueCounters object and access to all its QueueCounter objects:

```
Dim vissim As Vissim
Dim queuecounters As QueueCounters
Dim queuecounter As QueueCounter
Set vissim = NEW Vissim
vissim.LoadNet "c:\vissim\daten\example.inp"
Set queuecounters = Vissim.Net.QueueCounters
FOR EACH queuecounter IN queuecounters 'access to _NewEnum
…
NEXT queuecounter

'or also:

FOR i = 1 TO queuecounters.Count
SET queuecounter = queuecounters (i)  'or queuecounters.Item(i)
…
NEXT i
```

2.52.1 Properties of the IQueueCounters Interface

_NewEnum ([out, retval] LPUNKNOWN *ppEnum)

This property creates a collection (or enumeration) of all QueueCounter objects. Once a collection is created, individual members can be returned using the GetQueueCounterByNumber method, while the entire collection can be iterated using a FOR ...TO ... NEXT statement. In Visual Basic you can use the statement FOR EACH…NEXT without using the property _NewEnum, which VB calls internally (see QueueCounters example above).

Parameters

```
[out, retval] LPUNKNOWN *ppEnum : returned Enumeration
```

Item ([in] VARIANT Index, [out, retval] IQueueCounter **ppQC)

Returns a single QueueCounter of the collection selected by position. This is useful only when all queue counters shall be accessed (see the QueueCounters example above). To select a data collection by its identifier number use GetQueueCounterByNumber (see below). Since Item is the default property for a collection the following commands are equivalent:

```
SET qc = QueueCounters(1)
SET qc = QueueCounters.Item(1)
```

Parameters

```
[in] long Index :                       index between 1 and QueueCounters.Count
[out, retval] IQueueCounter **ppQC :    returned QueueCounter object
```

Count ([out, retval] long *pCount)

Returns the number of QueueCounter objects in the collection. See the QueueCounters example above.

Parameters

```
[out, retval] long *pCount :    returned number of objects.
```

2.52.2 Methods of the IQueueCounters Interface

GetQueueCounterByNumber ([in] long Number, [out, retval] IQueueCounter **ppQC)

Returns the QueueCounter object with the identifier number Number.

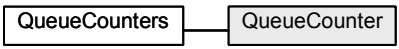
Parameters

```
[in] long Number :                      identifier number
[out, retval] IQueueCounter **ppQC :    returned QueueCounter object
```

Example

```
DIM qc AS QueueCounter
SET qc = QueueCounters.GetQueueCounterByNumber (123)
```

2.53 QueueCounter



A QueueCounter object represents a queue counter element and belongs to the QueueCounters object. It can be accessed through the QueueCounters object to in two ways:

- access via iteration through the collection

```
DIM queuecounter As QueueCounter
FOR EACH queuecounter IN vissim.Net.QueueCounters
List.AddItem queuecounter.ID
NEXT queuecounter
```

- individual access via identifier

```
DIM queuecounter As QueueCounter
SET queuecounter = vissim.Net.QueueCounters.GetQueueCounterByNumber(123)
```

The QueueCounter object enables access to the properties of the queue counter through the IQueueCounter interface.

2.53.1 Properties of the Interface IQueueCounter

ID ([out, retval] long *pID)

Returns the queue counter identifier number. If the QueueCounter object doesn't refer to a valid VISSIM queue counter element anymore the returned value is 0.

Parameters

```
[out, retval] long *pID :returned identifier.
```

Example

```
id = queuecounter.ID
```

Name ([out, retval] BSTR *pName)

Returns the queue counter name.

Parameters

```
[out, retval] BSTR *pName :      returned name.
```

Example

```
name = queuecounter.Name
```

Name ([in] BSTR Name)

Sets the name of the queue counter.

Parameters

```
[in] BSTR Name : new name.
```

Example

```
queuecounter.Name = „xxx"
```

AttValue ([in] BSTR Attribute, [out, retval] VARIANT *pValue)

Returns a queue counter attribute. Please get the language independent attribute tag from the table at the end of this section.

Parameters

```
[in] BSTR Attribute :            attribute name (see table below)
[out, retval] VARIANT *pValue :  returned value of the attribute
```

Example

```
name = queuecounter.AttValue(„NAME")
```

AttValue ([in] BSTR Attribute, [in] VARIANT Value)

Sets a queue counter attribute. Please get the language independent attribute tag from the table at the end of this section.

Parameters

```
[in] BSTR Attribute :    attribute name (see below)
[in] BSTR Value :        attribute value. (type according to attribute)
```

Example

```
queuecounter.AttValue(„NAME")=„xxx"
```

Attribute outline

R	W	Attribute	Description
✓		ID	Identifier number
✓	✓	NAME	Name

2.53.2 Methods of the Interface IQueueCounter

GetResult ([in] double Time, [in] BSTR Parameter, [out, retval] VARIANT *pValue)

This method returns the last collected result (see note below) for the requested named parameter (see parameters table below). The returned value refers to the data collected up to this moment for the time interval enclosing the specified time point.

Parameters

```
[in] double Time :               Time point in seconds
[in] BSTR Parameter :            parameter name (see below)
[out, retval] VARIANT *pValue :  returned value (real number)
```

Example

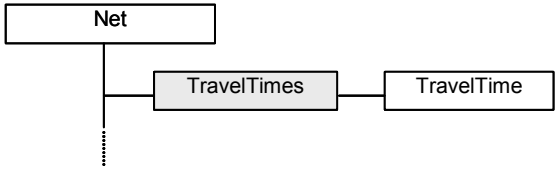
```
mean = queuecounter.GetResult(600, „MEAN")
max = queuecounter.GetResult(600, "MAX")
stops = queuecounter.GetResult(600, "STOPS")
```

Parameters outline

Parameter	Description
MEAN	Average queue length ([m] or [ft], depending on current unit selection)
MAX	Maximum queue length ([m] or [ft], depending on current unit selection)
NSTOPS	Number of stops in the queue area

To get results, the Offline Analysis option for queue counters must be enabled. Otherwise the result will be 0.0.

2.54 TravelTimes



The TravelTimes object is a collection of TravelTime objects (see page 176). It belongs to the Net object and can be accessed through the TravelTimes property of the INet interface.

It contains all currently defined travel time measurements of the loaded network (during a simulation run including the temporary ones created for node evaluation) and enables iteration through the collection or individual access to a TravelTime object.

Example

Instantiation of a TravelTimes object and access to all its TravelTime objects:

```
Dim vissim As Vissim
Dim traveltimes As TravelTimes
Dim traveltime As TravelTime
Set vissim = NEW Vissim
vissim.LoadNet "c:\vissim\daten\example.inp"
Set traveltimes = Vissim.Net.TravelTimes
FOR EACH traveltime IN traveltimes 'access to _NewEnum
…
NEXT traveltime

'or also:

FOR i = 1 TO traveltimes.Count
SET traveltime = traveltimes (i)  'or traveltimes.Item(i)
…
NEXT i
```

2.54.1 Properties of the ITravelTimes Interface

_NewEnum ([out, retval] LPUNKNOWN *ppEnum)

This property creates a collection (or enumeration) of all TravelTime objects. Once a collection is created, individual members can be returned using the GetTravelTimeByNumber method, while the entire collection can be iterated using a FOR ...TO ... NEXT statement. In Visual Basic you can use the statement FOR EACH…NEXT without using the property _NewEnum, which VB calls internally (see TravelTimes example above).

Parameters

`[out, retval] LPUNKNOWN *ppEnum : returned Enumeration`

Item ([in] VARIANT Index, [out, retval] ITravelTime **ppTT)

Returns a single TravelTime of the collection selected by position. This is useful only when all travel time measurements shall be accessed (see the TravelTimes example above). To select a data collection by its identifier number use GetTravelTimeByNumber (see below). Since Item is the default property for a collection the following commands are equivalent:

```
SET tt = TravelTimes(1)
SET tt = TravelTimes.Item(1)
```

Parameters

[in] long Index :	index between 1 and TravelTimes.Count
[out, retval] ITravelTime **ppTT :	returned TravelTime object

Count ([out, retval] long *pCount)

Returns the number of TravelTime objects in the collection. See the TravelTimes example above.

Parameters

[out, retval] long *pCount :	returned number of objects.

2.54.2 Methods of the ITravelTimes Interface

GetTravelTimeByNumber ([in] long Number, [out, retval] ITravelTime **ppTT)

Returns the TravelTime object with the identifier number Number.

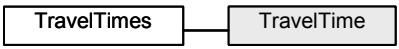
Parameters

[in] long Number :	identifier number
[out, retval] ITravelTime **ppTT :	returned TravelTime object

Example

```
DIM tt AS TravelTime
SET tt = TravelTimes.GetTravelTimeByNumber (123)
```

2.55 TravelTime



A TravelTime object represents a travel time measurement element and belongs to the TravelTimes object. It can be accessed through the TravelTimes object to in two ways:

- access via iteration through the collection

```
DIM traveltime As TravelTime
FOR EACH traveltime IN vissim.Net.TravelTimes
List.AddItem traveltime.ID
NEXT traveltime
```

- individual access via identifier

```
DIM traveltime As TravelTime
SET traveltime = vissim.Net. TravelTimes.GetTravelTimeByNumber (123)
```

The TravelTime object enables access to the properties of the travel time measurement through the ITravelTime interface.

2.55.1 Properties of the Interface ITravelTime

ID ([out, retval] long *pID)

Returns the travel time measurement identifier number. If the TravelTime object doesn't refer to a valid VISSIM travel time measurement element anymore the returned value is 0.

Parameters

```
[out, retval] long *pID :returned identifier.
```

Example

```
id = traveltime.ID
```

Name ([out, retval] BSTR *pName)

Returns the travel time measurement name.

Parameters

```
[out, retval] BSTR *pName :      returned name.
```

Example

```
name = traveltime.Name
```

Name ([in] BSTR Name)

Sets the name of the travel time measurement. Maximal 255 charatcters.

Parameters

```
[in] BSTR Name : new name.
```

Example

```
traveltime.Name = „xxx"
```

AttValue ([in] BSTR Attribute, [out, retval] VARIANT *pValue)

Returns a travel time measurement attribute. Please get the language independent attribute tag from the table at the end of this section.

Parameters

```
[in] BSTR Attribute :            attribute name (see table below)
[out, retval] VARIANT *pValue :  returned value of the attribute
```

Example

```
name = traveltime.AttValue(„NAME“)
```

AttValue ([in] BSTR Attribute, [in] VARIANT Value)

Sets a travel time measurement attribute. Please get the language independent attribute tag from the table at the end of this section.

Parameters

```
[in] BSTR Attribute :    attribute name (see below)
[in] BSTR Value :        attribute value. (type according to attribute)
```

Example

```
traveltime.AttValue(„NAME“)=„xxx“
```

Attribute outline

R	W	Attribute	Description
✓		ID	Identifier number
✓	✓	NAME	Name

2.55.2 Methods of the Interface ITravelTime

GetResult ([in] double Time, [in] BSTR Parameter, [in] BSTR Function, [in] long VehicleClass, [out, retval] VARIANT *pValue)

This method returns the collected result for the requested named parameter (see parameters table below) and vehicle class. The returned value refers to the data collected up to this moment for the time interval enclosing the specified time point (it includes only travel times of vehicles that have already passed the travel time destination section). The Function parameter has currently no meaning.

Parameters

```
[in] double Time :               Time point in seconds
[in] BSTR Parameter :            parameter name (see below)
[in] BSTR Function :             not used
[in] long VehicleClass:          vehicle class number. 0 for all vehicle types
[out, retval] VARIANT *pValue:   returned value (real number)
```

Example

```
nv = traveltime.GetResult(600, „NVEHICLES“, “”, 0) ‘vehicle throughput
tt = traveltime.GetResult(600, “TRAVELTIME”, “”, 1) ‘travel time for vehicles of class 1
```

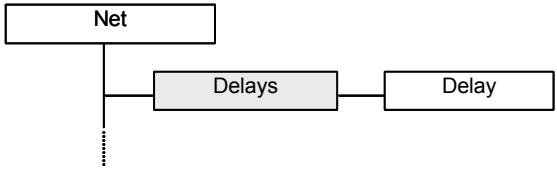
Parameters outline

Parameter	Description
NVEHICLES	Vehicle throughput (numbe of vehicle measured)
TRAVELTIME	Travel time [s]

To get results, the Offline Analysis option for travel times must be enabled. Otherwise the result will be 0.0.

Data for each defined vehicle class can be requested even if the vehicle class is not selected for the travel time measurement.

2.56 Delays



The Delays object is a collection of Delay objects (see page 181). It belongs to the Net object and can be accessed through the Delays property of the INet interface.

It contains all currently defined delay measurements of the loaded network (during a simulation run including the temporary ones created for node evaluation) and enables iteration through the collection or individual access to a Delay object.

Example

Instantiation of a Delays object and access to all its Delay objects:

```
Dim vissim As Vissim
Dim delays As Delays
Dim delay As Delay
Set vissim = NEW Vissim
vissim.LoadNet "c:\vissim\daten\example.inp"
Set delays = Vissim.Net.Delays
FOR EACH delay IN delays 'access to _NewEnum
…
NEXT delay

'or also:

FOR i = 1 TO delays.Count
SET delay = delays (i)  'or delays.Item(i)
…
NEXT i
```

2.56.1 Properties of the IDelays Interface

_NewEnum ([out, retval] LPUNKNOWN *ppEnum)

This property creates a collection (or enumeration) of all Delay objects. Once a collection is created, individual members can be returned using the GetDelayByNumber method, while the entire collection can be iterated using a FOR ...TO ... NEXT statement. In Visual Basic you can use the statement FOR EACH…NEXT without using the property _NewEnum, which VB calls internally (see Delays example above).

Parameters

`[out, retval] LPUNKNOWN *ppEnum : returned Enumeration`

Item ([in] VARIANT Index, [out, retval] IDelay **ppDelay)

Returns a single Delay of the collection selected by position. This is useful only when all delay measurements shall be accessed (see the Delays example above). To select a data collection by its identifier number use GetDelayByNumber (see below). Since Item is the default property for a collection the following commands are equivalent:

```
SET delay = Delays(1)
SET delay = Delays.Item(1)
```

Parameters

```
[in] long Index :                index between 1 and Delays.Count
[out, retval] IDelay **ppDelay : returned Delay object
```

Count ([out, retval] long *pCount)

Returns the number of Delay objects in the collection. See the Delays example above.

Parameters

```
[out, retval] long *pCount :     returned number of objects.
```

2.56.2 Methods of the IDelays Interface

GetDelayByNumber ([in] long Number, [out, retval] IDelay **ppDelay)

Returns the Delay object with the identifier number Number.

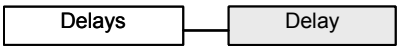
Parameters

```
[in] long Number :               identifier number
[out, retval] IDelay **ppDelay : returned Delay object
```

Example

```
DIM delay AS Delay
SET delay = Delays.GetDelayByNumber (123)
```

2.57 Delay



A Delay object represents a delay measurement element and belongs to the Delays object. It can be accessed through the Delays object to in two ways:

- access via iteration through the collection

```
DIM delay As Delay
FOR EACH delay IN vissim.Net.Delays
List.AddItem delay.ID
NEXT delay
```

- individual access via identifier

```
DIM delay As Delay
SET delay = vissim.Net. Delays.GetDelayByNumber (123)
```

The Delay object enables access to the properties of the delay through the IDelay interface.

2.57.1 Properties of the Interface IDelay

ID ([out, retval] long *pID)

Returns the delay identifier number. If the Delay object doesn't refer to a valid VISSIM delay measurement element anymore the returned value is 0.

Parameters

`[out, retval] long *pID :returned identifier.`

Example

```
id = delay.ID
```

Name ([out, retval] BSTR *pName)

Returns the delay measurement name. If no name was given the name of the first travel time measurement for this delay will be used.

Parameters

`[out, retval] BSTR *pName :  returned name.`

Example

```
name = delay.Name
```

Name ([in] BSTR Name)

Sets the name of the delay measurement.

Parameters

`[in] BSTR Name : new name.`

Example

```
delay.Name = „xxx"
```

AttValue ([in] BSTR Attribute, [out, retval] VARIANT *pValue)

Returns a delay measurement attribute. Please get the language independent attribute tag from the table at the end of this section.

Parameters

```
[in] BSTR Attribute :            attribute name (see table below)
[out, retval] VARIANT *pValue :  returned value of the attribute
```

Example

```
name = delay.AttValue(„NAME")
```

AttValue ([in] BSTR Attribute, [in] VARIANT Value)

Sets a delay measurement attribute. Please get the language independent attribute tag from the table at the end of this section.

Parameters

```
[in] BSTR Attribute :    attribute name (see below)
[in] BSTR Value :        attribute value. (type according to attribute)
```

Example

```
delay.AttValue(„NAME")=„xxx"
```

Attribute outline

R	W	Attribute	Description
✓		ID	Identifier number
✓	✓	NAME	Name

2.57.2 Methods of the Interface IDelay

GetResult ([in] double Time, [in] BSTR Parameter, [in] BSTR Function, [in] long VehicleClass, [out, retval] VARIANT *pValue)

This method returns the collected result for the requested named parameter (see parameters table below) and vehicle class. The returned value refers to the data collected up to this moment for the time interval enclosing the specified time point. The Function parameter has currently no meaning.

Parameters

```
[in] double Time :               Time point in seconds
[in] BSTR Parameter :            parameter name (see below)
[in] BSTR Function :             not used
[in] long VehicleClass:          vehicle class number. 0 for all vehicle types
[out, retval] VARIANT *pValue:   returned value (real number)
```

Example

```
delay = delay.GetResult(600, „DELAY","", 0) 'average total delay per vehicle
delay = delay.GetResult(600, „DELAY","", 1) 'average total delay per vehicle for class 1
```

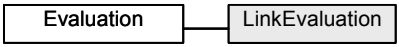
Parameters outline

Parameter	Description
DELAY	Average total delay per vehicle [s]
PERSONS	Average total delay per person [s]
NPERSONS	Persons throughput
NVEHICLES	Vehicles throughput
NSTOPS	Average number of stops per vehicle [s]
STOPPEDDELAY	Average stand still time per vehicle [s]

To get results, the Offline Analysis option for delays must be enabled. Otherwise the result will be 0.0.

Data for each defined vehicle class can be requested even if the vehicle class is not selected for the delay measurement.

2.58 LinkEvaluation



This object allows the configuration of link evaluations through the methods and properties of the ILinkEvaluation interface.

Offline evaluations are possible setting the appropriated flag (see attribute LINK of the IEvaluation interface on page 46) and online evaluations are possible through the method GetSegmentResult() of the ILink interface (refer to page 53).

Example

Loading an existing link evaluation configuration (*.sak):

```
DIM linkeval As LinkEvaluation
SET linkeval = vissim.Evaluation.LinkEvaluation
linkeval.LoadConfiguration("c:\vissim\daten\example.sak")
```

2.58.1 Properties of the Interface ILinkEvaluation

AttValue ([in] BSTR Attribute, [out, retval] VARIANT *pValue)

Returns a configuration's attribute. Please get the language independent attribute tag from the table at the end of this section.

Parameters

```
[in] BSTR Attribute :            attribute name (see table below)
[out, retval] VARIANT *pValue :  returned value of the attribute
```

Example

```
interval = linkeval.AttValue(„INTERVAL")
```

AttValue ([in] BSTR Attribute, [in] VARIANT Value)

Sets a configuration's attribute. Please get the language independent attribute tag from the table at the end of this section.

Parameters

```
[in] BSTR Attribute :    attribute name (see table below)
[in] BSTR Value :        attribute value. (type according to attribute)
```

Example

```
linkeval.AttValue(„UNTIL")= 3600
```

Attribute outline

R	W	Attribute	Description
✓	✓	DATABASE	Database flag
✓	✓	EXPORT	Export to VISUM

R	W	Attribute	Description
✓	✓	FILE	Write evaluation file flag (true/false)
✓	✓	FILENAME	Path and filename for the configuration file
✓	✓	FROM	First second
✓	✓	INTERVAL	Data collection interval
✓	✓	PERLANE	Separate Data collection for each lane
✓	✓	TABLENAME	Database name
✓	✓	UNTIL	Last second

Setting the FILE attribute to FALSE avoids writing the *.STR evaluation file while collecting evaluation data (if link segment evaluation is activated) for possible online requests. This attribute is specific to the COM interface and is global to the VISSIM program (i.e. it will preserve its state independently from the loaded *.INP file).

2.58.2 Methods of the Interface ILinkEvaluation

LoadConfiguration ([in] BSTR ConfigurationPath)

Loads a link evaluation configuration file (*.sak). No check is done of the file extension and path. It is up to the user to make sure that the file has the appropriate format. If no file path is given, a file browser dialog will appear.

Parameters

```
[in] BSTR ConfigurationPath :    path incl. filename of configuration file to be loaded
```

Example

```
linkeval.LoadConfiguration("example.sak")
```

SaveConfiguration ([in] BSTR ConfigurationPath)

Saves the currently selected configuration to the file in VISISM format (*.sak). No check of the file extension and working path is done.

Parameters

```
[in] BSTR ConfigurationPath :    path incl. filename of configuration file to be saved
```

Example

```
linkeval.SaveConfiguration("example.sak")
```


When loading/saving a configuration with LoadConfiguration(), the configuration's filename for the link evaluation of the currently loaded network will not be updated. To do that use the "FILENAME" attribute. If another path than the current network working path is used, the simulation will show an error on starting (if link evaluation is activated).

AddParameter ([in] BSTR Parameter, [in] long VehicleClass)

Adds a new parameter to evaluate, restricted to the given vehicle class or for all vehicle classes if VehicleClass is 0. No check is done for repetitions.

Parameters

```
[in] BSTR Parameter :    parameter name (see table below)
[in] long VehicleClass : vehicle class number or 0 for all classes
```

Example

```
linkeval.AddParameter("SPEED", 0)  'segment average speed for all vehicle types
```

RemoveParameter ([in] BSTR Parameter, [in] long VehicleClass)

Removes the specified parameter for the for the given vehicle class (0 for all vehicle classes) from the list of evaluated parameters. If there is no entry matching the parameter/vehicle class combination nothing will be done. If there are several matches the only first one will be removed.

Parameters

```
[in] BSTR Parameter :    parameter name (see table below)
[in] long VehicleClass : vehicle class number
```

Example

```
linkeval.RemoveParameter("SPEED", 0)  'segment average speed for all vehicle types
```

Parameter outline

R	W	Parameter	Description
✓	✓	TIMESTEP	Simulation time step [s]
✓	✓	SPEED	Average speed [km/h] or [mph]
✓	✓	VOLUME	Average volume [veh/h]
✓	✓	NVEHICLES	Cumulated number of vehicles
✓	✓	DENSITY	Average density [veh/km]
✓	✓	DELAY	Relative lost time [s/s]

2.59 DataCollectionEvaluation

This object allows the configuration of data collection evaluations through the methods and properties of the IData-CollectionEvaluation interface.

Offline evaluations are possible setting the appropriated flag (see attribute DATACOLLECTION of the IEvaluation interface on page 46) and online evaluations are possible through the method GetResult() of the IDataCollection interface.

2.59.1 Properties of the Interface IDataCollectionEvaluation

AttValue ([in] BSTR Attribute, [out, retval] VARIANT *pValue)

Returns a configuration's attribute. Please get the language independent attribute tag from the table at the end of this section.

Parameters

```
[in] BSTR Attribute :            attribute name (see table below)
[out, retval] VARIANT *pValue :  returned value of the attribute
```

Example

```
writtingfile = dceval.AttValue(„FILE")
```

AttValue ([in] BSTR Attribute, [in] VARIANT Value)

Sets a configuration's attribute. Please get the language independent attribute tag from the table at the end of this section.

Parameters

```
[in] BSTR Attribute :    attribute name (see table below)
[in] BSTR Value :        attribute value. (type according to attribute)
```

Example

```
dceval.AttValue(„FILE")= FALSE
```

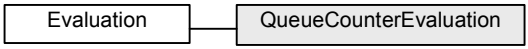
Attribute outline

R	W	Attribute	Description
✓	✓	COMPILED	Write the flag for compiled output (true/false)
✓	✓	FILE	Write evaluation file flag (true/false)
✓	✓	RAW	Write the flag for raw data output (true/false)

Setting the FILE attribute to FALSE avoids writing the *.MES and *.MER evaluation files while collecting evaluation data (if data collection evaluation is activated) for possible online requests. This attribute is specific to the COM interface and is global to the VISSIM program (i.e. it will preserve its state independently from the loaded *.INP file).

2.59.2 Methods of the Interface IDataCollectionEvaluation

2.60 QueueCounterEvaluation



This object allows the configuration of queue counter evaluations through the methods and properties of the IQueueCounterEvaluation interface.

Offline evaluations are possible setting the appropriated flag (see attribute QUEUECOUNTER of the IEvaluation interface on page 46). and online evaluations are possible through the method GetResult() of the IQueueCounter interface.

2.60.1 Properties of the Interface IQueueCounterEvaluation

AttValue ([in] BSTR Attribute, [out, retval] VARIANT *pValue)

Returns a configuration's attribute. Please get the language independent attribute tag from the table at the end of this section.

Parameters

```
[in] BSTR Attribute :            attribute name (see table below)
[out, retval] VARIANT *pValue :  returned value of the attribute
```

Example

```
writtingfile = qceval.AttValue(„FILE")
```

AttValue ([in] BSTR Attribute, [in] VARIANT Value)

Sets a configuration's attribute. Please get the language independent attribute tag from the table at the end of this section.

Parameters

```
[in] BSTR Attribute :    attribute name (see table below)
[in] BSTR Value :        attribute value. (type according to attribute)
```

Example

```
qceval.AttValue(„FILE")= FALSE
```

Attribute outline

R	W	Attribute	Description
✓	✓	FILE	Write evaluation file flag (true/false)

Setting the FILE attribute to FALSE avoids writing the *.STZ evaluation file while collecting evaluation data (if queue counter evaluation is activated) for possible online requests. This attribute is specific to the COM interface and is global to the VISSIM program (i.e. it will preserve its state independently from the loaded *.INP file).

2.60.2 Methods of the Interface IQueueCounterEvaluation

2.61 TravelTimeEvaluation

This object allows the configuration of travel time evaluations through the methods and properties of the ITravelTimeEvaluation interface.

Offline evaluations are possible setting the appropriated flag (see attribute TRAVELTIME of the IEvaluation interface on page 46).

2.61.1 Properties of the Interface ITravelTimeEvaluation

AttValue ([in] BSTR Attribute, [out, retval] VARIANT *pValue)

Returns a configuration's attribute. Please get the language independent attribute tag from the table at the end of this section.

Parameters

```
[in] BSTR Attribute :            attribute name (see table below)
[out, retval] VARIANT *pValue :  returned value of the attribute
```

Example

```
writtingfile = tteval.AttValue(„FILE")
```

AttValue ([in] BSTR Attribute, [in] VARIANT Value)

Sets a configuration's attribute. Please get the language independent attribute tag from the table at the end of this section.

Parameters

```
[in] BSTR Attribute :    attribute name (see table below)
[in] BSTR Value :        attribute value. (type according to attribute)
```

Example

```
tteval.AttValue(„FILE")= FALSE
```

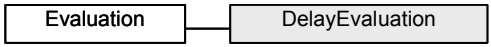
Attribute outline

R	W	Attribute	Description
✓	✓	COMPILED	Write the flag for compiled output (true/false)
✓	✓	FILE	Write evaluation file flag (true/false)
✓	✓	RAW	Write the flag for raw data output (true/false)

Setting the FILE attribute to FALSE avoids writing the *.RSZ and *.RSR evaluation files while collecting evaluation data (if travel time evaluation is activated) for possible online requests. This attribute is specific to the COM interface and is global to the VISSIM program (i.e. it will preserve its state independently from the loaded *.INP file).

2.61.2 Methods of the Interface ITravelTimeEvaluation

2.62 DelayEvaluation



This object allows the configuration of delay evaluations through the methods and properties of the IDelayEvaluation interface.

Offline evaluations are possible setting the appropriated flag (see attribute DELAY of the IEvaluation interface on page 46).

2.62.1 Properties of the Interface IDelayEvaluation

AttValue ([in] BSTR Attribute, [out, retval] VARIANT *pValue)

Returns a configuration's attribute. Please get the language independent attribute tag from the table at the end of this section.

Parameters

```
[in] BSTR Attribute :            attribute name (see table below)
[out, retval] VARIANT *pValue :  returned value of the attribute
```

Example

```
writtingfile = deval.AttValue(„FILE")
```

AttValue ([in] BSTR Attribute, [in] VARIANT Value)

Sets a configuration's attribute. Please get the language independent attribute tag from the table at the end of this section.

Parameters

```
[in] BSTR Attribute :    attribute name (see table below)
[in] BSTR Value :        attribute value. (type according to attribute)
```

Example

```
deval.AttValue(„FILE")= FALSE
```

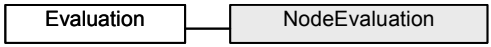
Attribute outline

R	W	Attribute	Description
✓	✓	COMPILED	Write the flag for compiled output (true/false)
✓	✓	FILE	Write evaluation file flag (true/false)
✓	✓	RAW	Write the flag for raw data output (true/false)

Setting the FILE attribute to FALSE avoids writing the *.VLZ and *.VLR evaluation files while collecting evaluation data (if delay evaluation is activated) for possible online requests. This attribute is specific to the COM interface and is global to the VISSIM program (i.e. it will preserve its state independently from the loaded *.INP file).

2.62.2 Methods of the Interface IDelayEvaluation

2.63 NodeEvaluation



This object allows the configuration of node evaluations through the methods and properties of the INodeEvaluation interface.

Offline evaluations are possible setting the appropriated flag (see attribute NODE of the IEvaluation interface on page 46).

2.63.1 Properties of the Interface INodeEvaluation

AttValue ([in] BSTR Attribute, [out, retval] VARIANT *pValue)

Returns a configuration's attribute. Please get the language independent attribute tag from the table at the end of this section.

Parameters

```
[in] BSTR Attribute :            attribute name (see table below)
[out, retval] VARIANT *pValue :  returned value of the attribute
```

Example

```
writtingfile = nodeeval.AttValue(„FILE“)
```

AttValue ([in] BSTR Attribute, [in] VARIANT Value)

Sets a configuration's attribute. Please get the language independent attribute tag from the table at the end of this section.

Parameters

```
[in] BSTR Attribute :    attribute name (see table below)
[in] BSTR Value :        attribute value. (type according to attribute)
```

Example

```
nodeeval.AttValue(„FILE“)= FALSE
```

Attribute outline

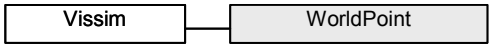
R	W	Attribute	Description
✓	✓	FILE	Write evaluation file flag (true/false)

i

Setting the FILE attribute to FALSE avoids writing the *.KNA evaluation file while collecting evaluation data (if node evaluation is activated) for possible online requests. This attribute is specific to the COM interface and is global to the VISSIM program (i.e. it will preserve its state independently from the loaded *.INP file).

2.63.2 Methods of the Interface INodeEvaluation

2.64 WorldPoint



This object defines the very general type world point which can be used as a parameter and/or returned as a result within some methods. As all objects of the VISSIM COM interface, this object must be created using the interface IVissim (see method NewWorldPoint) when needed as a parameter.

Examples

```
DIM wp AS WorldPoint
SET wp = vissim.NewWorldPoint(100.0, 100.0, 100.0)
SET so = vissim.Net.StatictsObjects.GetStaticObjectByCoord(wp)

DIM veh AS Vehicle
DIM pos AS WorldPoint
SET veh = vissim.Net.Vehicles.GetVehiclesByNumber(1)
SET pos = veh.AttValue("POINT")
x = pos.X
y = pos.Y
z = pos.Z
```

2.64.1 Properties of the Interface INodeEvaluation

X ([out, retval] double *pX)

Gets the X coordinate of the world point (x, y, z).

Parameters

```
[out, retval] double *pX :       returned x coordinate.
```

X ([in] double X)

Sets the X coordinate of the world point (x, y, z).

Parameters

```
[in] double X :  new x coordinate.
```

Y ([out, retval] double *pY)

Gets the Y coordinate of the world point (x, y, z).

Parameters

```
[out, retval] double *pY :       returned y coordinate.
```

Y ([in] double Y)

Sets the Y coordinate of the world point (x, y, z).

Parameters

```
[in] double Y :  new y coordinate.
```

Z ([out, retval] double *pZ)

Gets the Z coordinate of the world point (x, y, z).

Parameters

```
[out, retval] double *pZ :       returned z coordinate.
```

Z ([in] double Z)

Sets the Z coordinate of the world point (x, y, z).

Parameters

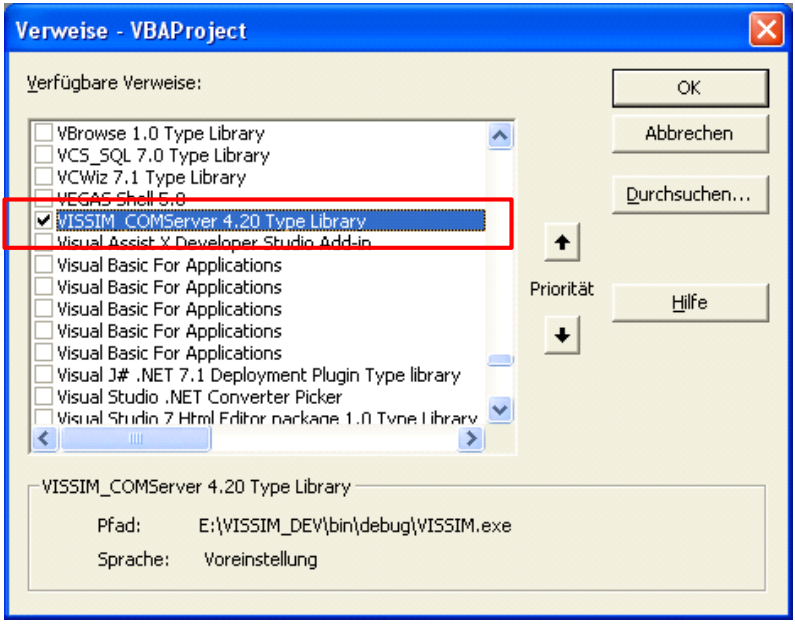
```
[in] double Z :  new z coordinate.
```

3 COM Access Using Visual Basic

All VISSIM COM interfaces support automation – they are based on the Microsoft standard interface IDispatch. Automation interfaces disclose the COM objects allowing the use of generic programming environments like Visual Basic or Script languages like Visual Basic Script (VBScript) or Java Script (JScript). This chapter introduces the use of Visual Basic to create client applications.

3.1 Creation of a Visual Basic Client

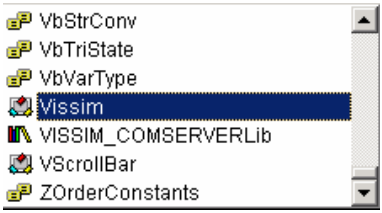
To make VISSIM objects available from Visual Basic code, set a reference to the VISSIM COM server type library:



With an active reference to the COM type information, Visual Basic is able to list the possible object declarations and method calls during code editing:

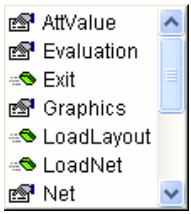
```
Dim vissim As Vissim
Dim net As Net
```

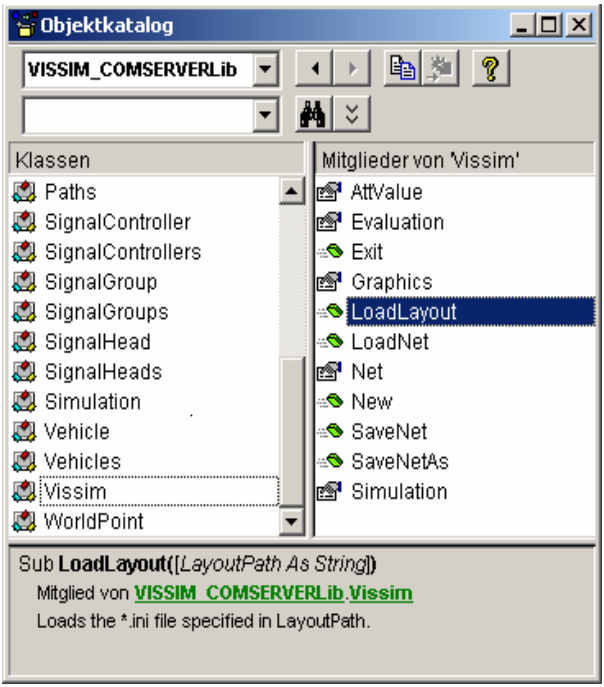
Identifying the class name also provides Auto List Member functionality. Each time you enter the object variable name, you get the drop-down list of the properties and methods exposed by the object:



Vissim

Furthermore, the object catalog enables you to browse through all available objects, interface properties and methods of the VISSIM COM server library. It also offers a short description of every method and property of the VISSIM COM server interfaces:




Objektkatalog
VISSIM_COMSERVERLib
Klassen
Paths
SignalController
SignalControllers
SignalGroup
SignalGroups
SignalHead
SignalHeads
Simulation
Vehicle
Vehicles
Vissim
WorldPoint
Mitglieder von 'Vissim'
AttValue
Evaluation
Exit
Graphics
LoadLayout
LoadNet
Net
New
SaveNet
SaveNetAs
Simulation
Sub LoadLayout([LayoutPath As String])
Mitglied von VISSIM_COMSERVERLib.Vissim
Loads the *.ini file specified in LayoutPath.

3.2 Collections (Different Ways to Enumerate)

Visual Basic has a special language constructor to handle collections:

```
FOR EACH element IN collection
' do something with the element
NEXT element
```

In order to use this kind of syntax to loop over collections, a specific COM standard interface must be supported (IEnumVARIANT). The VISSIM objects named with plural (like Nodes, Paths, Vehicles, …) are collection objects that implement this functionality. These objects can be treated as enumerations and allow to handle all the objects they contain sequentially: For example, the following Visual Basic code will loop over all link objects of the collection links:

```
DIM links AS Links
DIM link AS Link
SET links = vissim.Net.Links
FOR EACH link IN links
WITH link
    nr        = .ID
    name  = .Name
    length = .AttValue ("LENGTH")
END WITH
NEXT link
```

The number of elements of a VISSIM object collection can be requested with the method Count:

```
Dim number As Long
number = links.Count
```

3.3 Arrays

Another language element for object sequences is the array. In the context of the VISSIM COM server an array is a fixed n-dimensional sequence of indexed elements. These elements can be VISSIM objects or Visual Basic types and are encapsulated by a VARIANT type. The array type returned for some of the VISSIM object methods conforms to automation and allows the use of the Visual Basic array functionality as LBOUND and UBOUND to ensure safe access within the array limits. To know which method returns or needs an array as a parameter, refer to the respective specification in this document. For example: to access to the link geometry the AttValue together with the named attribute “POINTS” can be used. This attribute returns an array of world xy-coordinates:

```
DIM polyline()   ´array of VARIANT
polyline = link.AttValue(„POINTS“)
FOR i = LBOUND(polyline) TO UBOUND(polyline)
    x = polyline (i).X
    y = polyline (i).Y
NEXT i
```

Results from a data collection point (like speed or accelerations) will be returned in a two dimensional array:

```
res = datacollection.GetResult(„ACCELERATION“, „FREQUENCIES“)  '2-dim array
FOR i = LBOUND(res) TO UPBOUND(res)
    from = res(i, 0)
    to     = res(i, 1)
    val   = res(i, 2)
NEXT i
```

Methods like AddPathAsNodeSequence from the IPaths interface, need an array as a parameter. In this case the use of the Visual Basic function Array is suggested, which creates an array as a VARIANT from a sequence of given parameters:

```
DIM path AS Path
SET path = paths.AddPathAsNodeSequence (1, Array(10, 20, 30, 40))
```

3.4 Error Handling

When an interface returns an error, the intrinsic object “Err” with global scope is created and can be used within the exception handling mechanism of Visual Basic: On Error GoTo. If you don't use an On Error GoTo statement, any run-time error that occurs is fatal; that is, an error message is displayed and execution stops. The following example shows a way to handle errors:

```
DIM net as Net
DIM links AS Links
SUB Load_Click()
    SET net = vissim.Net
    SET links = net.Nodes
    ON ERROR GOTO exception  ' an Err object is created and continues in exception

    NodesListbox.Clear
    FOR EACH link IN links
    NodesListbox.AddItem link.ID
    NEXT link

    EXIT SUB  'subroutine ends here

    exception:
    MsgBox Err.Description  ' use here the created Err object
END SUB
```

The error-handling routine – a section of code marked by a label, here “exception” – should test or save relevant property values of the Err object before any other error can occur or before a procedure that might cause an error is called. The property values of the Err object reflect only the most recent error. The error message associated with the property Err.Number is contained in property Err.Description. See the annex on page 232 for a summary of all error messages of the VISSIM server.

Other possible error handling mechanisms of Visual Basic are the On Error Resume Next and On Error Goto 0. Refer to the VB manual for more details.

3.5 A Visual Basic Client Example

The following simple VB client shows how to run several simulations with different random seeds:

```
'declare some VISSIM COM types
DIM vissim AS Vissim
DIM simulation AS Simulation
Sub Main()
    ON ERROR GOTO exception
    'start VISSIM and create an instance of a Vissim object
    SET vissim = NEW Vissim
    'load a network
    vissim.LoadNet App.Path + "\3path.inp"
    'run a simulation
    DIM seeds AS VARIANT
    seeds = Array(22, 34, 56, 11)
    SET simulation = vissim.Simulation
    FOR i = LBOUND(seeds) TO UBOUND(seeds)
    simulation.RandomSeed = seeds(i)
    simulation.RunContinuous
    NEXT i
    vissim.Exit
    EXIT SUB
    exception:
    MsgBox Err.Description  ' use here the create Err object
END SUB
```

It is possible to collect results of every simulation like path evaluations using the method RunIndex of the ISimulation interface. This method allows you to index the evaluation files of each run. It is suggested to configure an *.ini file with the desired options and to load it with the method LoadLayout of the IVissim interface before starting the iteration procedure.

3.6 Advanced Issues Using Visual Basic

Creating the Vissim class object

Internally, VB has two mechanisms for creating objects: VB can use its own object creation services, or it can use the services of the Component Object Model (COM). VB selects the services depending on the location of the class and on the code used to create the object. There are three coding techniques for creating a Vissim object. You can create it explicitly using the NEW keyword with the SET statement as seen on the examples above. You can create it implicitly using the NEW keyword on the declaration or you can use the CreateObject function:

```
' 1) explicit creation through VB internal creation services
DIM vissim AS Vissim
SET vissim = NEW Vissim

' 2) implicit creation through VB internal creation services
DIM vissim AS NEW Vissim
Vissim.LoadNet("example.inp")

' 3) creation through the register (no type library reference in VB is needed)
DIM v AS Object
SET v = CreateObject("VISSIM.Vissim")
```

VB uses its own object services (and the referenced type library information) when you use the NEW keyword either on the declaration or in the SET statement. VB uses COM (with the register information) if you use the CreateObject function (a reference to the VISSIM COM server type library is not necessary in this case). This distinction can make a difference in how you can develop your application and how it executes.

Furthermore it is possible to create an instance of a Vissim object class remotely using the optional parameter of CreateObject:

```
 ' 3) creation through the register information of a remote machine
DIM v AS Object
SET v = CreateObject("VISSIM.Vissim", "MyServer")  ' name or IP address
```

Please refer to the remote access chapter on page 229 for details.

Early-Bound versus Late-Bound

When you call a property or method on an object, VB must look up the property or method, place the arguments on the stack, invoke the property or method, and return. Early binding implies VB can look up the properties and methods of an object at compile time. It can then store the internal location of the property or method in the compiled code. When VB calls a method at run time, it can execute the method directly. Late binding means VB can't look up an object's properties and methods until run time, this lookup means another cross process call, thereby doubling the call overhead.

How you declare the object, not how you create the object, determines whether the properties and methods of the object are early-bound or late-

bound. For early binding, you identify the name of the class on the declaration:

```
DIM vissim AS Vissim
```

This allows VB to perform the binding at compile time. For late binding, you simply identify an object:

```
DIM vissim AS Object
```

In this case the object is identified at run time. VB must then look up each property and method for that particular object. This means there's a greater chance of runtime errors, because VB can't determine the list of properties and methods exposed by the object until run time.

When using early binding a reference to the VISSIM COM server type library must be set within the VB programming environment. This can cause some conflicts when installing newer versions of VISSIM, which could have modified the type library. Unselecting and selecting again the reference to the VISSIM COM server forces VB to reinterpret the type library. Please refer to the annex "Tips and hints" on page 239 for details about VISSIM versions and its COM server interface.

4 COM Access Using Visual C++

The programming a COM client in C++ alters according to platform, compiler and library. The initialization and declarations for using the COM functionality and components suggested here are focused on the Microsoft Visual C++ compiler and its standard COM library (ole32.lib). Other less specific alternatives will be also pointed to in the course of this chapter.

4.1 Creation of a VC++ Client

Client applications must initialize the COM library before they can call COM library functions. This can be done with the functions CoInitialize() or CoInitializeEx(). The respective end of initialization function CoUninitialize() must be called on application shutdown; it will release automatically all created COM objects. The preprocessor directive _WIN32_DCOM must be defined and the header file <objbase.h> must be included in order to use these functions.

```
#define _WIN32_DCOM
#include <iostream>
using namespace std;
#include <objbase.h>
int main()
{
  cout << "Client: Calling CoInitialize()" << endl;
  CoInitializeEx(NULL, COINIT_MULTITHREADED);
  // do something
  cout << "Client: Calling CoUninitialize()" << endl;
  CoUninitialize ();
  return 0;
}
```

With the Visual C++ compiler you can use the #import preprocessor directive to make VISSIM objects available for Visual C++ code:

```
// modify path to your needs
#import "c:\Programs\ptv-vision\VISSIM400\exe\vissim.exe"
using namespace VISSIM_COMSERVERLib;
```

You can use the directive option no_namespace if you don't want to use a namespace for the VISSIM COM server. Alternatively you can import the supplied TLB library VISSIM_COMServer.tlb with the same effect. The application of this directive allows the direct use of the Microsoft specific smart pointers when working with COM objects (see the compiler-generated Primary Type Library Header File, *.tlh, for details on the declarations). Smart pointers offer a straight access to the VISSIM COM server objects; the following code line creates an instance of a Vissim object:

```
IVissimPtr spVissim(__uuidof(Vissim));
```

The smart-pointer IVissimPtr is declared in the generated *.tlh file using the template _com_ptr_t. This implementation encapsulates interface pointers and eliminates the need of keeping track of the reference count calling AddRef() and Release() functions. In addition, it hides the CoCreateInstance() call.

Another possibility is the use of the CreateInstance() method:

```
HRESULT hr;
IVissimPtr spVissim;
hr = spVissim.CreateInstance("VISSIM.Vissim", NULL, CLSCTX_LOCAL_SERVER);
```

The directive #import also facilitates usage of the interface methods and properties by declaring appropriate wrapping inline member methods for every interface (see the compiler-generated Secondary Type Library Header File, *.tli, for details on the declarations). Once a VISSIM instance is created

the Net and Simulation objects can be accessed through the IVissim interface:

```
INetPtr spNet;
spNet = spVissim->GetNet();
ISimulationPtr spSim;
spSim = spVissim->GetSimulation();
double period = spSim->GetPeriod();
```

When using other compilers than Microsoft Visual C++ a more standard way of using COM components must be applied. The supplied files VISSIM_COMServer.h and VISSIM_COMServer_i.c contain the standard C++ code declarations for the VISSIM server interfaces. Including these to files has a similar effect like the VC++ #import preprocessor directive:

```
#include "VISSIM_COMServer.h"
#include "VISSIM_COMServer_i.c"
```

In this case it is necessary to work with C++ pointers and to use the available functions of the COM technology implementation. In the case of Microsoft, the objbase.h file must be included and the use of CoCreateInstance allows the first instantiation of the Vissim object:

```
HRESULT hr;
IVissim* pVissim;
hr = CoCreateInstance(CLSID_Vissim, NULL,
CLSCTX_LOCAL_SERVER, IID_IVissim, (void**)&pVissim);
if (FAILED(hr)) exit;
```

The access to the methods and properties must use the functions declarated in the header file: VISSIM_COMServer.h:

```
HRESULT hr;
INet* pNet;
hr = pVissim->get_Net(&pNet);
if (FAILED(hr)) exit;
ISimulation* pSim;
hr = pVissim->get_Simulation(&pSim);
if (FAILED(hr)) exit;
double period;
hr = pSim->get_Period(&period);
if (FAILED(hr)) exit;
```

4.2 Collections (Different Ways to Enumerate)

As in Visual Basic it is possible to iterate through a collection in different ways: 1) using the IEnumVARIANT interface, supported for all VISSIM COM server enumeration objects, and 2) using the Item() method.

While the use of the of the _NewEnum() method is implicit in VB, in C++ it must be called explicitly. But thanks to the combination of template classes that allow to cast smart pointers instead of having to call QueryInterface(), the code just takes one line:

```
IEnumVARIANTPtr spEnum(spLinks->Get_NewEnum());
```

The smart-pointer spEnum points to the enumeration object of the collection and offers the method Next() to iterate over its elements:

```
long id;
unsigned long ulFetched;
_variant_t var;
spEnum->Next(1, &var, &ulFetched);
while (ulFetched == 1) {
id = ((ILinkPtr)var.pdispVal)->GetID();
spEnum->Next(1, &var, &ulFetched);
}
```

a more efficient form is calling the method Next() just once for an array:

```
long id, count = spLinks->GetCount();
unsigned long ulFetched;
_variant_t* avar= new _variant_t[count];
spEnum->Next(count, avar, &ulFetched);
for (int i = 0; i < count; i++) {
id = ((ILinkPtr)avar[i].pdispVal)->GetID();
}
```

and alternatively it is possible to iterate as in VB, using the Item() method:

```
long jCount = spLinks->GetCount();
_variant_t jvar = 1L;
for (long j = 1; j <= jCount; j++, jvar = j) {
id = spLinks->GetItem(jvar)->GetID();
}
```

4.3 Arrays

The Automation type for arrays is the SAFEARRAY. A SAFEARRAY is a self-descriptive, multidimensional vector type. It was introduced originally by Visual Basic and therefore this type is identical to the Visusal Basic array. In C++ it is necessary some help to deal with them. We purpose here two possibilites to deal with SAFEARRAYS with C++: 1) using API functions and 2) using the Active Template Library ATL wrappers (starting at version ATL 7.0).

The use of the API function SafeArrayCreate() is somehow more tediuos and implies more lines of code. For example to create an arry containing

```
long ids[] = {1, 2, 5, 11, 3, 7};
VARIANT var;
VariantInit(&var);
SAFEARRAY *psa;
SAFEARRAYBOUND rgsabound[1];
long rgindice[1];
rgsabound[0].lLbound = 0;
rgsabound[0].cElements = sizeof ids / sizeof *ids;
psa = SafeArrayCreate(VT_VARIANT, 1, rgsabound);
for (long i = 0; i < sizeof ids / sizeof *ids; i++) {
  rgindice[0] = i;
  var.lVal = ids[i];
  var.vt = VT_I4;
  SafeArrayPutElement(psa, rgindice, &var);
}
var.parray = psa;
var.vt = VT_ARRAY | VT_VARIANT;
spPaths->AddPathAsNodeSequence(1, var);
```

The available SAFEARRAY wrapper classe introduced in ATL7 CComSafeArray<T>, simplify the construction and access:

```
CComSafeArray<VARIANT> array(sizeof ids / sizeof *ids);
for (long i = 0; i < sizeof ids / sizeof *ids; i++) {
  array.SetAt(i, CComVariant(ids[i]));
}
spPaths->AddPathAsNodeSequence(1, CComVariant(array));
```

4.4 Error Handling

The wrapping functions declared in the Secondary Type Library Header File (*.tli) check the availability of an error after calling the interface's virtual functions, convert it into a _com_error (a class that encapsulates the COM error object) and throw it. Therefore, when using the #import preprocessor directive, error handling is limited to the catch of _com_error exceptions:

```
try {
IVissimPtr spVissim(__uuidof(Vissim));
ISimulationPtr pSim;
pSim = pVissim->GetSimulation();
}
catch (_com_error &error) {
cout << error.Description() << endl;
}
```

The Microsoft specific error-handling wrapper class _com_error encapsulates the HRESULT and offers the possibility to get more precise error information (if an ErrorInfo object is associated to the error, which will be the case for all VISSIM VOM server objects).

When you do not use the #import directive you must check the returned HRESULT and access the IErrorInfo manually:

```
HRESULT hr;
INet* pNet;
hr = pVissim->get_Net(&pNet);
If (FAILED(hr)) {
BSTR description = 0;
IErrorInfo* pEI;
GetErrorInfo (0, & pEI);
pEI->GetDescription (&description);
}
```

4.5 A Visual C++ Client Example

The following simple VC++ client shows how to run several simulations with different random seeds:

```
#define _WIN32_DCOM
#include <iostream>
using namespace std;
// import of all VISSIM COM server interfaces // modify path to your needs
#import "G:\VISSIM\Exes\371#\vissim.exe"
using namespace VISSIM_COMSERVERLib;
int main()
{
    HRESULT hr;

    cout << "Client: Calling CoInitialize()" << endl;
    hr = CoInitialize(NULL);
    if (FAILED(hr)) return 1;
    try {
        // Create the Vissim object (connection to ISSIM COM server)
        cout << "Client: Creating a Vissim instance with the smartpointer IVissimPtr" <<
endl;
        IVissimPtr spVissim(__uuidof(Vissim));
        // Create a Simulation object
        cout << "Client: Calling GetSimulation()" << endl;
        ISimulationPtr spSim;
        spSim = spVissim->GetSimulation();
        // read network and layout
        char buf[255] = "";
        GetCurrentDirectory (sizeof(buf), buf);
        _bstr_t path(buf);
        spVissim->LoadNet (path + "\\fixed_time.inp", 0);
        spVissim->LoadLayout (path + "\\link_eval.ini");
        // initialize simulations
        cout << "Client: Setting simulations" << endl;
        long  seeds[4] = {10, 20, 30, 42};
        spSim->PutPeriod(100);
        spSim->PutResolution(1);
        spSim->PutSpeed(5);
        // run simulations
        cout << "Client: Starting simulations:" << endl;
        for (int index = 0; index < 4; index++) {
            cout << "  Client: Initializating simulation " << index+1 << endl;
            spSim->PutRunIndex(index);
            spSim->PutRandomSeed(seeds[index]);
            spSim->PutComment("Random seed =" + _bstr_t(ltoa(seeds[index], buf, 10)));
            cout << "  Client: Running simulation " << index+1 << " ..." << endl;
            spSim->RunContinuous();
        }
    }
    catch (_com_error &error) {
        cout << (char*)(error.Description()) << endl;
    }
  cout << "Client: Calling CoUninitialize()" << endl;
  CoUninitialize ();
  return 0;
}
```

5 COM Access with .NET

There is a two-way interoperability between COM and .NET technology. Because of that, the use of the VISSIM COM Server from a .NET based client is almost automatic. Almost, because it is necessary the generation of an assembly that matches the metadata of the VISSIM COM Server type library.

There are four ways to get the respective assembly:

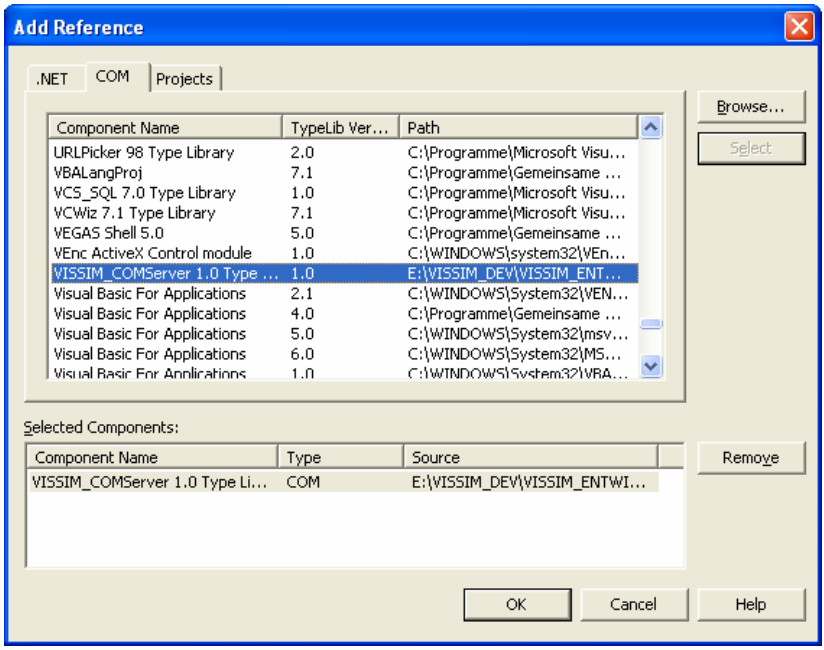
Using Visual Studio .NET	Automatically converts COM types in a type library to metadata in an assembly.
Using the Type Library Importer tool (tlbimp.exe)	Provides command-line switches to adjust metadata in the resulting file, imports types from an existing type library, and generates an assembly and a namespace.
Using the TypeLibraryConverter class	Exposes methods that perform conversion-related actions.
Programming a customized wrapper.	As a less desirable option, you can create type definitions from scratch.

If you are using Visual Studio .NET to generate a .NET client we recommend the first possibility. If not, the use of .NET Framework tool TLBIMP.EXE. The other two possibilities are required for the use of the VISSIM CM Server.

5.1 Creation of a Client Using Visual Studio .NET

Independently of the chosen programming language, when using Visual Studio .NET it is necessary to add a reference to the VISSIM COM Server by:

1. From the Project menu, select References.
2. Select the COM tab.
3. Select the VISSIMCOM_Server library from the references list
4. Click OK.



This will automatically create the wrapper assembly Interop.VISSIM_COMSERVERLib.dll (Interop stands for the interoperability between COM and .NET) that allows the access to the objects and interfaces of the VISSIM COM Server through the default namespace VISSIM_COMSERVERLib.

Interfaces, normally prefixed with an ”I”, get an additional name without prefix (for example, VISSIM_COMSERVERLib::ISimulation and VISSIM_COMSERVERLib:: Simulation, are allowed to refer for ISimulation interface).

COM Objects are suffixed with “Class”. In our case this is only relevant for the Vissim object, which must be referred as VISSIM_COMSERVERLib::VissimClass.

Example with Visual Basic .NET

```
Dim vis As VISSIM_COMSERVERLib.Vissim
Dim sim As VISSIM_COMSERVERLib.Simulation

vis = New VISSIM_COMSERVERLib.VissimClass

vis.LoadNet("example.inp")
sim = vis.Simulation
sim.RunContinuous()
```


Please, refer to the chapter “Upgrading from Visual Basic 6.0” of the MSDN for a description of the differences between the Visual Basic and Visual Basic .NET.

Example with Visual C++.NET

```
#using <mscorlib.dll>
using namespace VISSIM_COMSERVERLib;
int main()
{
  Vissim  *vis = new VissimClass;
  vis->LoadNet("example.inp", false);
  Simulation *sim = vis->Simulation;
  sim->RunContinuous();
  return 0;
}
```

Example with Visual C#

```
using System;

using VISSIM_COMSERVERLib;

public class Client
{
  public static void Main()
  {
    Vissim vis = new VissimClass();
    vis.LoadNet("E:\\VISSIM\\COM\\VBA\\GROSSFKT.INP", 0);

    Simulation sim = vis.Simulation;
    sim.RunContinuous();

    System.Console.WriteLine("Hello, World!");
  }
}
```

Example with Visual J#

```
package VissimClient;
import VISSIM_COMSERVERLib.*;
public class Client
{
  public static void main()
  {
    Vissim vis = new VissimClass();
```

```
    vis.LoadNet("E:\\VISSIM\\COM\\VBA\\GROSSFKT.INP", (ubyte)0);
    Simulation sim = vis.get_Simulation();
    sim.RunContinuous();
  }
}
```


Please, refer to the chapter “Upgrading from Visual J++ 6.0” of the MSDN for a description of the differences between the Visual J++ and Visual J#.

5.2 Arrays

The corresponding .NET Framework built-in type for the COM Automation type VARIANT is the System.Object type. Because array in VISSIM are passed as VARIANTs and the managed arrays of the .NET Framework are System.Objects, we can use directly the managed array operator __gc[ ] to construct array for VISSIM. Methods like AddPathAsNodeSequence() from the IPaths interface, need and array of VARIANTs therefore we need also to encapsulate the values on System.Objects. For this purpose we use the keyword __box:

```
System::Object* array __gc[] = {__box(1), __box(2), __box(5),  __box(7)};
pPaths->AddPathAsNodeSequence(1, array);
```

The variable array is here a managed array of System.Objects which will be passed as an Object itself and marshaled by default into a VARIANT.

Similarly in Visual Basic .Net it would be:

```
Dim array() As Object = {1, 2, 5, 11, 3, 7}
paths.AddPathAsNodeSequence(1, array)
```

5.3 Events

Some objects of the VISSIM COM Server (see the description of the DynmamicAssignment on page 39 for an example) sent events to the client. To consume an event in a client application, you must provide an event handler (an event-handling method) that executes program logic in response to the event and register the event handler with the event source. The WithEvents statement and the Handles clause provide a declarative way of specifying event handlers. Events raised by an object declared with the WithEvents keyword can be handled by any procedure with a Handles statement for that event. The following Visual Basic .NET code illustrates how to do this, taking the example of the DynamicAssignment object and its event EdgeSmoothing:

```
Dim WithEvents dyn As DynamicAssignemnt
Private Function dyn_EventHandler(ByVal a As Double, ByVal b As Double) As Double Handles
dyn.EdgeSmoothing
Return (1.0# - 1.0# / nIt) * a + 1.0# / nIt * b
End Function
```

The EdgeSmoothing events raised by the obejct dyn will be handled by the function dyn_EventHandler.

5.4 Error Handling

COM methods report errors by returning HRESULTs; .NET methods report them by throwing exceptions. The runtime automatically maps the HRESULT from COM interop to more specific exceptions. For example, E_OUTOFMEMORY becomes OutOfMemoryException. If the HRESULT is a custom result or if it is unknown to the runtime, the runtime passes a generic COMException to the client. The ErrorCode property of the COMException contains the HRESULT value. The in C++ could be:

```
try {
  vis->LoadNet("example.inp", false);
}
catch (System::Exception* pEx) {
  System::Console::WriteLine(S"Generic Exception Handler: {0}", pEx->Message);
}
```

and in Visual Basic .NET:

```
Try
  vis.LoadNet("example.inp")
Catch ex As Exception
  MsgBox(ex.Message)
End Try
```

6 COM Access Using Java

Similarly to the use of the use of .NET, the use of Java is very homogeneous. The objects and interfaces of the VISSIM COM Server type library are made available to a Java client by generating a package with wrapper classes for all entities present in the type library. This package can be used by any Java code, typically with a import statement to make possible the use of abbreviated names for the classes.

6.1 Creation of a COM Wrapper

A large list of tools to create Java-COM bridges, commercials and not commercials, are available (Jawin, Jace, JACOB, R-JAX, jacoZoom, jactivex, J-Integra, JunC++ion, JCOM, ...). Depending on the concrete platform and software environment, a tool will be preferred to another. The support of COM Automation types must be guaranteed. For example, a map of the COM VARIANT type into a java type, e.g. java.lang.Object type and a VARIANT containing an array maps to a Java array. If you are looking for a Java-COM bridge that works for any Java VM on any platform, including UNIX, please check out the J-Integra product.

If using Microsoft Java Virtual Machine Support (MSJVM) and the tool jactivex, please consider the notes on http://www.microsoft.com/mscorp/java/. The MSJVM will reach its end of life on December 31, 2007.

An alternative ist the use of a software solution for bridging Java and .NET. Consider the way interoperability between COM and .NET technology (see chapter on using COM eith .NET on page 217) and the use of the VISSIMCOMServer assembly.

6.2 Creation of a Java client

Once the COM wrapper classes has been created, Java can use the VISSIM objects and interfaces exactly as if they were native Java objects and interfaces. The wrapper classes have the same names as the corresponding VISSIM COM Server entities:

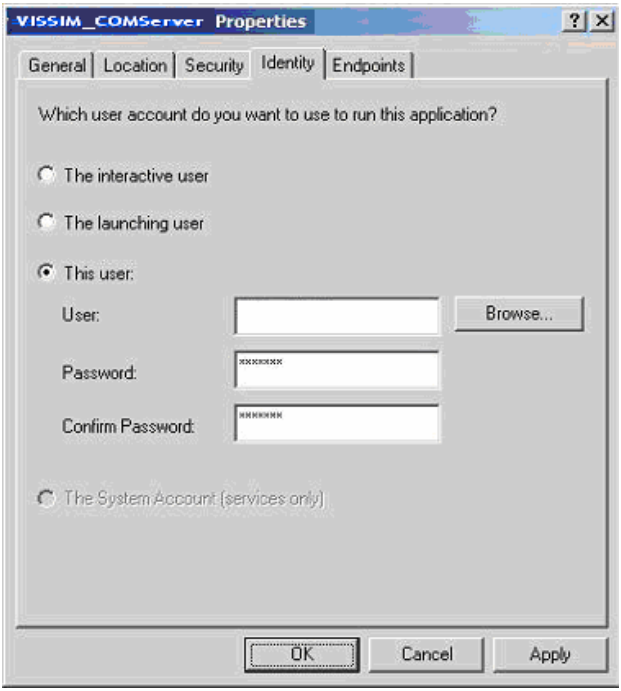
```
import VISSIM.VISSIM_COMSERVER.*;
class VissimSim
{
  private static IVissim vis;
  private static ISimulation sim;
  public static void main ( String[] args )
  {
    try {
      vis = new Vissim();
      sim = vis.getSimulation();
      sim.runContinuous();
    }
    catch (Exception e) {
      e.printStackTrace();
    }
  }
}
```

Please refer to the documentation of the used Java-COM bridge for the use of Automations types.

7 COM Remote Access

Some issues must be taken in account when accessing the VISSIM COM Server from a client on a remote machine. First of all, VISSIM must be registered on the server machine as usual (see page 13 about registration). Second, launch and access permissions must be properly set in order to allow the client's identification to run and use the registered VISSIM COM server.

In order to configure the security matters the DCOMCNFG.EXE is suggested (please refer to the Windows documentation for the use of this tool). This tool allows mainly to set properly the necessary permissions flags for the authentication level, the launch permission and the access permission. It also allows you to select the identity (or the RunAs register) for the server process 1) interactive user, 2) launching user and 3) a specific user:



If you choose interactive user, you're commanding COM to assign the server process the identity of the person logged in at the server console -the "interactive user"- when a launch occurs. Interactive is the only option that permits to start VISSIM normally with its GUI. But it can't be launched if there's no one logged in on the server -that is, if there is no interactive user.

The launching user simply tells COM to assign the server process the identity of the remote client process that did the launching. This enables the process to launch even if no interactive user is logged in, which means that the VISSIM will work in underground mode without a GUI.

The "This user" option sets up a special user account for each server to run under. If a COM server is configured to run as Vissim Object, and if Vissim Object is a valid account on the server, then when launched, the remote server process is assigned the identity of Vissim Object. It matters not who's logged in on the server (or if anyone is logged in at all) or who launched the process.

8 Annexes

8.1 Error Messages

All possible errors that can occur during the use of the VISSIM COM Server are classified in four categories:

1. Windows errors: Low level exceptions generated by the system (wrong memory handling, division by zero, etc). It is not possible to recover the COM server when such an error occurs. These errors must be reported and fixed by the VISSIM development team. Please, contact the VISSIM hotline.
2. COM errors: Exception generated by one of the COM support classes and libraries (e.g.: the COM object Vissim couldn't be instantiated). A common COM error occurs when using a previously working client, having installed a new version of VISSIM. Refer to the Tips and Hints annex on page 239 to solve this problem. Other probably causes of these kind of errors are corrupted registering of VISSIM or user system rights. Please ensure that VISSIM has been installed correctly (refer to 13) and with administration rights and that all register entries are available (refer to annex for a detailed list of the register information). If the error persists or you are not aware of the system register function, please contact the VISSIM hotline.
3. VISSIM errors: Exception generated by internal VISSIM procedures (e.g.: a simulation couldn't write the required results). These kinds of errors can be of very different nature, depending on the context and the concrete used interface and method. Please, refer to the list of messages below to handle or workaround adequately the error. Another possibility is the use of the interface on a not expected but valid way; this could imply a desired enhancement for future versions. Please contact the VISSIM hotline to report the concrete case.
4. User errors: Errors generated by the wrong use of the COM interfaces (e.g.: a wrong node sequence is used for the creation of a new path). Again, these kind of errors can be of very different nature, depending on the context and the concrete used interface and method. Please, refer to the list of messages below to handle or workaround adequately the error.

COM error messages	Description
An unspecified error has occurred in COM.	General message if no more concrete information is available.
An error has occurred while initializing the VISSIM COM server. Error <number> returned.	The COM server couldn't be initialized. VISSIM will work without supporting the COM functionality. This error should never show up, please contact the VISSIM hotline.
Access denied while registering the COM server.	VISSIM couldn't set up register with the entries for the COM server interfaces, because of system access rights. VISSIM will work without supporting the COM

COM error messages	Description
	functionality. Please, be sure you have he user rights for accessing (reading and writing) the system register.
An error has occurred while registering the VISSIM COM server. Error <number> returned.	VISSIM couldn't set up register with the entries for the COM server interfaces, because of unknown reasons. VISSIM will work without supporting the COM functionality. Please, be sure you have he user rights for accessing (reading and writing) the system register.
Access denied while unregistering the COM server.	VISSIM couldn't set up remove the register entries for the COM server interfaces, because of system access rights.
An error has occurred while unregistering the VISSIM COM server. Error <number> returned.	VISSIM couldn't set up remove the register entries for the COM server interfaces, because unknown reasons.
An error has occurred while creating an object.	It has not been possible to create an instance of a requested object. A possible cause is the use of a previously working client with a new installed VISSIM version (refer to the Tips and Hints annex on page 239).
An error has occurred while binding an object.	Internal error. The requested object couldn't be bound to the VISSIM data. The causes of this error are of internal nature a can neither be solved by the user nor have a possible workaround. Please report it to the VISSIM hotline.
Not implemented method called.	The interface method is not supported. Possibly the method is still not implemented or the concrete object instance don't support it.

VISSIM error messages	Description
An unspecified error has occurred in VISSIM.	General message if no more concrete information is available. Please contact to the VISSIM hotline.
An error has occurred dispatching the VISSIM event <number>.	VISSIM couldn't server a requested event from a COM call. There is any concrete cause for this error. Please report it to the VISSIM hotline.
The object link to the	The object on the client side is no longer valid

VISSIM error messages	Description
VISSIM data is no longer valid.	in VISSIM. The most common case for this error is the Vehicle objects, which are available only during a time frame of the simulation period. Trying to access them outside this time frame, once they have been previously instantiated and accessed successfully, produce this error. Refer to the notes in the IVehicle interface description on this document for more information. Another possibility is when the data (network element, e.g. link, data collection ...) has been removed manually from the network editor.
The file couldn't be loaded.	It is not possible to open or read the requested file. Please be sure of the existence, format and access rights to the file.
The file couldn't be saved.	It is not possible to write the requested file. Please be sure that you have the writing accesses and that the hard drive is not full.
No valid time text.	The passed string format couldn't be interpreted to a valid time. Valid formats are: [hh:mm:ss] or [h:mm:ss], [hh:mm.ss] or [h:mm.ss], [hh:mm] or [h:mm].
The simulation couldn't be initialized	Not possible to start a simulation. The reasons are of different nature. Please, try to start the simulation manually from the network editor and ensure that all necessary data for the requested simulation is available (e.g. configuration files for results, signal control files, etc.). Try also to save the layout (*.ini file) of the working simulation and load it within the IVissim interface with the LoadLayout() method.

User error messages	Description
An unspecified error has occurred.	General message if no more concrete information is available. Please contact to the VISSIM hotline.
A wrong data type has been passed as a parameter.	No valid data type has been passed to a method. Please, refer to the use description of the interface method in this document.
No valid directory.	The passed path string don't match a valid directory on the hard drive.

User error messages	Description
No valid file name	The passed name string doesn't represent any existing file.
Method call only possible during a simulation run.	Because of internal VISSIM simulation initializations some methods have a sense only in the context of a running simulation. It is possible to use them, using the method RunSingleStep() of the ISimulation interface. This error message occurs when calling them outside a simulation running.
The object with the specified identifier number doesn't exist.	When identifier numbers are needed as a parameter for the execution of a method, this error occurs if one of this identifiers doesn't represent a valid element. For example passing a no valid node number to the method AddPathAsNodeSequence()of the IPaths interface.
Identifier number overflow.	Because of the Automation conformance, the maximum identifier number using the COM interfaces is 2,147,483,647. Otherwise VISSIM allows to use identifier numbers up to 4,294,967,295. An overflow error occurs when using methods interchanging identifier numbers greater than 2,147,483,647.
Too large value.	To large value has been passed. Please, refer to the concrete interface method use description of this document.
Too low value.	To low value has been passed. Please, refer to the concrete interface method use description of this document.
An object with the same identifier number already exists.	The passed identifier number has been used before. Use a different identifier.
The specified attribute <attribute name> is unknown.	The passed string name doesn't match any valid attribute.
The specified attribute <attribute name> is not allowed.	The passed attribute is not valid for the method. Please, refer to the listed attributes on the interface method use description of this document.
The specified attribute <attribute name> is not readable.	The passed attribute is not valid to get data. Please, refer to the listed attributes on the interface method use description of this

User error messages	Description
	document.
The specified attribute <attribute name> is not writable.	The passed attribute is not valid to set data. Please, refer to the listed attributes on the interface method use description of this document.
The specified attribute <attribute name> requires one parameter.	An additional parameter is needed for the passed attribute. Please, refer to the listed attributes on the interface method use description of this document.
The specified attribute <attribute name> requires two parameters.	Two additional parameter is needed for the passed attribute. Please, refer to the listed attributes on the interface method use description of this document.
First passed parameter is not valid.	The first passed parameter can't be used. For example no valid lane number using the attribute LANECLOSED" of the ILink interface.
Second passed parameter is not valid.	The second passed parameter can't be used. For example no valid vehicle class number using the attribute LANECLOSED" of the ILink interface.
The passed value is not valid.	The passed value can't be used for setting the requested data. For example no valid desired speed fractil using the attribute DESSPEEDFRACTIL" of the IVehicle interface.
The specified parameter is unknown.	The passed string name doesn't match any valid named parameter.
The specified parameter is not allowed.	The passed named parameter is not valid for the method. Please, refer to the listed possible parameters on the interface method use description of this document (normally for evaluations results).
The specified function is unknown.	The passed string name doesn't match any valid named function.
The specified function is not allowed.	The passed named function is not valid for the method. Please, refer to the listed possible functions on the interface method use description of this document (normally for evaluations results).
The specified configuration is not defined within	Some methods for evaluations results need a previously configuration for data collection.

User error messages	Description
VISSIM.	This error occurs when requesting results that have not been previously configured. For example, using the GetSegmentResult() method of the ILink interface to request density results can end up with this error if the density has not been requested within the configuration.
The specified vehicle class doesn't exist.	Methods that require a vehicle class number can produce this error if a wrong identifier has been passed. Please, be sure that the vehicle class number exist in the used project.
A path requires at less 3 nodes.	Error message when passing an array with less than 3 node identifiers to the method AddPathAsNodeSequence() of the IPaths interface.
No valid node sequence.	It is not possible to build a path from the passed node sequence using the method AddPathAsNodeSequence() of the IPaths interface. Please, be sure that all passed nodes exist and are connected with links and connectors in the same order as the passed sequence.
No origin parking lot between the first two nodes.	A path must start on a parking lot. The existence of a parking lot in-between the first two nodes must be guaranteed.
No destination parking lot between the last two nodes.	A path must end up on a parking lot. The existence of a parking lot in-between the last two nodes must be guaranteed.

8.2 Warning Messages

Warning messages are written in the VISSIM trace file (with the same name as the input file and the extension *.ERR).

Warning messages	Description
The <network element> with the specified identifier number <number> doesn't exist.	When using the MultiAttValues() methods of object collections, identifier numbers must be passed as parameter. This warning is written in to the trace file for each not existing element.
The path <p>, starting from parking lot <plo>, can't be assigned to vehicle <v> parked in parking lot <pl>.	A path p was has been tried to be assigned to a vehicle parked on a different parking lot pl than the origin plo of the path.

8.3 Tips and Hints

Using Excel as client comes the warning "Microsoft Excel is waiting for another application to complete an OLE action"

A call to the VISSIM COM interface, like RunContinuous in IVissim, is taking more time than the expected. Excel indicates that it is waiting for VISSIM to finish. In order to disable this warning you can set the Excel property DisplayAlerts of the Application object to false.

The VISSIM COM server interface and the VISSIM versions

Because of the currently evolving state of the VISSIM COM server, the interface is subjected to constant enhancements. A direct consequence of interface changes is the no longer consistent type library information of earlier working clients. For this reason it is recommended either to code client with late binding when possible or to recompile the clients (or reset the reference to the VISSIM type library if it is the case of the programming environment) when a new VISSIM version has been installed. On a more complete state of the VISSIM COM server the need of this client actualization will not be longer necessary.

8.4 Registry

The Windows Registry plays a central role during the creation and use of COM objects. The VISSIM COM Server must be appropriately registered before a client can use some of its interfaces (see chapter 1.1 License and Registration on page 13 on how to do this registration).

Different types of information are registered in the system for the use of the VISSIM COM Server. The following detailed list of entries can be used to check the correctness of the registration, when necessary (For notational convenience HKEY_CLASSES_ROOT will be abbreviated to HKCR):

ProgID

```
[HKEY_CLASSES_ROOT\VISSIM.Vissim]
@="Vissim Class"
[HKEY_CLASSES_ROOT\VISSIM.Vissim\CLSID]
@="{D9A469E1-16AA-4b59-A051-96F2FD127D94}"
[HKEY_CLASSES_ROOT\VISSIM.Vissim\CurVer]
@="VISSIM.Vissim.420"
[HKEY_CLASSES_ROOT\VISSIM.Vissim.420]
@="Vissim Class"
[HKEY_CLASSES_ROOT\VISSIM.Vissim.420\CLSID]
@="{D9A469E1-16AA-4b59-A051-96F2FD127D94}"
```

AppID

```
[HKEY_CLASSES_ROOT\AppID\{79C4AA74-E6DB-4727-B5A9-2E6D16917263}]
@="VISSIM_COMServer"
[HKEY_CLASSES_ROOT\AppID\VISSIM_COMServer.EXE]
"AppID"="{79C4AA74-E6DB-4727-B5A9-2E6D16917263}"
```

Type library ID

```
[HKEY_CLASSES_ROOT\TypeLib\{DB7C8CF6-1D94-4aca-A4D7-35906E07AE15}]
[HKEY_CLASSES_ROOT\TypeLib\{DB7C8CF6-1D94-4aca-A4D7-35906E07AE15}\1.0]
@="VISSIM_COMServer 1.0 Type Library"
[HKEY_CLASSES_ROOT\TypeLib\{DB7C8CF6-1D94-4aca-A4D7-35906E07AE15}\1.0\0]
[HKEY_CLASSES_ROOT\TypeLib\{DB7C8CF6-1D94-4aca-A4D7-35906E07AE15}\1.0\0\win32]
@="C:\Programme\PTV_Vision\VISSIM\Exe\VISSIM.exe"
[HKEY_CLASSES_ROOT\TypeLib\{DB7C8CF6-1D94-4aca-A4D7-35906E07AE15}\1.0\FLAGS]
@="0"
[HKEY_CLASSES_ROOT\TypeLib\{DB7C8CF6-1D94-4aca-A4D7-35906E07AE15}\1.0\HELPDIR]
@="C:\Programme\PTV_Vision\VISSIM\Exe\VISSIM.exe"
```

VISSIM class object ID

```
[HKCR\CLSID\{D9A469E1-16AA-4b59-A051-96F2FD127D94}]
@="Vissim Class"
"AppID"="{79C4AA74-E6DB-4727-B5A9-2E6D16917263}"
[HKCR\CLSID\{D9A469E1-16AA-4b59-A051-96F2FD127D94}\LocalServer32]
@="C:\Programme\PTV_Vision\VISSIM\Exe\VISSIM.exe"
 [HKCR\CLSID\{D9A469E1-16AA-4b59-A051-96F2FD127D94}\ProgID]
@="VISSIM.Vissim.420"
[HKCR\CLSID\{D9A469E1-16AA-4b59-A051-96F2FD127D94}\Programmable]
[HKCR\CLSID\{D9A469E1-16AA-4b59-A051-96F2FD127D94}\TypeLib]
@="{DB7C8CF6-1D94-4aca-A4D7-35906E07AE15}"
[HKCR\CLSID\{D9A469E1-16AA-4b59-A051-96F2FD127D94}\VersionIndependentProgID]
@="VISSIM.Vissim"
```