the mind of movement

INTRODUCTION TO THE PTV VISSIM 6 COM API

IMPRESSUM:

PTV AG
Traffic Software
Haid-und-Neu-Straße 15
D - 76131 Karlsruhe
Germany
Tel. +49 721 9651-300
Fax +49 721 9651-562
E-Mail: info@vision.ptvgroup.com
www.ptvgroup.com
www.vision-traffic.ptvgroup.com

Table of contents

Preamble

Starting with PTV Vissim 6, the reference documentation of the COM interface of Vissim is no longer included in this document. Instead, it is provided in a more accessible HTML form as part of the Vissim Online Help system (last chapter: “Vissim – COM”). This document contains only the conceptual parts of the former COM documentation and some examples. All objects available through the Vissim COM interface and their attributes and relations are listed in the HTML COM reference in the Online Help.

1 Introduction to COM programming

1.1 General information on COM programming

1.1.1 What is COM?

The Component Object Model describes how binary components of different programs collaborate. COM gives access to data and functions contained in other programs. Since Vissim version 4.0, data and algorithms contained in Vissim can be accessed via the COM interface using Vissim as automation server. The Vissim COM interface is automatically included when the software Vissim is installed (but not in the Demo version). Since Vissim Version 4.30, COM scripts can be called directly from the Vissim main menu.

COM does not depend on a certain programming language. COM Objects can be used in a wide range of programming and scripting languages, including VBA, VBS, Python, C, C++, C#, Delphi and Matlab. In the following examples VBA is used most often. Exceptions using the programming language Python are marked explicitly.

1.1.2 Visual Basic dialects (VBA, VBS)

Visual Basic for Applications (VBA) is a programming language that facilitates working with tables, data and diagrams. VBA

- supports automatic processing and recording of repeated steps in the workflow.
- allows users to add own functions to applications as Vissim, Excel or Access.
- allows users to start and control applications as e.g. Vissim from Excel or Access.

Since VBA is a “real” programming language, it will take a certain time and continuous training to become well experienced in VBA programming. Powerful programs will be worth the efforts finally, as saving time and safety in data processing are essential. VBA is provided with e.g. Excel and thus available on many personal computers. This document describes the basic structure of a VBA program and further options for specific requirements, illustrated by various simple examples.

Visual Basic Script (VBS) is a script language closely related to VBA. These scripts are run on Windows computers using the so-called Windows Script Host. Unlike VBA, VBS is no typed language, i.e. there is only one data type called Variant which can store all kind of data. The functional scope is reduced compared to VBA. It is neither allowed nor necessary to denote the main program (like „Sub ...()“).

1.1.3 Excel as container for VBA programs

When Excel is used as container for a VBA program that activates Vissim via COM, the code can be integrated either in tables or in separate modules. If the code is part of a table, error messages are not transmitted to the GUI correctly. For example when trying to load a non-existing network file, VBA shows the error message "automation error". If the same code is located in a module, VBA produces the correct error message „CVissim::LoadNet failed! File...".

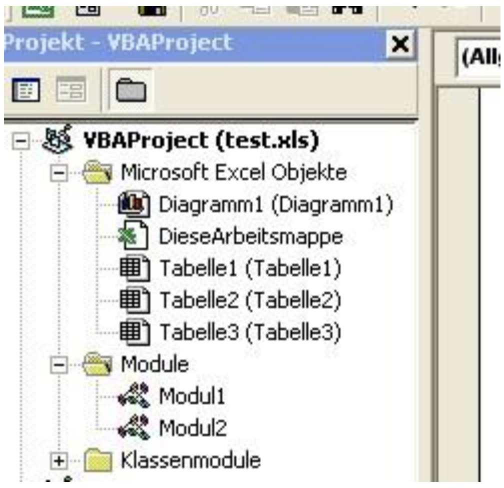


Figure 1: VBA project elements in Excel

Notice: Calling very long COM statements causes in Excel the following warning message: "Microsoft Office Excel is waiting for another application to complete an OLE action". In order to avoid this warning message you can add the following line to your code:

```
Application.DisplayAlerts = False
```

1.1.4 Python

Python provides a second, very powerful programming language for direct execution in the Vissim context. Users can execute Python scripts from the Vissim script menu, or start a Python script manually in the Python interpreter. This script can create instances of Vissim and connect to them via COM.

In contrast to VBA included with the installation of Excel, Python must be installed explicitly in most cases. Besides the interpreter and libraries there are several development environments, but every text editor can be used as well to create Python scripts. For information about Python, visit http://www.python.org/ and http://numpy.scipy.org/.

Python provides a large number of basic data types, e.g. for numerical values and strings. Besides the properties of the programming language itself Python comes with a large library covering the most different applications. Integral part of this library are data structures for matrices comprising arithmetical operations. Therefore Python is convenient for manipulations of source – destination or skim matrices. Using the interface to the Tk toolkit, but most notably through the availability of wxPython, Python provides a means to create graphical user interfaces.

Python supports properties only if the Set variant of the property takes exactly one parameter. In all other cases, properties are handled with Get and Set methods, and the latter receives an additional "Set" prefix. The most prominent example is the property AttValue which is available as two separate methods in Python:

Read value: `number = node.AttValue("NO")`
Set value: `node.SetAttValue("NO", number)`

Properties with one parameter are still treated as properties:

Read value: `mat = demandSegment.ODMatrix`
Set value: `demandSegment.ODMatrix = mat`

1.1.5 Object interface identifiers

Within different programming languages the object interfaces appear differently. The difference consists of the leading “I” (for “**I**nterface”). In VB dialects (VBA, VBS), the leading “I” is discarded from the identifier. For example for addressing an “ILinks” object, the identifier “Links” is used in the code. In VBA there is the possibility to look up the correct identifier in the so-called object catalogue.

Within all other programming languages the objects are addressed including the leading “I”. This form is used within this documentation (except in VBA examples, of course).

2 Script menu

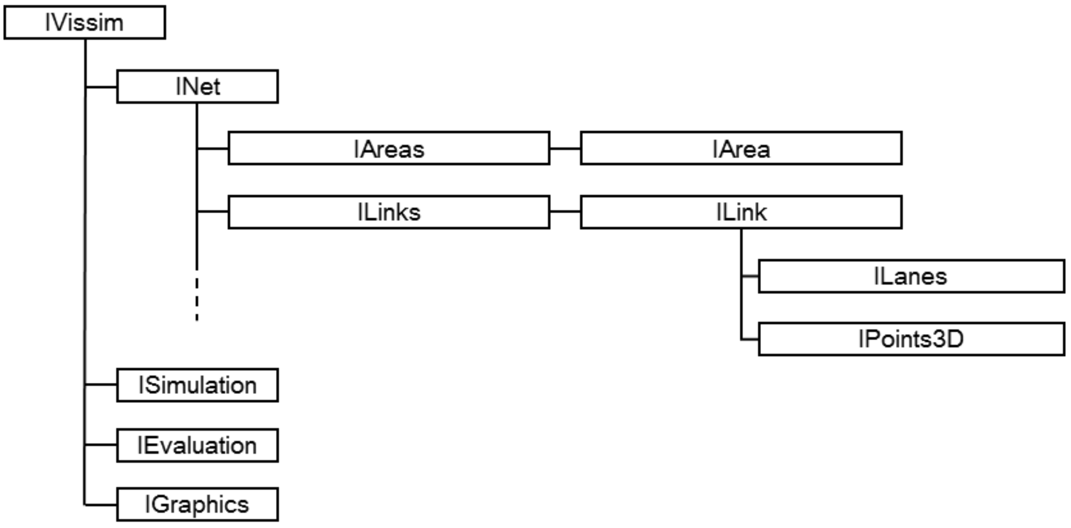
2.1.1 Starting scripts from the Vissim Script menu

You can execute a script written in VBS or Python directly from Vissim. To do so, click SCRIPT - RUN SCRIPT FILE... and select the script file to be run in the file selection dialog.

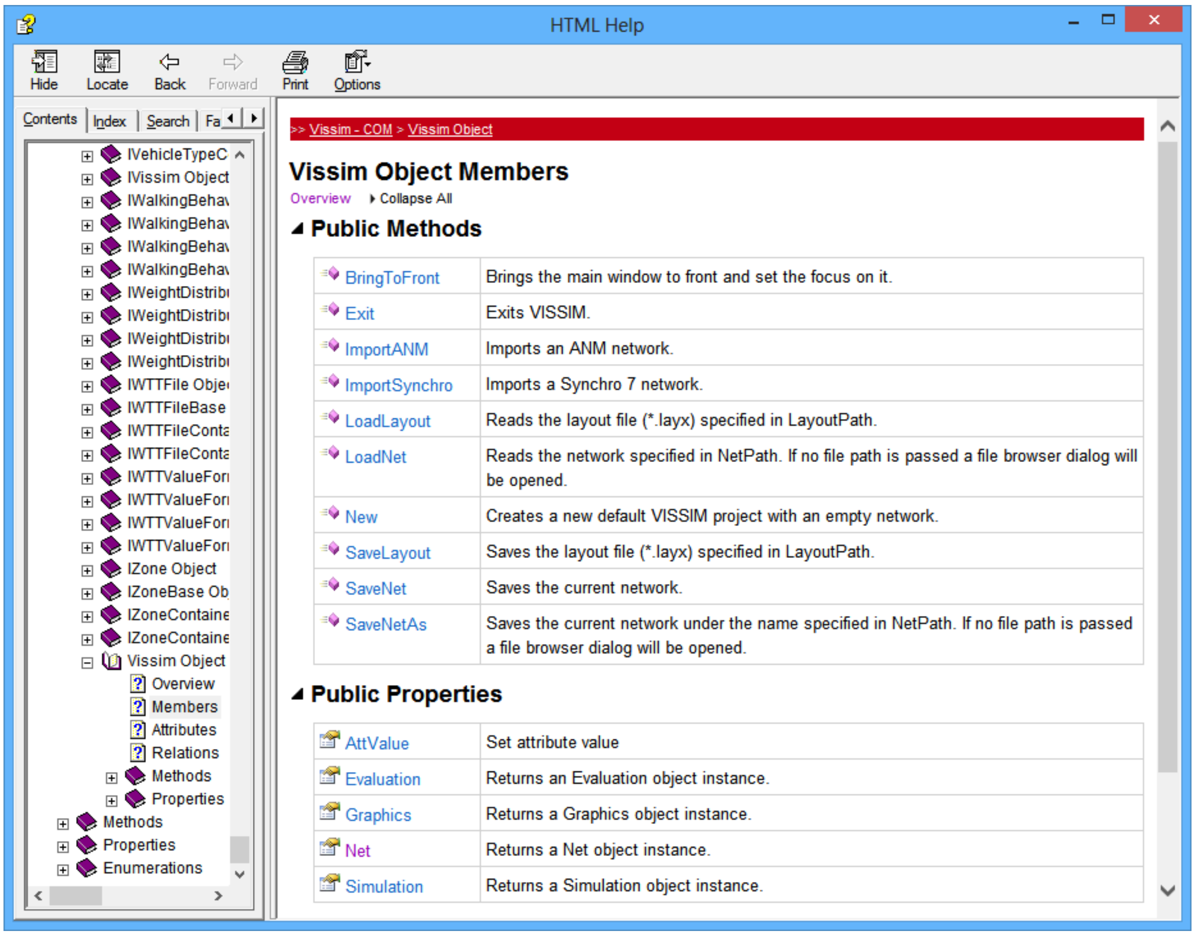
A predefined variable „Vissim“ is available in a script if started via the Script menu. It points to the instance of Vissim where the menu item was clicked. The script does not need to start an additional Vissim instance as a COM server. By this it is possible to access the currently loaded net and execute some recurring operations on it, e. g. to execute several simulation runs, changing some parameters before each run. Other data like graphic parameters can be changed by the script as well.

3 The Vissim object model

The Vissim COM Model is subject to a strict object hierarchy (see figure below). IVissim is the highest-ranking object. To access a sub-object, e.g. a link in the network, one must follow the hierarchy.



Please see the COM interface reference included with the Online Help of Vissim for details about the various objects and their methods and properties:



Vissim Object Members

Public Methods

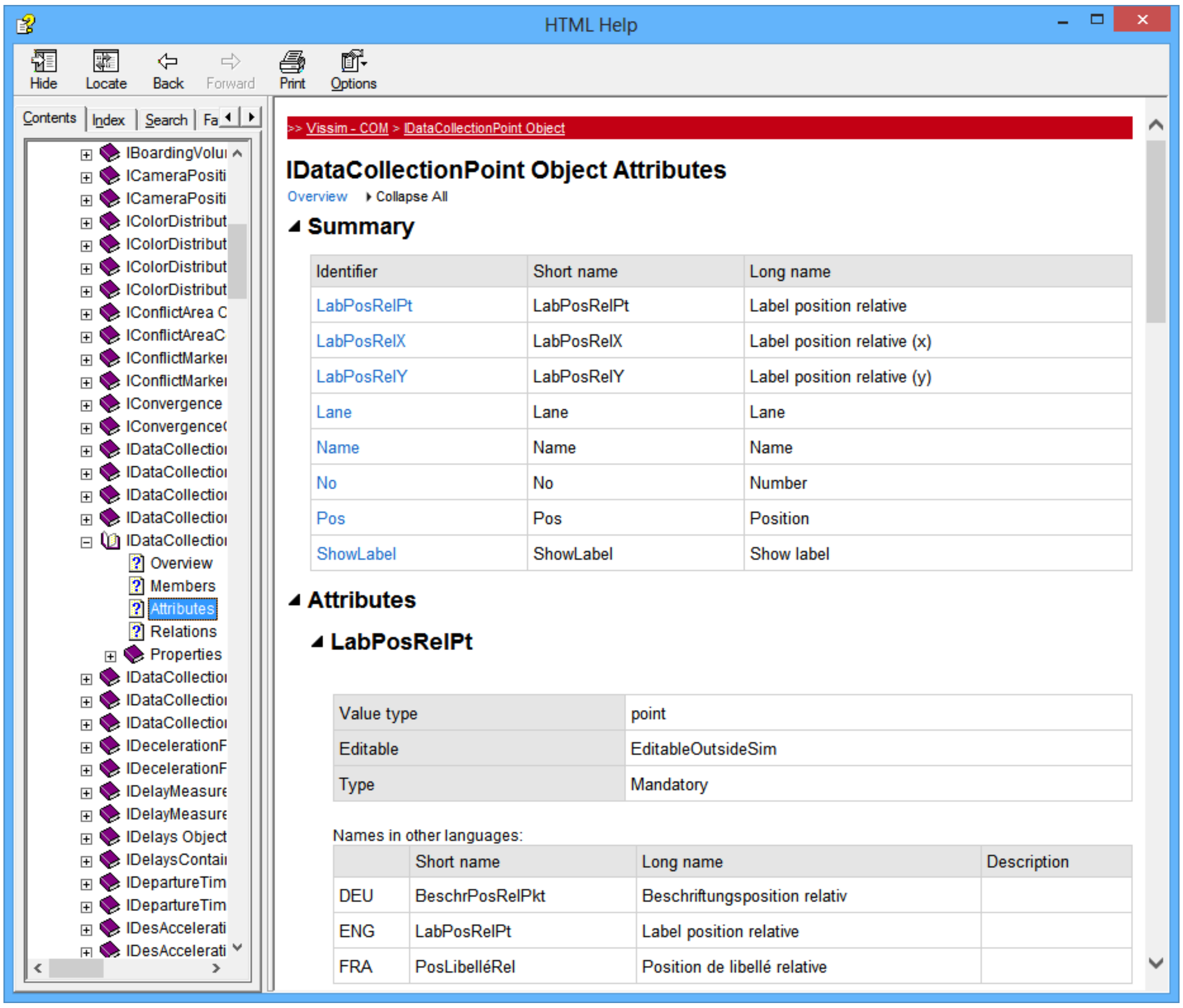
BringToFront	Brings the main window to front and set the focus on it.
Exit	Exits VISSIM.
ImportANM	Imports an ANM network.
ImportSynchro	Imports a Synchro 7 network.
LoadLayout	Reads the layout file (*.layx) specified in LayoutPath.
LoadNet	Reads the network specified in NetPath. If no file path is passed a file browser dialog will be opened.
New	Creates a new default VISSIM project with an empty network.
SaveLayout	Saves the layout file (*.layx) specified in LayoutPath.
SaveNet	Saves the current network.
SaveNetAs	Saves the current network under the name specified in NetPath. If no file path is passed a file browser dialog will be opened.

Public Properties

AttValue	Set attribute value
Evaluation	Returns an Evaluation object instance.
Graphics	Returns a Graphics object instance.
Net	Returns a Net object instance.
Simulation	Returns a Simulation object instance.

All attributes of an object are listed on the respective attribute page in the HTML COM reference. For each attribute, the following information is shown:

- the identifier (which must be used in COM code)
- the short name and the long name (in German, English and French)
- the value type
- if and when the attribute can be read and set through COM:
  - AlwaysEditable,
  - EditableOutsideSim (only when no simulation run is active),
  - ReadOnly,
  - SimAttr_Editable (only during a simulation run),
  - SimAttr_ReadOnly (exists only during a simulation run)
- the value source (“Type”):
  - Mandatory (input attribute),
  - Optional (input attribute, can be empty),
  - Calculated (derived from other attributes),
  - Evaluation (determined by an activated evaluation during a simulation run)
- up to 3 sub-attributes
- minimum, maximum and default value (if not empty)
- add-on modules which are required for the attribute to be accessible (if not empty)



>> Vissim - COM > IDataCollectionPoint Object

IDataCollectionPoint Object Attributes

Overview ▸ Collapse All

Summary

Identifier	Short name	Long name
LabPosRelPt	LabPosRelPt	Label position relative
LabPosRelX	LabPosRelX	Label position relative (x)
LabPosRelY	LabPosRelY	Label position relative (y)
Lane	Lane	Lane
Name	Name	Name
No	No	Number
Pos	Pos	Position
ShowLabel	ShowLabel	Show label

Attributes

LabPosRelPt

Value type	point
Editable	EditableOutsideSim
Type	Mandatory

Names in other languages:

	Short name	Long name	Description
DEU	BeschrPosRelPkt	Beschriftungsposition relativ	
ENG	LabPosRelPt	Label position relative	
FRA	PosLibelléRel	Position de libellé relative	

4 First steps

4.1 Access to Vissim versions and files

Vissim Instances

When working with Vissim manually it is possible to start several instances of Vissim. You can also start several different versions (e.g. Vissim 5.40 and Vissim 6.0). The same applies when working with COM: If the version number is not specified (i.e. CreateObject ("Vissim.Vissim")), the Vissim version which has been registered as COM server most recently is started. Specifying the version number during the call allows you to start a specific version of Vissim: CreateObject ("Vissim.Vissim.540") for Vissim 5.40 or CreateObject ("Vissim.Vissim.600") for Vissim 6. If you have installed a 32-bit and a 64-bit edition, you can use "Vissim.Vissim-32.600" or "Vissim.Vissim-64.600" to start a specific edition. A Vissim instance started via COM is automatically closed as soon all COM objects are set to nothing.

If a COM program is expected to connect to an already started instance of Vissim, this Vissim instance must have been started with the command line parameter *–Automation* as automation server. This Vissim instance is used by all COM programs which access the object Vissim – until the instance is closed manually.

Basic Example

This example creates a Vissim object. On screen, the Vissim GUI window appears. The network file D:\Data\example.inpx is loaded. A simulation run is started. After completion of the run, the Vissim object is deleted.

Example file: RunSimulation.xls

```
'This example shows how to load a network and run a simulation.
Sub FirstVissimExample()

    'variable declaration
    Dim Vissim As Object                     ' Variable Vissim

    'create the Vissim-Object
    '(connect the variable Vissim to the software Vissim 6)
    Set Vissim = CreateObject("Vissim.Vissim.600")

    'load the network
    Vissim.LoadNet ("D:\Date\example.inpx")

    'run the simulation
    Vissim.Simulation.RunContinuous

    'delete the Vissim object to close the Vissim software
    Set Vissim = Nothing
End Sub
```

4.2 Access to objects and attributes

4.2.1 Introduction

For access to single Vissim objects, the hierarchy of objects (see fig. Object Model) has to be followed. IVissim is the highest-ranking object of the model. IVissim allows access to the sub-objects INet (network) and ISimulation (simulation) and further lower-ranking objects.

From the object INet, access is possible to the sub-objects ILinks (links), INodes (nodes), IAreas (pedestrian areas) etc. These are so-called collection objects i.e. they represent a number of objects of the same object type. Usually collection objects within the Vissim COM model are named using the plural form. The object IAreas includes all pedestrian areas, the object INodes includes all nodes, the object ILinks includes all links of the network. (The type of the object which is listed in the “Objects” list in the “Contents” tab of the HTML COM reference usually is named with the suffix “Container”, e.g. “ILinksContainer Collection”.)

Via the collection object, each individual object of the collection can be accessed. Random access to one specified object, access to a list of all objects, and loops over all objects of the collection are possible.

4.2.2 Random access to a specified object using its key

Access via ItemByKey

The ItemByKey method guarantees unambiguous access to a single Vissim object. To access a particular Vissim object, the object’s external key (a number which is unique in the collection) is used. Then access to the properties (attributes) of the particular Vissim object is possible. Access to a network object depends on the network object type.

➨ Link #10:
```
Set Obj = Vissim.Net.Links.ItemByKey(10)
Length = Obj.AttValue("Length2D")
```

➨ Signal group #1 of signal controller #42:
```
Set SCObj = Vissim.Net.SignalControllers.ItemByKey(42)
Set SGObj = SCObj.SGs.ItemByKey(1)
State = Obj.AttValue("State")
```

4.2.3 Access to all objects of a collection

It is always possible to get all objects of a collection as a list using GetAll. The result is an array, the objects themselves can be accessed using the array index. The array must be declared with a dynamic length before. The client code must ensure that the bounds of the array are respected, otherwise an error will occur.

In principle, it is possible to construct loops if you get all objects into an array using GetAll and then iterate over the array (see below in *Loop using Item* and *Loop using For-Each*). But for programming simple loops a faster and memory-saving method is provided.

4.2.4 Loops over all objects of a collection

Loop using object enumeration

This is the most convenient method for programming a loop over all objects of a collection, since it is fast and memory saving. We recommend using this method only within new scripts.

VBA:

```
for each node in vissim.net.nodes
  node.AttValue("...") = ...
next
```

Python:

```
for node in vissim.net.nodes:
  node.SetAttValue("...", ...)
```

Loop using Item

This method also allows programming of loops:

VBA:

```
AllNodes = Vissim.Net.Nodes.GetAll
For i = 0 To UBound(AllNodes)
  Set currentNode = AllNodes(i)
  currentNode.AttValue("...") = ...
next
```

This method transfers all objects of the collection into a list using GetAll. When accessing a single object (here INode) via Item, the object is selected by its index in the collection object (here INodes). Since the index of an object may change due to changes to the Vissim network, an object cannot necessarily be determined unambiguously by its index number. That is why this type of access should be used only in loops which are completed before such index changes can happen.

Usually the method using object enumeration is preferable, but some special requirements such as loop over every second object of the collection can only be realized using Item.

Loop using For-Each

A for-each loop allows iterations over the collection objects providing access to every single object of the collection. The for-each loop can be used alternatively instead of the Item method.

VBA:

```
AllNodes = Vissim.Net.Nodes.GetAll
For each node in allNodes
  node.AttValue("...") = ...
next
```

Access using an iterator object

Each collection object provides an iterator object that points to the first object of the collection originally (and after Reset). The iterator can be incremented to point to the next object (until there is no next object anymore and the iterator becomes invalid). In contrast to the property GetAll that saves all objects of a collection within an array and that must be used before using the methods Item or For...Each, the use of the iterator object does not depend on size limits of such arrays. However, the use of iterator objects is slower than the object enumeration method.

VBA:

```
Set nodeIter = Vissim.Net.Nodes.Iterator
  While nodeIter.Valid
    Set curNode = nodeIter.Item
    curNode.AttValue("...") = ...
    nodeIter.Next
  Wend
```

4.2.5 Read and write attribute values

Read access is provided for all attributes (for some dynamic objects like vehicles only during a simulation run), whereas write access is limited to a number of attributes (and sometimes only outside of a simulation run or only during a simulation run). Which attributes are available for which objects and what kind of access is possible is documented in the HTML COM reference.

Attributes are always accessed by the method AttValue:

```
number = Node.AttValue("NO")
Node.AttValue ("NO") = number + 1
```

For access to the attributes, only the (English) identifiers shown in the HTML COM reference can be used (not the language specific short and long names) and they are accepted case insensitive. If the GUI language of Vissim is set to English, the column headers in the list windows display the English short names which are identical with the identifiers.

Sub-attributes are added in parentheses to the attribute, e.g. “Volume(1)” for the volume in the first time interval of a vehicle input. If there are multiple sub-attributes, they are separated by commas, e.g. “RelFlow(1,4)” for the relative flow of vehicle type number 1 with desired speed distribution number 4 in a vehicle composition. The sub-attributes for each attribute are also listed in the HMTL COM reference, and their values can be seen in the column headers in the list windows in Vissim (depending on the attribute selection, of course). For time intervals, the column header in the list window shows the start of the time interval while the COM interface requires the index (starting at 1).

If the value of a calculated numerical attribute is not defined, since e.g. the respective time interval has not yet been simulated, the function AttValue can return a useless value.

4.2.6 Run simulations

The COM object Vissim.Simulation permits the execution of simulation runs in several manners. The simulation can be run continuous which means that no other operations can be performed before the simulation run is completed. The other option is to run the simulation step by step which means that after every simulation step other operations can be performed before the next time step is simulated. Stopping the simulation can be one of these operations. For examples please refer to 6.3.

5 COM in other programming languages

5.1 COM in C#

It is possible to use COM functions within the C# programming language without much expenditure. In the development environment used, set a reference to the Vissim executable file installed (currently this is VISSIM.EXE) in order to use Vissim COM objects. Then the objects and methods of the Vissim COM object model can be used in the C# project. When using the development framework Microsoft Visual C# .NET, the object model can be browsed using the „Object Browser“ similar to the object catalogue in VBA.

In classes using Vissim objects set the clause „using VISSIMLIB“ to access the Vissim COM object model. A Vissim entity is created by calling the constructor using the expression „VissimClass Vissim = new VissimClass()“.

Note that object names and some of the method and property names differ from the notation used in VBA (and in this documentation). For example the „I“ at the beginning of the object name can be written or left out. The property „AttValue“ occurs twice according to the different signatures using input or output parameters. The variants are called „get_AttValue“ and „set_AttValue“, where this last one takes as parameters the attribute identifier and the new value. Other examples of differing identifiers are „get_ItemByKey“ and „get_GetMultiAttValues“.

Furthermore, in C# you might have to specify optional parameters explicitly, which means there are no optional parameters. For example for loading a input file you could have to specify the (in VBA optional) input parameter „Additive“. This depends on your development environment.

Since C# does not accept implicit casting explicit type conversions have to be used if necessary. Most methods of the Vissim COM interface return objects of (C#-)type System.Object. These must be converted by a static cast to their „real“ type, e.g. ILink, if the type specific properties and methods are needed.

C# example:

```
using System;
using VISSIMLIB;

namespace ConsoleApplication1
{
	/// <summary>
	/// Class NameReader reads the names of all Links in Vissim network.
	/// <summary>
	class NameReader
	{
		/// <summary>
		/// The main entry point for the application.
		/// <summary>

		static void Main(string[] args)
		{
			try
			{
				VissimClass vissim = new VissimClass();

				vissim.LoadNet(@"D:\Date\example.inpx");

				foreach (ILink link in vissim.Net.Links)
				{
					Int32 linkNumber = (Int32)link.AttValue["No"];
					string linkName = (string)link.AttValue["Name"];
					Console.WriteLine("Link " + linkNumber + " (" + linkName + ")");
				}

				vissim.Exit();
			}
			catch (Exception e)
			{
				Console.WriteLine("Error while reading links via COM: " + e.ToString());
			}

		}
	}
}
```

This example loads an input file, iterates over all links and writes their numbers and names to the console. An example output for a network consisting of only two links could look like this:

C# example output:

```
Link 1 (Main North Incoming)
Link 2 (Main North Outgoing)
```

5.2 COM in JAVA

For pure JAVA applications that run with any common JAVA virtual machine without using a Microsoft-specific development framework, the concept of COM interfaces is not available. But any single COM server can be made accessible for JAVA applications using so-called COM bridges. To do so, wrapper classes corresponding to the objects provided by the Vissim COM interface must be created in an offline process. These classes must be integrated into the JAVA application. Once this process is finished, the use of Vissim COM objects in the JAVA application is as easily possible as in other programming languages.

There exist several different JAVA COM bridges. Between those there are commercial products such as Bridge4Java (IBM) as well as open source software like JaCoB (http://sourceforge.net/projects/jacob-project).

Because of the previously described difficulties the use of COM in a pure JAVA environment requires deep knowledge of these software development techniques.

5.3 COM in C++

Although the concept is similar, the use of Vissim objects via COM in C++ is more complex compared to C#. First, the directive „#import ...“ includes the installed executable into the project. In the example the namespace „VISSIMLIB“ is assigned to it. After initializing the COM interface which is triggered in the Microsoft Visual C++ development environment by calling „CoInitialize(NULL)“ a Vissim object can be created.

Now the objects of the Vissim COM object model are available and can be used to declare variables. Communication via the interface uses pointers whose names are created by appending „Ptr“ to the name contained in this documentation.

These pointers have to be connected to Vissim objects explicitly. This provides access to the Vissim data model. After use these connections have to be disconnected explicitly.

The example we show down here has been created using the Microsoft Visual C++ development environment.

C++ example:

```
#include <iostream>

// import the *.exe file providing the COM interface
// use your Vissim installation path here as the example shows
#import "C:\\Program Files\\PTV Vision\\PTV Vissim 6\\Exe\\Vissim.exe" rename_namespace ("VISSIMLIB")

using namespace std;

template<typename T> inline void QueryAttach(T& cp, IUnknown *pUnknown)
{
	T::Interface		*pT;
	HRESULT			hr;

	hr = pUnknown -> QueryInterface(__uuidof(T::Interface), reinterpret_cast<void **>(&pT));
	if (hr != S_OK)
		throw _com_error(hr);
	else
		cp.Attach(pT);
}

int main(int argc, char* argv[])
{
	// initialize COM
	CoInitialize(NULL);

	// create Vissim object
	VISSIMLIB::IVissimPtr pVissim;
	HRESULT hr = pVissim.CreateInstance("Vissim.Vissim.600");
	if (hr != S_OK)
	{
		cout << "COM connection to Vissim cannot be established." << endl;
	}

	try {
		pVissim->LoadNet(bstr_t("D:\Date\example.inpx "), false);
		VISSIMLIB::INetPtr pNet;
		VISSIMLIB::ILinkContainerPtr pLinkContainer;
		VISSIMLIB::IIteratorPtr pIter;
		VISSIMLIB::ILinkPtr pLink;
		int number;
		bstr_t name;
		// attach pointer to Vissim objects
		QueryAttach(pNet, pVissim -> GetNet());
		QueryAttach(pLinkContainer, pNet->GetLinks());
		QueryAttach(pIter, pLinkContainer->GetIterator());
		while (pIter -> GetValid()) {
			QueryAttach(pLink, pIter->GetItem());
			number = pLink->GetAttValue("No");
			name = pLink->GetAttValue("Name");
			cout << "Link " << number << " (" << name << ")" << endl;
			pIter -> Next();
		}

		// release pointers
```

```
			if (pLink.GetInterfacePtr() != NULL)
				pLink.Detach()->Release();

			if (pIter.GetInterfacePtr() != NULL)
				pIter.Detach() -> Release();

			if (pLinkContainer.GetInterfacePtr() != NULL)
				pLinkContainer.Detach() -> Release();

			if (pNet.GetInterfacePtr() != NULL)
				pNet.Detach() -> Release();
		}
		catch (...) {
			cout << "Error during COM connection." << endl;
		}

		// free Vissim object
		if (pVissim.GetInterfacePtr() != NULL)
			pVissim.Detach()->Release();

		// uninitialize COM
		CoUninitialize();

		return 0;
}
```

This example loads an input file, iterates over all links and writes their numbers and names to the console. This is exactly what the C# example does. Therefore the possible output is the same.

<u>C++ example output:</u>

```
Link 1 (Main North Incoming)
Link 2 (Main North Outgoing)
```

6 Vissim Examples

Preliminary note concerning examples

In most of the examples object variables are declared using the so-called „early binding". Early binding means that objects are not just declared as „Object" in VBA, but their precise object type (called class) is indicated. The advantages are:

- The programs are easier to read and understand for the user.
- Program execution is generally faster
- The "statement building tools" of the development environment can be used when writing the program. Available properties and methods of an object are shown automatically and can be selected by the programmer.

Early binding is available only after the VBA environment is made acquainted with the Vissim object library. This is done by setting a reference to the Vissim object library in the Extras+References menu. The object library is part of the Vissim program and is automatically available after the installation of Vissim.

The main disadvantage of early binding is the fact that under certain circumstances, the execution of the program is bound to the concrete computer. If Vissim is installed under another path on another computer, the reference points to nowhere. VBA then rejects the execution. If no reference is set at all the program will be executed on all computers if any Vissim instance can be reached. If you want to share programs and use them on several computers it will be more convenient to avoid early binding. The example files that we deliver do not use early binding for that reason; that is why the program codes differ slightly from the printed versions.

6.1 Reading and Saving Data

6.1.1 Creating Vissim Objects and Reading the Input File

This example creates a Vissim object and loads an input file. The name of the input file is read from the Excel worksheet. The net is shown in the Vissim window. Additionally the Vissim window is displayed in the foreground.

Example: LoadInputFile

```
'This example shows how to create a Vissim object and load an input file from worksheet cell.
Sub LoadInputFileExample()

    'variable declaration of variable Vissim
    Dim Vissim As Object

    'create the Vissim object (connect the variable Vissim to the software Vissim)
    Set Vissim = CreateObject("Vissim.Vissim.600")

    'Load input file (file name from cell B1)
    Vissim.LoadNet  Cells(1, 2)

    'Display message to show you that Vissim has loaded the input file
    MsgBox "Finished loading input file"

    'Bring Vissim window to front and set focus on it
    Vissim.BringToFront
    'delete the Vissim object to close the Vissim software
    Set Vissim = Nothing
End Sub
```

6.1.2 Reading and Saving the Input File

This example creates a Vissim object and loads an input file. The name of the input file is read from the Excel worksheet. The width of all lanes, which have the width attribute set, is to be modified. Then the result is saved to a new input file.

Example: SaveInputFile

```
'This example creates a Vissim object and loads an input file defined in a
'worksheet cell.
'The width attribute of each lane is modified and the result is saved in a
'new input file.

Sub SaveInputFileExample()

    'declare object Vissim
    Dim Vissim As Object

    'declare other variables
    Dim Width As Variant

    'create the Vissim object (connect variable Vissim to the software Vissim)
    Set Vissim = CreateObject("Vissim.Vissim.600")

    'load input file (filename from cell B1)
    Vissim.LoadNet Cells(1, 2)

    'iterate over all links
    For Each Link In Vissim.Net.Links
        ' iterate over all associated lanes
        For Each Lane In Link.Lanes
            'get width of lane
            Width = Lane.AttValue("Width")
            'check if lane has width
            If Not Width = Empty Then
                'calculate and set new width
                Width = Width * 1.1
                Lane.AttValue("Width") = Width
            End If
        Next Lane
    Next Link

    'save input file (filename from cell B2)
    Vissim.SaveNetAs Cells(2, 2)

    'delete all objects to close Vissim software
    Set Vissim = Nothing

End Sub
```

6.2 Access to Objects and Attributes

6.2.1 Finding vehicle inputs using ItemByKey

The example below illustrates the access to a single node. The required node number is read from the specified cell in the Excel worksheet.

Example: GetItemByKey_VehicleInput

```
'This example shows how to access single vehicle inputs of a network
'using ItemByKey

Sub GetItemByKey_VehicleInput()

    'declaration of variables
    Dim Vissim As Object             ' Variable Vissim
    Dim VehicleInput As Variant

    'clear all rows in worksheet from number 12 to the end
    Rows("13:65536").ClearContents

    'create the Vissim object (connect variable Vissim to the software Vissim)
    Set Vissim = CreateObject("Vissim.Vissim.600")
    'load input file (filename from cell B1)
    Vissim.LoadNet Cells(1, 2)

    'access vehicle input by number (key), the number is read from cell C1
    Set VehicleInput = Vissim.Net.VehicleInputs.ItemByKey(Cells(1, 3))

    'read vehicle input attributes and write it in cells
    Cells(13, 1) = VehicleInput.AttValue("No")
    Cells(13, 2) = VehicleInput.AttValue("NAME")
    Cells(13, 3) = VehicleInput.AttValue("Link")

    'delete all objects to close Vissim software
    Set VehicleInput = Nothing
    Set Vissim = Nothing

End Sub
```

6.2.2 Loop using Object Enumeration

The example demonstrates access to vehicle inputs inside a loop using object enumeration. It iterates over all objects and writes some attribute values to an Excel sheet.

Example: VehicleInputAttributeEnumerationExample

```
'This example shows how to access single vehicle inputs of a net
'by using an enumeration

Sub VehicleInputAttributeEnumerationExample()

    'declaration of variables
    Dim Vissim As Object             ' Variable Vissim
    Dim VehicleInput As Variant

    'clear all rows in worksheet from number 12 to the end
    Rows("13:65536").ClearContents

    'Create the Vissim object (connect variable Vissim to the software Vissim)
    Set Vissim = CreateObject("Vissim.Vissim.600")
    'load input file (filename from cell B1)
    Vissim.LoadNet Cells(1, 2)

    Row = 13
    For Each VehicleInput In Vissim.Net.VehicleInputs
        'read vehicle input attributes and write it in cells
        Cells(Row, 1) = VehicleInput.AttValue("No")
        Cells(Row, 2) = VehicleInput.AttValue("NAME")
        Cells(Row, 3) = VehicleInput.AttValue("Link")
        Row = Row + 1
    Next VehicleInput

    'delete all objects to close Vissim software
    Set VehicleInput = Nothing
    Set Vissim = Nothing

End Sub
```

6.2.3 Loop using Item

The example demonstrates access to vehicle inputs inside a loop using Item. A list of vehicle inputs references is assigned to AllVehicleInputs, then the loop over that list is started and prints some attribute contents to the excel sheet.

Example: VehicleInputAttributeItemExample

```
'This example shows how to access single vehicle input of a net by using Item.
'Attention: Item identifies a vehicle input by its index in the Container.
'A vehicle input cannot be uniquely identified through the index, the index is
'subject to change if the Vissim network is altered.
'Accessing by Item should only be used with loops (i.e. processing all available
'vehicle inputs).

Sub VehicleInputAttributeItemExample()

    'declaration of variables
    Dim Vissim As Object            ' Variable Vissim
    Dim VehicleInput As Variant
    Dim AllVehicleInputs As Variant
    Dim Row As Long

    'clear all rows in worksheet from number 12 to the end
    Rows("13:65536").ClearContents

    'Create the Vissim object (connect variable Vissim to the software Vissim)
    Set Vissim = CreateObject("Vissim.Vissim.600")
    'load input file (filename from cell B1)
    Vissim.LoadNet Cells(1, 2)

    Row = 13
    AllVehicleInputs = Vissim.Net.VehicleInputs.GetAll
    For i = 0 To UBound(AllVehicleInputs)
        Set VehicleInput = AllVehicleInputs(i)

        'read vehicle input attributes and write it in cells
        Cells(Row, 1) = VehicleInput.AttValue("No")
        Cells(Row, 2) = VehicleInput.AttValue("NAME")
        Cells(Row, 3) = VehicleInput.AttValue("Link")
        Row = Row + 1
    Next

    'delete all objects to close Vissim software
    Set VehicleInput = Nothing
    Set AllVehicleInputs = Nothing
    Set Vissim = Nothing

End Sub
```

6.2.4 Loop via For...Each

The example demonstrates access using a For...Each loop. The collection of vehicle inputs is assigned to AllVehicleInputs, then the loop over that list is started and prints some attribute contents to the excel sheet.

Example: VehicleInputAttributeForEachExample

```
'This example shows how to access single vehicle inputs with For-Each-Loop.
Sub VehicleInputAttributeForEachExample()

  'declaration of variables
  Dim Vissim As Object                     ' Variable Vissim
  Dim VehicleInput As Variant
  Dim AllVehicleInputs As Variant
  Dim Row As Long

  'clear all rows in worksheet from number 12 to the end
  Rows("13:65536").ClearContents

  'Create the Vissim object (connect the variable Vissim to the software Vissim)
  Set Vissim = CreateObject("Vissim.Vissim.600")
  'load input file (filename from cell B1)
  Vissim.LoadNet Cells(1, 2)

  Row = 13
  AllVehicleInputs = Vissim.Net.VehicleInputs.GetAll
  For Each VehicleInput In AllVehicleInputs
    'read vehicle input attributes and write it in cells
    Cells(Row, 1) = VehicleInput.AttValue("No")
    Cells(Row, 2) = VehicleInput.AttValue("NAME")
    Cells(Row, 3) = VehicleInput.AttValue("Link")
    Row = Row + 1
  Next

  'delete all objects to close Vissim software
  Set VehicleInput = Nothing
  Set AllVehicleInputs = Nothing
  Set Vissim = Nothing

End Sub
```

6.2.5 Loop using Iterator Object

The example demonstrates a loop that is set up using the IIterator object of the IVehicleInputContainer object. The iterator is received from the VehicleInputsContainer object. Then the loop is started. In every round the current item of the iterator is received and at the end the iterator is incremented. Please note that the variable for the iterator has to be declared as of type Object and not of type Variant.

Example: VehicleInputAttributeIteratorExample

```
'This example shows how to access single vehicle inputs with Iterator.
Sub VehicleInputAttributeIteratorExample()

    'declaration of variables
    Dim Vissim As Object             ' Variable Vissim
    Dim VehicleInput As Variant
    Dim VehicleInputIterator As Object
    Dim Row As Long

    'clear all rows in worksheet from number 12 to the end
```

```
        Rows("13:65536").ClearContents

        'Create the Vissim object (connect variable Vissim to the software Vissim)
        Set Vissim = CreateObject("Vissim.Vissim.600")
        'load input file (filename from cell B1)
        Vissim.LoadNet Cells(1, 2)

        Row = 13
        Set VehicleInputIterator = Vissim.net.VehicleInputs.Iterator
        While VehicleInputIterator.Valid
            Set VehicleInput = VehicleInputIterator.Item

            'read vehicle input attributes and write it in cells
            Cells(Row, 1) = VehicleInput.AttValue("No")
            Cells(Row, 2) = VehicleInput.AttValue("NAME")
            Cells(Row, 3) = VehicleInput.AttValue("Link")

            Row = Row + 1
            VehicleInputIterator.Next
        Wend

        'delete all objects to close Vissim software
        Set VehicleInput = Nothing
        Set VehicleInputIterator = Nothing
        Set Vissim = Nothing

    End Sub
```

6.2.6 Reading and Saving Individual Attributes

The following example shows the access for reading and writing attributes. First the name of the vehicle input is changed so it includes its number. The attribute “Name” is received and written via the AttValue property. Then the volume of the vehicle input is decreased by 10 percent for the first time interval. The time interval has to be specified together with the attribute name “Volume”. This can be done by writing the attribute name followed by a comma separated list of the sub attributes in parenthesis. For a list of necessary sub attributes please refer to the COM interface reference, which is part of the Vissim Online Help. Please note that you have to pass the key of the related sub attribute type. For example you have to write “Volume(1)” to refer to the first time interval by its key 1. Using the start time of the time interval won’t work.

Example: VehicleInputNameAndVolumeExample

```
    Sub VehicleInputNameAndVolumeExample()

        'declaration of variables
        Dim Vissim As Object             ' Variable Vissim
        Dim VehicleInput As Variant

        'Create the Vissim object (connect variable Vissim to the software Vissim)
        Set Vissim = CreateObject("Vissim.Vissim.600")
        'load input file (filename from cell B1)
        Vissim.LoadNet Cells(1, 2)

        For Each VehicleInput In Vissim.net.VehicleInputs
            'change name of vehicle input
            newName = "Input #" + Str(VehicleInput.AttValue("No"))
            VehicleInput.AttValue("Name") = newName

            'decrease volume of vehicle input for first time interval by 10 percent
            volume = VehicleInput.AttValue("Volume(1)")
            volume = 0.9 * volume
            VehicleInput.AttValue("Volume(1)") = volume
        Next VehicleInput
        'delete all objects to close Vissim software
        Set VehicleInput = Nothing
        Set Vissim = Nothing

    End Sub
```

6.2.7 Reading and Saving Attributes with Combined Keys

Attributes which reference network objects contain keys. A key can be a single key or a combined key. Such a combined key can be found in signal heads which can reference a signal group. Since the signal group is a part of a signal controller it can only be accessed by using the signal controller. Therefore the key in the signal heads list in Vissim looks like “5 – 1” where 5 stands for the number of the signal controller and 1 for the number of the signal group. If you want to use this attribute in COM you also have to specify the combined key. There you have to write “5-1” without spaces to get the same result as with the string mentioned above for lists. The following example changes the signal group of a given signal head to the first signal group of the first signal controller.

Example: SignalHeadsExample

```
Sub SignalHeadsExample()

    'declaration of variables
    Dim Vissim As Object             ' Variable Vissim
    Dim SignalHead As Variant
    Dim SignalController As Variant
    Dim SignalGroup As Variant

    'Create the Vissim object (connect variable Vissim to the software Vissim)
    Set Vissim = CreateObject("Vissim.Vissim.600")
    'load input file (filename from cell B1)
    Vissim.LoadNet Cells(1, 2)

    'get the signal head
    Set SignalHead = Vissim.net.SignalHeads.ItemByKey(Cells(1, 3))

    'search the first signal group to use
    For Each SignalController In Vissim.net.SignalControllers
        If SignalController.SGs.Count > 0 Then
            Set SignalGroup = SignalController.SGs.Iterator.Item
            Exit For
        End If
    Next

    'set the new signal group
    SCNo = Str(SignalController.AttValue("No"))
    SGNo = Str(SignalGroup.AttValue("No"))
    SignalHead.AttValue("SG") = SCNo + "-" + SGNo

    'delete all objects to close Vissim-software
    Set SignalHead = Nothing
    Set SignalController = Nothing
    Set SignalGroup = Nothing

    Set Vissim = Nothing

End Sub
```

6.2.8 Reading and Saving Attributes of Several Network Objects

The program reads the values of the attributes “No”, “Name” and “Volume(1)” for all vehicle inputs using GetMultipleAttributes. These attributes are passed to this function as list typed as a variant. Then the new names and volums are calculated. The new attribute values are set via SetMultipleAttributes. The result of this operation is the same as in 6.2.6. Please note that the attribute values are stored in a matrix by these functions. You can access them with Result(i, j) where i is the index of the vehicle input in the matrix and j the index of the attribute. In this example Result(0, 1) refers to the name of the first vehicle input.

Example: VehicleInputNameAndVolumeExample

```
Sub VehicleInputNameAndVolumeExample()

    'declaration of variables
    Dim Vissim As Object            ' Variable Vissim
    Dim Result As Variant

    'Create the Vissim object (connect variable Vissim to the software Vissim)
    Set Vissim = CreateObject("Vissim.Vissim.600")
    'load input file (filename from cell B1)
    'Vissim.LoadNet Cells(1, 2)

    'create a variant with the relevant attributes
    Dim relevantAttributes(0 To 2) As Variant
    relevantAttributes(0) = "No"
    relevantAttributes(1) = "Name"
    relevantAttributes(2) = "Volume(1)"

    'get the attributes as matrix
    Result = Vissim.net.VehicleInputs.GetMultipleAttributes(relevantAttributes)

    'loop over the attributes of all vehicle inputs
    For i = 0 To UBound(Result)
        'change name of vehicle input
        Result(i, 1) = "Input #" + Str(Result(i, 0))
        'decrease volume of vehicle input by 10 percent
        Result(i, 2) = 0.9 * Result(i, 2)
    Next i

    'apply the modified attributes to the vehicle inputs
    Vissim.net.VehicleInputs.SetMultipleAttributes relevantAttributes, Result

    'delete all objects to close Vissim software
    Set Vissim = Nothing

End Sub
```

6.3 Simulation Runs

Besides operating on the network you can also configure and run the simulation. In Vissim you have two options for running the simulation which will be explained in the following sections.

6.3.1 Configuring the Simulation Run

You can configure the simulation run via the “Simulation” property of Vissim. In the following example the initial random seed is set to 21. The simulation is run four times while incrementing the random seed by three every time. So the simulation is run four times with the random seeds 21, 24, 27 and 30. For more configuration options please refer to the COM interface reference.

Example: ConfigureSimulationRun

```
Sub ConfigureSimulationRun()

    'declaration of variables
    Dim Vissim As Object              ' Variable Vissim
    Dim Result As Variant

    'Create the Vissim object (connect variable Vissim to the software Vissim)
    Set Vissim = CreateObject("Vissim.Vissim.600")
    'load input file (filename from cell B1)
    Vissim.LoadNet Cells(1, 2)

    'configure the simulation
    Vissim.Simulation.AttValue("RandSeed") = 21
    Vissim.Simulation.AttValue("RandSeedIncr") = 3
    Vissim.Simulation.AttValue("NumRuns") = 4

    'run the simulation
    Vissim.Simulation.RunContinuous

    'delete all objects to close Vissim software
    Set Vissim = Nothing

End Sub
```

6.3.2 Running the Simulation in Continuous Mode

Running the simulation in continuous mode means that you start the simulation and can not execute any other operations via COM while the simulation is running. As soon as the simulation is finished the instructions after the function call to run the simulation are processed. In the example the evaluation of the queue counters is read and written to cells after the simulation. Please note that you have to specify the sub attributes “SimulationRun” and “TimeInterval” as keys together with the attribute name as shown below.

Example file: StartAssignment.xls

```
Sub RunSimulationContinuous()

    'declaration of variables
    Dim Vissim As Object              ' Variable Vissim
    Dim QueueCounter As Variant
    Dim Row As Long

    'clear all rows in worksheet from number 12 to the end
    Rows("13:65536").ClearContents

    'Create the Vissim object (connect variable Vissim to the software Vissim)
    Set Vissim = CreateObject("Vissim.Vissim.600")
    'load input file (filename from cell B1)
    Vissim.LoadNet Cells(1, 2)

    'configure the evaluations
    Vissim.Evaluation.AttValue("QueuesCollectData") = True

    'run the simulation
    Vissim.Simulation.RunContinuous

    'read evaluation
    Row = 13
    For Each QueueCounter In Vissim.net.QueueCounters
        'read queue counter attributes and write it in cells
        Cells(Row, 1) = QueueCounter.AttValue("No")
        Cells(Row, 2) = QueueCounter.AttValue("QLen(1,1)")
        Cells(Row, 3) = QueueCounter.AttValue("QLenMax(1,1)")
        Row = Row + 1
    Next

    'delete all objects to close Vissim software
    Set Vissim = Nothing

End Sub
```

6.3.3 Running the Simulation in Single Steps

Running the simulation in single steps means that you can operate on the network or the simulation after each simulation step. Furthermore you can stop the simulation after each simulation step. Please note that some operations are not allowed during the simulation. Only attributes which are marked "AlwaysEditable" or "SimAttr_Editable" can be modified during a simulation run. Please refer to the COM interface reference for more information. The following example runs the simulation 3600 steps, which equals 3600 seconds for a simulation resolution of 1. The simulation is stopped earlier when the vehicle with the number from cell C1 has entered and left the network.

Example: RunSimulationSingleStep

```
Sub RunSimulationSingleStep()

    'declaration of variables
    Dim Vissim As Object             ' Variable Vissim
    Dim Vehicle As Variant
    vehicleWasInNet = False
    vehicleFound = False

    'Create the Vissim-Object (connect variable Vissim to the software Vissim)
    Set Vissim = CreateObject("Vissim.Vissim.600")
    'load input file (filename from cell B1)
    'Vissim.LoadNet Cells(1, 2)

    'configure simulation
    Vissim.Simulation.AttValue("SimRes") = 1

    'run the simulation
    For i = 0 To 3600
        'do one simulation step
        Vissim.Simulation.RunSingleStep
        'look for the vehicle with the wanted number
        vehicleFound = False
        For Each Vehicle In Vissim.net.Vehicles
            'look for vehicle number from cell C1
            If Vehicle.AttValue("No") = Cells(1, 3) Then
                vehicleWasInNet = True
                vehicleFound = True
                Exit For
            End If
        Next

        'quit if vehicle left the network
        If vehicleWasInNet And Not vehicleFound Then
            Exit For
        End If
    Next i

    'stop the simulation
    Vissim.Simulation.Stop

    'delete all objects to close Vissim-software
    Set Vissim = Nothing

End Sub
```

PTV GROUP

the mind of movement

PTV AG
Haid-und-Neu-Straße 15
D - 76131 Karlsruhe
Germany
Telefon +49 (0) 721 9651-300
Fax +49 (0) 721 9651-562
E-Mail: info.vision@ptv.de
www.ptvag.com
www.ptv-vision.com