PTV GROUP

PTV Vissim 2023

Introduction to the COM API

Imprint

PTV Planung Transport Verkehr GmbH

Address:

Haid-und-Neu-Str. 15

76131 Karlsruhe, Germany

Management Board:

Christian U. Haas (Vors.), Johannes Klutz

Contact:

Phone: +49 (0)721 9651-0

Fax: +49 (0)721-9651-699

E-Mail: info@ptvgroup.com

www.ptvgroup.com

Entry in the Commercial Register:

Local Court Mannheim HRB 743055

Sales Tax ID:

Sales tax identification number according to §27 a Umsatzsteuergesetz: DE 812 666 053

Contents

Preamble

This document contains an introduction to and the conceptual parts of the PTV Vissim COM interface accompanied by some examples. The complete reference of the COM interface is not part of this document but provided in HTML format as part of the Vissim Online Help system (accessible from the Vissim GUI via HELP - COM HELP). There, all COM objects together with their methods, properties, attributes and relations are listed.

COM Examples

Beside examples of various COM applications, there are also examples provided for different programming languages demonstrating the basic syntax of how to control PTV Vissim through COM. These examples are located in ...\EXAMPLES TRAINING\COM\BASIC COMMANDS\ for the following programming languages:

- VBA File: COM Basic Commands.bas
- Python, File: COM Basic Commands.py
- C++, File: COM Basic Commands.cpp
- C# File: COM Basic Commands.cs
- Matlab File: COM Basic Commands.m
- Java File: COM Basic Commands.java, COM Basic Commands - Utils.java

For further information, see the description file 'COM Basic Commands Desc_ENG.pdf' or the comments in the corresponding files. You can open all source files by using a plain text editor.

Webinar "Scripting in PTV Vissim using the COM interface"

You are welcome to view our Webinar about COM scripting in PTV Vissim. In this webinar we explain how a COM script can be used in PTV Vissim. This webinar is for beginners in COM scripting, you do not need any previous knowledge of the COM interface or scripting. We will also demonstrate applications of COM scripts.

You can access all our Webinars through our Webinar Archive:
https://company.ptvgroup.com/en/resources/resource-library

1 Introduction to COM Programming

1.1 What is COM?

The Component Object Model describes how binary components of different programs collaborate. COM provides access to data and functions contained in other programs. Data and objects 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 (except in the Demo version) and set as COM server if the corresponding option is activated during the installation process.

COM does not depend on a specific 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.

1.2 Visual Basic dialects (VBA, VBS)

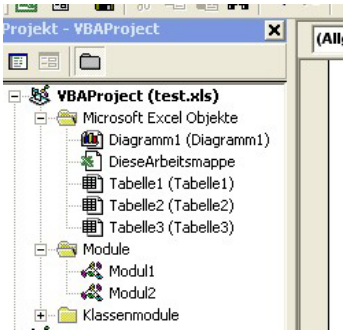
Visual Basic for Applications (VBA) is a programming language that facilitates working with tables, data and diagrams. VBA

- supports automatic processing and recording of repeated steps in the workflow.
- allows users to add own functionality to applications such as Vissim, Microsoft Excel or Access.
- allows users to start and control applications such as Vissim from Microsoft Office products (for example Excel or Access).

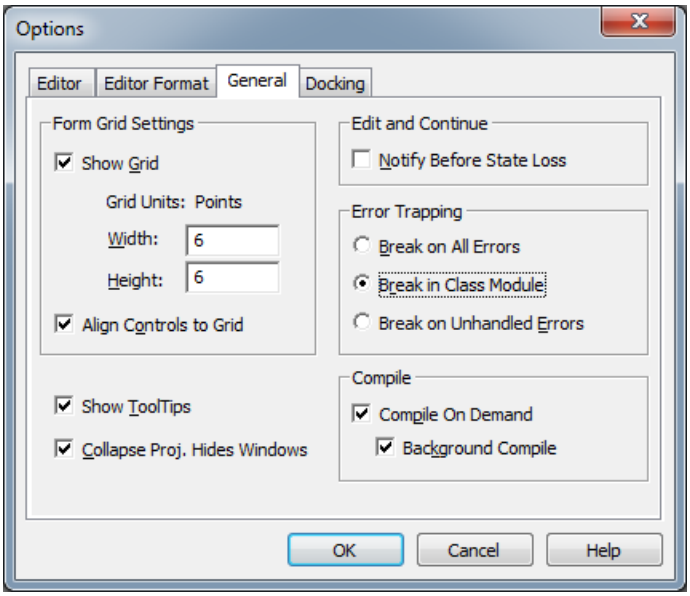
Since VBA is a “real” programming language, it will take a certain time and continuous training to become well experienced in VBA programming. Powerful programs will be worth the efforts finally, as saving time and safety in data processing are essential. VBA is provided with e.g. Microsoft Excel and 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 ...()“).

1.3 Microsoft Excel as container for VBA programs



If Microsoft Excel is used as container for a VBA program that activates Vissim via COM, the code can be integrated either in worksheets or in separate modules. If the code is part of a worksheet, it may happen that error messages are not shown detailed enough. This behavior is determined by a setting in the VBA editor which is accessed by TOOLS - OPTIONS - GENERAL:



We recommend the setting *Error trapping* to *Break in Class Module.* In case of an error at runtime, typically a specific error message is shown, and the VBA debugger points to the code line which caused the error.

If Error trapping is set to *Break on Unhandled Errors,* it may happen that a more general error message is shown. 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...“.

Calling very long COM statements in Microsoft Excel may trigger the following warning: “Microsoft Office Excel is waiting for another application to complete an OLE action”. In order to avoid this warning message, you may add the following line to your code:

```
Application.DisplayAlerts = False
```

Be aware that using this option may prevent useful warnings to be shown.

By default, the subscripts/indices of VBA arrays start at 0 (this is called the lower bound of the array). If you would prefer your array index numbers to start at 1, you can use the statement “Option Base 1”. However, the arrays returned from some COM functions (e.g. “GetMultiAttValues” or “GetMultipleAttributes”) do not support this setting and always will start at index 0.

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. We provide an installation package PTV Vision Python on our setup webpage: https://cgi.ptvgroup.com/php/vision-setups/.

We recommend installing the package under C:\Python and not under C:\Program Files\Python. 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/.

From PTV Vissim 2020 onwards, PTV Vissim supports Python from version 3.7. Currently supported version is 3.9.

- Vissim does not support the Python distribution ***anaconda***.
- Vissim does not support the Wrapper ***wxPython***.

Python provides many 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.

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)`

For the basic syntax see chapter 7.2 and the Python COM example at:
`…\Examples Training\COM\Basic Commands\`

1.5 Object interface identifiers

The name of the COM-Interface is the same as the Vissim class name plus the leading “I”. The leading “I” of an object (e.g. at ILink) stands for Interface. The leading “I” is discarded from the identifier when using object access through the Vissim object. For type declarations, the “I” is needed.

For example, addressing a Vissim link,

```
Vissim.Net.Links
```

is used, whereas declaring a link object is done by

```
Dim linkObj as ILink
```

Some programming languages (including VBA) offers to look up the correct identifier in the so-called object catalog.

2 Executing Scripts from within Vissim

There are two options to execute a script directly within Vissim, if it is written in VBS (Visual Basic Script), Java-Script or Python:

- Direct, single call of a script file: SCRIPTS - RUN SCRIPT FILE...
- Event-based scripts: SCRIPTS - EVENT-BASED SCRIPTS (see dedicated section below)

While for VBS scripts no additional software installation is required, for Python scripts it required a separate installation of Python.

We provide a separate setup for PTV Vision Python which you can download here: https://cgi.ptvgroup.com/php/vision-setups/. This installation will set up all required steps to run Python scripts from within PTV Vissim.

From PTV Vissim 2020 on, PTV Vissim supports Python version 2.7 and Python 3.7. You can switch between the two Python versions in the user preferences (Edit > User Preferences > Working Environment > Script Files).

In you manually want to install Python, the required steps to run a Python script are to install Python 2.7 or 3.7 (https://www.python.org/downloads/) and the directory of python.exe (e.g. C:\Python) has to be added to the Windows path in order to execute python.exe from any directory. This can be done in the “Environment Variables” settings of Windows. In addition, a compatible version of pyWin Build 218 or higher must be installed (http://sourceforge.net/projects/pywin32/files/pywin32/).

If the Python installation is located in a folder under User Account Control (UAC) (e.g. C:\Program Files or C:\Windows), PTV Vissim may need to run with administrator rights to create Python cache files in the Python folder. Therefore, we recommend installing Python in a non-UAC folder (e.g. C:\Python).

Access to Vissim Objects

The script does not need to start an additional Vissim instance as a COM server. To access any Vissim object, a predefined global variable `Vissim` is available. It points to the instance of Vissim where the script is currently executed. It allows accessing the currently loaded net and executing some recurring operations on it, for example executing several simulation runs, changing some parameters before each run etc.

Event-Based Scripts

Scripts can also be started automatically at specific times before, during and after a simulation run. Such scripts, along with their attributes, are defined in the Vissim list *Event-based scripts* which is accessible via SCRIPTS - EVENT-BASED SCRIPTS. Please see the Vissim online help or manual for details.

For event-based scripts, in addition to `Vissim`, two more predefined variables are available while a script is running:

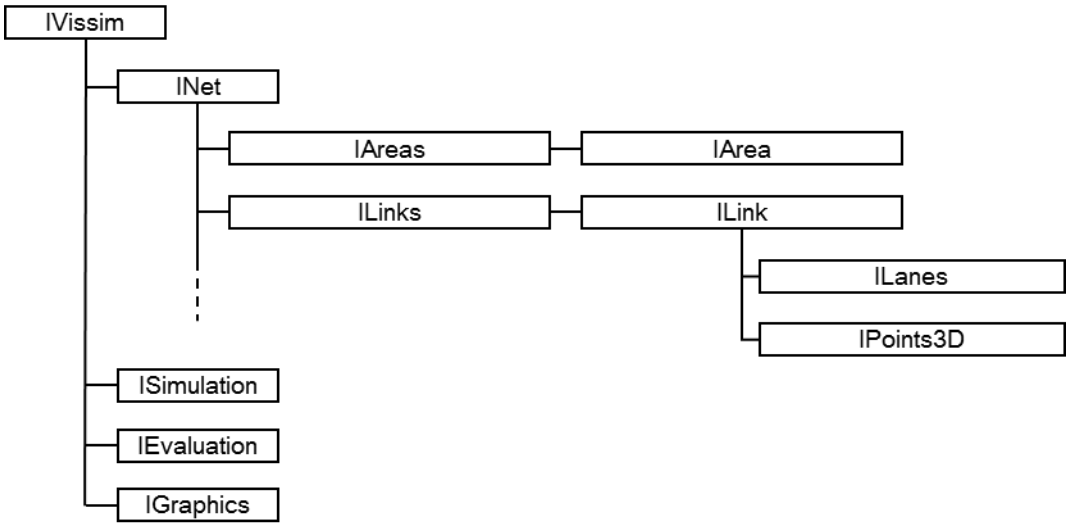
- `CurrentScript` returns the Vissim object of the currently running script. It allows direct access to all its attributes (including user-defined attributes). Example (VBS): `CurrentScript.AttValue("Period")`.
- `CurrentScriptFile` returns the filename of the script.

Example

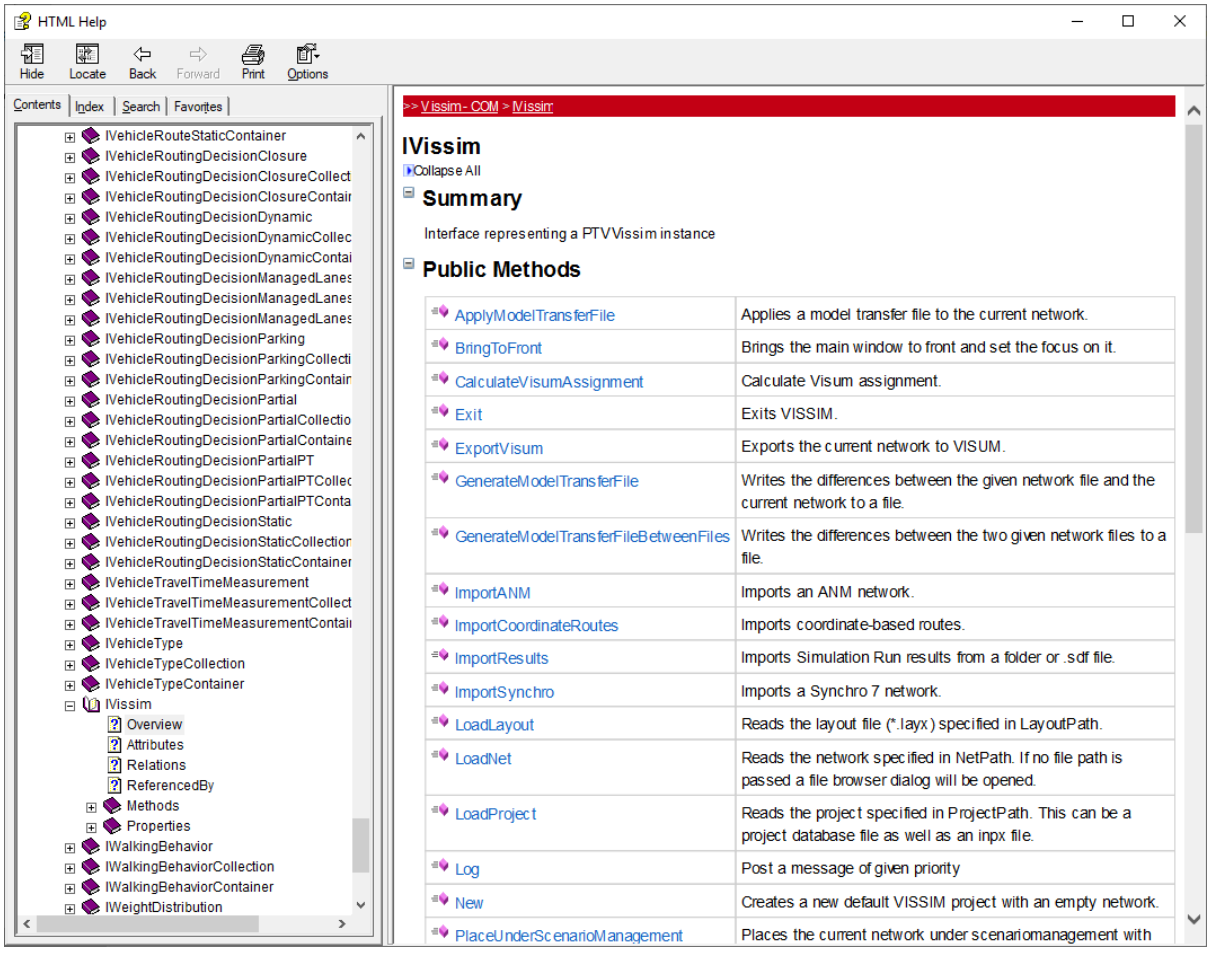
An example containing event-based scripts is provided with your Vissim installation in both VBS and Python languages at …\Examples Training\COM\Drop-off Zone

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 (COM Help) of Vissim for details about the various objects and their methods and properties:

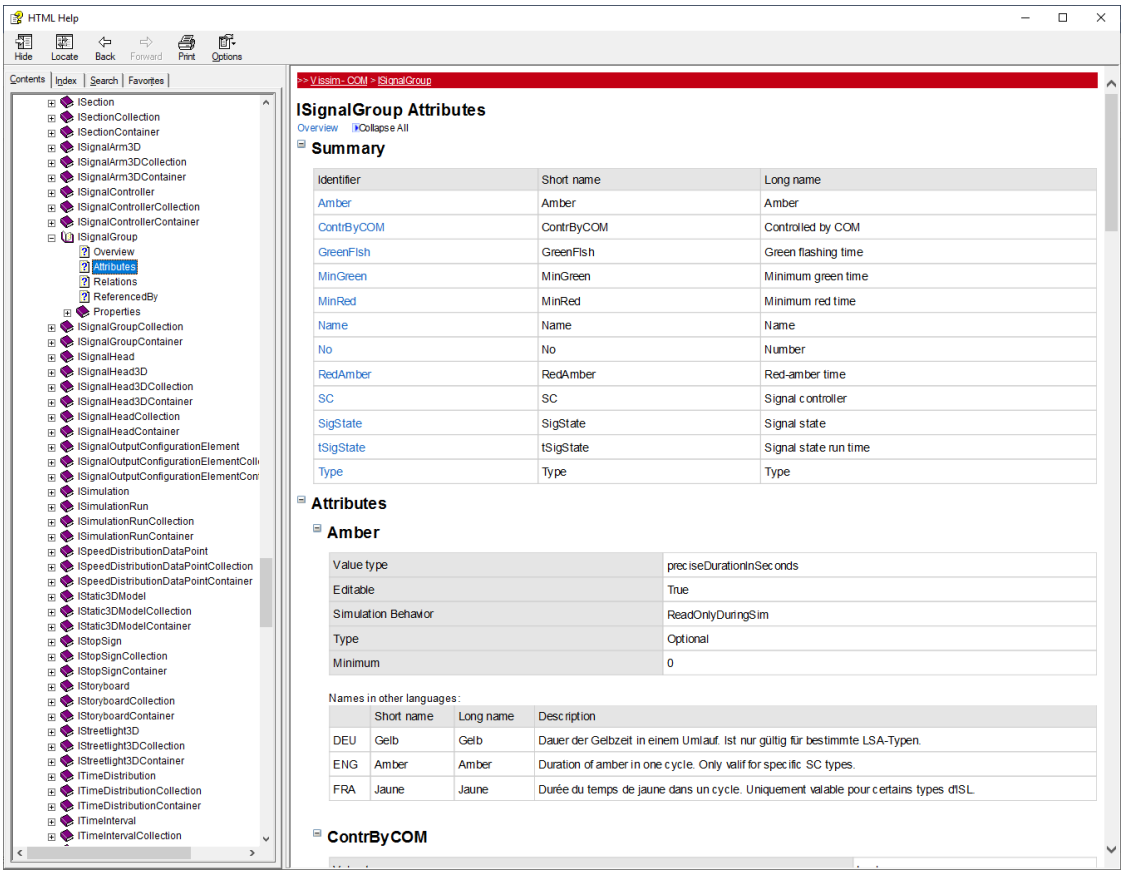


All attributes of an object are listed on the respective attribute page in the COM Help reference. For each attribute, the following information is shown:

- 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 the attribute can be edited through COM
- and the behavior of the attribute during the simulation:
  - EditableDuringSim (always editable)
  - ReadOnlyDuringSim (not editable during the simulation)
  - SimAttr (attribute exists only during the simulation) – it's editable if Editable = True
- 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)

The value type "color" uses a String with four double-digit hexadecimal numbers [00...FF] for Alpha (opacity), Red, Green and Blue. For example, for Green color and 100% opacity, the String would be: "FF00FF00": Alpha FF, Red 00, Green FF, Blue 00".

- 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)



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 2022 and Vissim 2023). 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.22") for Vissim 2022 or CreateObject ("Vissim.Vissim.23") for Vissim 2023. 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 2023)
   Set Vissim = CreateObject("Vissim.Vissim.23")

   '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) must 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 which means they represent several 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 and 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”.)

For several network objects both **collections** and **containers** exist. Containers contain the actual objects whereas collections contain only references to the objects, but not the objects themselves. This distinction is needed because objects are linked to one another.

> **Example:** The link sequence of a static vehicle route uses a collection of links. If you change the course of a route, the link sequence and thereby the link collection changes (some links may be added, some removed). The change of links in this collection does not affect the links in the network, the same links do still exist in the network. Only references to the links were changed in the link sequence of the route.

Through a collection, all objects of a specific type can be accessed in various ways: loops, iterators, and methods for manipulation of several attributes simultaneously. Random access to a specific object is possible only through the container using “ItemByKey” (see section below).

Please note that collections and containers may become invalid after the next simulation time step (for example because a vehicle leaves the network), so the iterators might not work anymore. Because of this, a container or collection should be retrieved from Vissim immediately before usage, with no simulation time steps in between.

The leading “I” of an object (for example at IVissim) stands for Interface. The name of the COM-Interface is the same as the Vissim class name plus the leading “I”.

You can use the COM methods `Vissim.SuspendUpdateGUI()` (before modifying attributes) and `Vissim.ResumeUpdateGUI()` (after the last modification) which stop respectively allow the updating of the complete Vissim workspace (network editor, list, chart and signal time table windows). Suspending the GUI update can save a lot of time during the execution of a COM script (if any windows are open) because otherwise each modification of an attribute of a static network object causes a complete redraw of the network and an update of these windows (even if the quick mode is active).

4.2.2 Filter

Filters are available for containers and collections to return only objects which meet a given criteria. Two options are available, and both return a collection of all objects for which the given formula returns true. But they differ in the volatility of the returned collection:

- **GetFilteredSet**: the returned collection will contain a fixed set of objects. That means, an object remains in the filter collection even if it does not match the filter condition at a later stage anymore.
- **FilteredBy**: the returned collection is subject to change: as soon as an object where the filter condition returned true at first is changed as such that the filter is not true anymore, the object is excluded from the filter collection and thus may lead to unexpected behavior.

The formula which defines the filter condition needs to be a character string similar as formulas defined in user defined attributes (UDA) of type ‘Formula’.

Example:

```
Set filteredVehicles = Vissim.Net.Vehicles.GetFilteredSet("[SPEED]<10")
```

4.2.3 Random Access to a Specific 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 that 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 Obj = Vissim.Net.SignalControllers.ItemByKey(42).SGs.ItemByKey(1)
State = Obj.AttValue("SigState")
```

For time intervals, the ItemByKey method is used in combination with enumeration. To access the time interval container of a specific type, the internal number of that type is

required. The enumeration list of all time intervals types is available in the COM Online Help, page “TimeIntervalSetNo Enumeration”.

Second time interval of Vehicle Inputs (enumeration 1):

```
Set TimeIntVehInput = Vissim.Net.TimeIntervalSets.ItemByKey(1).TimeInts
Set TimeIntNo2 = TimeIntVehInput.ItemByKey(2)
TimeIntNo2.AttValue("Start") = 500
```

Third time interval of Vehicle Routes Static (enumeration 2):

```
Set TimeIntVehStaRoute = Vissim.Net.TimeIntervalSets.ItemByKey(2).TimeInts
Set TimeIntNo3 = TimeIntVehStaRoute.ItemByKey(3)
TimeIntNo3.AttValue("Start") = 600
```

If you use Early Binding in your IDE (see chapter 6), you can use the enumeration object directly.

4.2.4 Simplified Handling of Objects Using References

References use so called keys to identify the associated objects. A key is an attribute or attribute combination which uniquely identifies a network object of a specific type. For every type of network object the attributes which form the key can be different. In almost any situation the key of a network object type is its "No" attribute. For network objects, which depend on other network objects a combination of keys might be necessary as it is the case for lanes. To refer to a lane you have to first give the key of the link and after that the number of the lane. The resulting key could look like "1-2" for the link with number 1 and the lane with number 2. In general you can refer to the list of the network object in Vissim for which you want to set the reference. For example, you could open the list for Stop Signs and see the Lane attribute, where the value could be "1 - 2". When using COM you can use the same key by just omitting the whitespaces.

References which refer to more than one network object are just comma separated lists of keys.

It is possible to use the references instead of the objects. However, getting an attribute of value type “Reference” using AttValue will always return a String and never the object itself.

Example:

```
'Using the object
Vissim.Net.VehicleInputs.ItemByKey(1).AttValue("Link") = Vissim.Net.Links.ItemByKey(2)
```

which is the same as:

```
'Using the reference (key)
Vissim.Net.VehicleInputs.ItemByKey(1).AttValue("Link") = 2
```

4.2.5 Access to all Objects of a Collection

To get or set attributes of multiple objects we recommend using the GetMulti / SetMulti functionality. For details, please refer to chapter 6.2.4 “Reading and Saving Attributes of Several Network Objects”.

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.

4.2.6 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 node
```

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 node
```

In VBA, the control variable of a `For Each` loop has to be declared as `Variant` or `Object`.

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
```

4.2.7 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 (Help > COM Help).

Attributes are always accessed by the method AttValue:

VBA:

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

For access to the attributes, only the (English) identifiers shown in the COM Help 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.

The method AttValue returns only three data types: Doubles, Integers and Strings. If the data type in Vissim is different (e.g. Boolean), the values are converted to one of these three types.

Attributes of unit's percent [%] are handled as values from 0 to 1 via COM (e.g. 0.45) although they are displayed as real percent (45%) in Vissim.

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 COM Help 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.

Result attributes can be saved for multiple simulation runs, time intervals and different vehicle classes. It is possible to access all saved result values by COM. To access these values three sub attributes are required:

Sub attribute #		Possible Input	Description
1	SimulationRun	`1,2,3, ...` or `Current`	number according to the attribute *No* of simulation runs (see *Simulation Runs* list) or `Current` for the simulation run currently running
		`Avg, StdDev, Min, Max`	aggregated value of all simulations runs
2	TimeInterval	`1,2,3 ...` or `Last`	index of one specific time interval (the index for time intervals always starts at 1 which refers to the first time interval) or `Last` to reference to the last completed time interval.
		`Avg, StdDev, Min, Max`	aggregated value of all time intervals of one simulation
		`Total`	sum over all time intervals of one simulation
3	VehicleClass	`10,20, ...` or `All`	one or more vehicle class numbers (according to the attribute *No* of *Vehicle Classes*) or `All`. Data is shown only for the vehicle classes defined here.

For some attributes, enumeration types are used. One example is the signalization state. Each state has a name and a value, for example SignalizationStateGreen equals value 3.

To change the signal state attribute of a signal group, you can make use of the value, the string of the enumeration or the enumeration object. Note: Using the enumeration object requires early binding (see chapter 6). All enumeration types are listed in the COM Help.

Example:

```
'Using the value
SignalGroup.AttValue("SigState") = 3
```

which is the same as:

```
'Using the enumeration string
SignalGroup.AttValue("SigState") = "GREEN"
```

and the same as:

```
'Using the enumeration
SignalGroup.AttValue("SigState") = SignalizationState.SignalizationStateGreen
```

Note: PTV Vissim returns mainly the String value as it easier to interpret.

You can see some examples of how to access result attributes at the example Basic Commands:

…\Examples Training\COM\Basic Commands\

4.2.8 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 chapter 6.3.

In order to run fast simulations, disable the visibility of all dynamic elements. Therefore simply activate Quick Mode:

Vissim.Graphics.CurrentNetworkWindow.AttValue("QuickMode") = 1.

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 `VISSIM230.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 „Vissim Vissim = new Vissim()“;.

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_Count“.

Furthermore, in C# you might have to specify optional parameters explicitly, which means there are no optional parameters. For example for loading an 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
{
   class NameReader
   {
       /// <summary>
       /// The main entry point for the application.
       /// <summary>

       static void Main(string[] args)
       {
           Vissim Vissim = new Vissim();
           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();
       }
   }
}
```

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)
```

For the basic syntax see chapter 7and the C# COM example at:
…\Examples Training\COM\Basic Commands\

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 (https://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.

For the basic syntax see chapter 7 and the Java COM example at: …\Examples Training\COM\Basic Commands\

In this folder there is a manual provided of how to set up the JaCoB COM bridge in Eclipse (COM Example - Java.pdf).

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 must be connected to Vissim objects explicitly. This provides access to the Vissim data model. After use these connections must 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 2023\\Exe\\Vissim230.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");
   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.

C++ example output:

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

For the basic syntax see chapter 7and the C++ COM example at:
…\Examples Training\COM\Basic Commands\

5.4 COM in MATLAB

MATLAB is a programming language and numerical computing environment. With MATLAB it is possible to control Vissim COM objects as easy as with VBA or Python. To start Vissim from MATLAB use the expression: „Vissim = actxserver('Vissim.Vissim')". Once Vissim is started, all methods, properties and object models are shown using the variable "Vissim".

MATLAB example:

```
Vissim = actxserver('Vissim.Vissim'); % Start Vissim
Vissim.LoadNet('D:\Date\example.inpx');
LinkNo = Vissim.Net.Links.GetMultiAttValues('No');
LinkName = Vissim.Net.Links.GetMultiAttValues('Name');
for i = 1 : Vissim.Net.Links.Count,
disp(['Link',32,num2str(LinkNo(i)),32,'(',LinkName{i},')']) % char(32) is whitespace
Vissim.release; % End Vissim
```

This example loads an input file and displays the numbers and the names of all links to the Command Window. This is exactly what the C# example does. Therefore the possible output is the same.

By default MATLAB passes 2D arrays to COM. PTV Vissim always expects 1D Arrays (SafeArray[0], SafeArray[1], ..., instead of SafeArray[0,0], SafeArray[0,1], ...). You can change the behavior in MATLAB with the command:

```
feature('COM_SafeArraySingleDim', 1)
```

For the basic syntax see chapter 7.5 and the MATLAB COM example at:

`...\Examples Training\COM\Basic Commands\`

6 Vissim Examples in VBA

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 OPTIONS - 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")

    '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")

   '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 Different Methods to Access Objects and Attributes in VBA

In an example five different methods to access Vissim objects and attributes with VBA will be shown. The Framework opens Vissim, loads a Vissim network and then calls one of the five subroutines. The subroutines are described in the following chapters 6.2.1.1 to 6.2.1.5. In order to run the code, copy the framework and all subroutines one after another in a VBA module.

Framework:

```
'declaration of variables
Public Vissim As Object            ' Variable Vissim
Public VehicleInput As Variant

Public Sub Framework()
    Dim No_programm As Integer

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

    '--------------------------------------

    No_programm = Cells(3, 2)      ' Programm No. is chosen in cell B3
    Select Case No_programm
        Case 1
            '6.2.1.1   Finding vehicle inputs using ItemByKey
            VehicleInputAttributGetItemByKeyExample
        Case 2
            '6.2.1.2   Loop using Object Enumeration
            VehicleInputAttributeEnumerationExample
        Case 3
            '6.2.1.3   Loop using Item
            VehicleInputAttributeItemExample
        Case 4
            '6.2.1.4   Loop via For...Each
            VehicleInputAttributeForEachExample
        Case 5
            '6.2.1.5   Loop using Iterator Object
            VehicleInputAttributeIteratorExample
    End Select

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

End Sub
```

6.2.1.1 Find a Vehicle Input 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: VehicleInputAttributGetItemByKeyExample

```
'This example shows how to access single vehicle inputs of a network
'using ItemByKey
Public Sub VehicleInputAttributGetItemByKeyExample ()
    '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")
End Sub
```

6.2.1.2 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
Public Sub VehicleInputAttributeEnumerationExample()
    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
End Sub
```

6.2.1.3 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).
Public Sub VehicleInputAttributeItemExample()
    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
End Sub
```

6.2.1.4 Loop Using 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.
Public Sub VehicleInputAttributeForEachExample()
      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
End Sub
```

6.2.1.5 Loop Using Iterator Object

The example demonstrates a loop that is set up using the Iterator 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.
Public Sub VehicleInputAttributeIteratorExample()
    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
End Sub
```

6.2.2 Reading and Saving Individual Attributes

The following example shows the access for reading and writing attributes. Firstly 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% for the first time interval. The time interval must be specified together with the attribute name “Volume”. This is 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 for some attributes you might have to pass a different value than seen in the GUI. For time intervals for example you need provide the index rather than the time, such as “Volume(1)” to refer to the first time interval. Using the start time of the time interval as sub attribute will not work. For more information on this please refer to chapter 4.2.7.

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")
   '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
```

6.2.3 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 must 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")
   '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
```

6.2.4 Reading and Saving Attributes of Several Network Objects

To read and save attributes of several network objects there are several methods available. Accessing multiple objects using AttValue() needs a more complex coding and is slower. Better use the following methods to access multiple attributes.

GetMultiAttValues

Get the attributes of all network objects of one network object type.

Example: Getting the names of all links:

```
Vissim.Net.Links.GetMultiAttValues("Name")
```

SetMultiAttValues

Set the attributes of several network objects of one network object type.
Example: Setting the names of link number 1 and 3:

```
Dim newNames(1, 1) As Variant
newNames(0, 0) = 1              'reference (key) of link
newNames(0, 1) = "newLink1name" 'new name for link #1
newNames(1, 0) = 4              'reference (key) of link
newNames(1, 1) = "newLink4name" 'new name for link #2
Vissim.Net.Links.SetMultiAttValues "Name", newNames
```

Parameter 1: name of attribute, here: "Name"
Parameter 2: new values, array with first column reference to network object and second column new value (here name)

GetMultipleAttributes

Get several attributes of all network objects of one network object type.
Example: Getting the names and the lengths of all links:

```
Vissim.Net.Links.GetMultipleAttributes(Array("Name", "Length2D"))
```

SetMultipleAttributes

Set several attributes of all network objects of one network object type.
Example: Set the attributes "name" and "cost per km" of all links:

```
Dim Values(3, 1) As Variant
Values(0, 0) = "name1"
Values(1, 0) = "name2"
Values(2, 0) = "name3"
Values(3, 0) = "name4"
Values(0, 1) = 12
Values(1, 1) = 7
Values(2, 1) = 5
Values(3, 1) = 3
Vissim.Net.Links.SetMultipleAttributes Array("Name", "CostPerKm"), Values
```

Parameter 1: names of attributes, here: `("Name", "CostPerKm")`
Parameter 2: new values, array containing the new values of attributes in the same order than the attributes names given in parameter 1.

The number of the network object is no input in this method. The first values in parameter 2 (here: `"name1" and 12`) are set to the network object with the lowest number. The following values are set to network objects with increasing number.

Example: VehicleInputNameAndVolumeExample (VBA)

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 volumes are calculated. The new attribute values are set via SetMultipleAttributes. The result of this operation is the same as in 6.2.2. 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.

```
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")
    '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")
   '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 cannot 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 must 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")
  '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 "EditableDuringSim" or "SimAttr" and "Editable" = True 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")
  '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
```

7 Basic Commands in Various Programming Languages

This chapter provides some basic commands for controlling Vissim via the COM interface for VBA, Python, MATLAB®, C++, C# and JAVA. For each program language, the following commands are provided:

- start Vissim
- open a network file
- save a network file
- run a simulation
- access a network object
- read and set attributes
- close Vissim

COM Examples

There are examples provided for different programming languages demonstrating the basic syntax of how to control PTV Vissim by COM. The examples are located in …\Examples Training\COM\Basic Commands\.

The COM examples do exist for the following programming languages:

- VBA File: COM Basic Commands.bas
- Python, File: COM Basic Commands.py
- C++, File: COM Basic Commands.cpp
- C# File: COM Basic Commands.cs
- Matlab File: COM Basic Commands.m
- Java File: COM Basic Commands.java, COM Basic Commands - Utils.java

For further information, see the description file COM Basic Commands Desc_ENG.pdf or the comments in the corresponding files. You can read all files by using a simple text editor.

7.1 VBA

Action	VBA Command
Create COM server	`Set Vissim = CreateObject("Vissim.Vissim")`
Create COM server with specific Vissim version	`Set Vissim = CreateObject("Vissim.Vissim.23")`
Load network	`Vissim.LoadNet(Filename)`
Save network	`Vissim.SaveNetAs Filename`
Run simulation single step	`Vissim.Simulation.RunSingleStep`
Run simulation continuously	`Vissim.Simulation.RunContinuous`
Access specific object	`Set ObjLink10 = Vissim.Net.Links.ItemByKey(10)`
Read attribute	`... = Obj.AttValue("...")`
Set attribute	`Obj.AttValue("...") = …`
Close COM server	`Set Vissim = Nothing`

7.2 Python

Action	Python Command
Create COM server	`import win32com.client as com`
Create COM server with specific Vissim version	`Vissim = com.gencache.EnsureDispatch("Vissim.Vissim")` `Vissim = com.gencache.EnsureDispatch("Vissim.Vissim.23")`
Load network	`Vissim.LoadNet(Filename)`
Save network	`Vissim.SaveNetAs(Filename)`
Run simulation single step	`Vissim.Simulation.RunSingleStep()`
Run simulation continuously	`Vissim.Simulation.RunContinuous()`
Access specific object	`ObjLink10 = Vissim.Net.Links.ItemByKey(10)`
Read attribute	`... = Obj.AttValue('…')`
Set attribute	`Obj.SetAttValue('…', ...)`
Close COM server	`Vissim = None`

7.3 C++

Action	C++ Command
Initialize COM	`#import "C:\\Program Files\\PTV Vision\\PTV Vissim 2023\\Exe\\Vissim230.exe" rename_namespace ("VISSIMLIB")` `CoInitialize(NULL);`
Create COM server	`VISSIMLIB::IVissimPtr Vissim;` `Vissim.CreateInstance("Vissim.Vissim");`
Create COM server with specific Vissim Version	`Vissim.CreateInstance("Vissim.Vissim.23");`
Load network	`Vissim->LoadNet(Filename, false);`
Save network	`Vissim->SaveNetAs(Filename);`
Run simulation single step	`Vissim->GetSimulation()->RunSingleStep();`
Run simulation continuously	`Vissim->GetSimulation()->RunContinuous();`
Access specific object	`Vissim->GetNet()->Links->GetItemByKey(10);`
Read attribute	`Obj->GetAttValue("...")`
Set attribute	`Obj->PutAttValue("...", "...");`
Close COM server	`Vissim.Detach()->Release();`
Uninitialize COM	`CoUninitialize();`

7.4 C#

In the development environment used, set a reference to the Vissim executable file installed (currently this is VISSIM230.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.

Please note that embedded Interop types are not supported. Make sure to set the Embed Interop Types property of VISSIMLIB to false, otherwise your code can not be compiled.

Action	C# Command
Initialize COM	`using System;` `using VISSIMLIB;`
Create COM server	`VissimClass Vissim = new VissimClass();`
Load network	`Vissim.LoadNet(Filename, false);`
Save network	`Vissim.SaveNetAs(Filename);`
Run simulation single step	`Vissim.Simulation.RunSingleStep();`
Run simulation continuously	`Vissim.Simulation.RunContinuous();`
Access specific object	`ILink ObjLink10 =` `(ILink)Vissim.Net.Links.ItemByKey(10);`
Read attribute	`... = (var)Obj.AttValue["..."];`
Set attribute	`Obj.set_AttValue("...", ...);`
Close COM server	`Vissim.Exit();`

7.5 MATLAB

By default MATLAB passes 2D arrays to COM. PTV Vissim always expects 1D Arrays (SafeArray[0], SafeArray[1], ..., instead of SafeArray[0,0], SafeArray[0,1], ...). You can change the behavior in MATLAB with the command:

```
feature('COM_SafeArraySingleDim', 1)
```

Action	MATLAB Command
Create COM server	`Vissim = actxserver('Vissim.Vissim');`
Create COM server with specific Vissim Version	`Vissim = actxserver('Vissim.Vissim.23');`
Load network	`Vissim.LoadNet(Filename)`
Save network	`Vissim.SaveNetAs(Filename)`
Run simulation single step	`Vissim.Simulation.RunSingleStep`
Run simulation continuously	`Vissim.Simulation.RunContinuous`
Access specific object	`ObjLink10 = Vissim.Net.Links.ItemByKey(10)`
Read attribute	`Obj.AttValue('…')`
Set attribute	`set(Obj, 'AttValue', '...', ...);`
Close COM server	`Vissim.Exit; Vissim.release`

7.6 JAVA

This commands uses jacob (Link) to access the COM Interface. After downloading jacob, copy the jacob-*.dll file to c:\windows\system32 and add jacob.jar to the library of your java project.

Action	JAVA Command
Initialize COM	`import com.jacob.com.*;` `import com.jacob.activeX.*;`
Create COM server	`ActiveXComponent Vissim = new ActiveXComponent("VISSIM.Vissim");`
Create COM server with specific Vissim Version	`ActiveXComponent Vissim = new ActiveXComponent("VISSIM.Vissim.23");`
Load network	`Vissim.invoke("LoadNet", Filename);`
Save network	`Vissim.invoke("SaveNetAs", Filename);`
Run simulation single step	`Vissim.invokeGetComponent("Simulation").invoke("RunSingleStep");`
Run simulation continuously	`Vissim.invokeGetComponent("Simulation").invoke("RunContinuous");`
Access specific object	`ActiveXComponent ObjLink10 = Vissim.invokeGetComponent("Net").invokeGetComponent("Links").invokeGetComponent("ItemByKey", new Variant(linkNumber));`
Read attribute	`Obj.invoke("AttValue", "..."); // Obj is class ActiveXComponent`
Set attribute	`Dispatch.invoke(Obj, "AttValue", Dispatch.Put, new Object[]{"...", "..."}, new int[1]);`
Close COM server	`Vissim.safeRelease();`