


PTV VISUM

the mind of movement



INTRODUCTION TO THE PTV VISUM COM-API

Copyright:

© 2013 PTV AG, Karlsruhe

PTV Visum® is a trademark of PTV AG

PTV Vissim® is a trademark of PTV AG

All brand or product names in this documentation are trademarks or registered trademarks of the corresponding companies or organizations. All rights reserved.

Disclaimer:

The information contained in this document is subject to change without notice and should not be construed as a commitment on the part of the vendor.

This document may not be used for any other purpose than the personal use of the purchaser.

No part of this handbook may be reproduced, stored in a retrieval system, or transmitted, in any form, or by any means, electronic, mechanical, photocopying, recording, or otherwise, edited or translated, except as permitted under the terms of the copyright, without the prior written permission of PTV AG.

Impressum:

PTV AG

Traffic Software

Haid-und-Neu-Straße 15

D - 76131 Karlsruhe

Germany

Phone +49 721 9651-300

Fax +49 721 9651-562

E-Mail: info.vision@ptv.de

www.ptvag.com

www.ptv-vision.com

© 2013 PTV AG, Karlsruhe

Table of contents

Preamble	6
1 Introduction to COM programming	7
1.1 General information on COM programming	7
1.1.1 What is COM?	7
1.1.2 Visual Basic dialects (VBA, VBS)	7
1.1.3 Excel as Container for VBA-programs	8
1.1.4 Python	8
1.1.5 A remark on object interface identifiers	9
2 Extending PTV Visum through scripts and Add-Ins using the COM-Interface	10
2.1.1 Including scripts into the Visumscript menu	10
2.1.2 Calling scripts in procedure seuquences	10
2.1.3 Add-Ins	10
2.1.4 Important Change to Python Script Execution	12
2.1.5 Debug Python from Visum 12 onwards	13
3 The Visum object model	15
4 First Steps	28
4.1 Access to Visum objects and files	28
4.2 Access to objects and attributes	29
4.2.1 Introduction	29
4.2.2 Random access to a specified object using its key	30
4.2.3 Access to all objects of a collection	30
4.2.4 Loops over all objects of a collection	30
4.2.5 Read and Save attributes	32
4.2.6 Execute Assignments	32
5 COM in other programming languages	33
5.1 COM in C#	33
5.2 COM in JAVA	34
5.2.1 Case 1: Microsoft Development Environment (J#)	34
5.2.2 Case 2: "Pure" JAVA Application	36
5.3 COM in C++	36
6 Visum Examples	38
6.1 Reading and Saving Data	38
6.1.1 Creating Visum Objects and Reading the Version File	38
6.1.2 Reading and Saving the Version File	39
6.1.3 Reading Network File Additively	39
6.2 Access to Objects and Attributes	40
6.2.1 Finding nodes using ItemByKey	40
6.2.2 Loop using Object Enumeration	41
6.2.3 Loop using Item	41

6.2.4	Loop via For...Each	42
6.2.5	Loop using Iterator Object	43
6.2.6	Reading and Saving Individual Attributes	43
6.2.7	Reading and Saving Attributes of Several Network Objects	44
6.2.8	Invalid Attribute Values	45
6.2.9	Creating a Network	46
6.2.10	Creating OD Connector Links	48
6.3	Assignment	50
6.3.1	Executing Assignments from the Assignment Parameters File	50
6.4	Lists	51
6.4.1	Creating a List and Exporting Data as an Attribute File	51
6.4.2	Route Analysis	51
6.5	Filters	53
6.5.1	Defining Filters via COM	53
6.6	Reports and Analyses	54
6.6.1	Flow Bundles	54
6.6.2	A Slightly More Complicated Flow Bundle	55
6.7	Matrix Operations	55
6.7.1	Symmetrizing Matrices with Python	55
6.7.2	Calculating Values for the Diagonal Entries of a Skim Matrix	56
6.8	Dialogs and Add-Ins	56
6.8.1	Selecting Graphic Parameters	56
6.8.2	Access to GPA via COM	57
6.9	Add-in components	58
7	The Python Library VisumPy	61
7.1	Introduction	61
7.2	Helpers Module	61
7.3	TISupport Module	68
7.4	Deprecated: Tk Module	70
7.4.1	NetobjSelector Class	71
7.4.2	ProgressDlg Class	72
7.4.3	Other Functions	73
7.5	wxHelpers Module	73
7.5.1	wxAttrIDButton Class	74
7.5.2	wxNetobjtypeCombo Class	75
7.5.3	wxNetobjCombo Class	76
7.6	Excelplot Module	77
7.6.1	Chart Class	77
7.7	Excel Module	78
7.8	Reports Module	79
7.9	Matrices Module	80
7.10	csvHelpers Module	85
7.11	AddIn Module	86
7.11.1	AddIn Class	86
7.11.2	AddInParameter Class	90

8	Matrix Editor Interface Script Muuli	93
8.1	Hints for the use of the matrix editor COM interface	93
8.2	General Muuli Methods	94
8.3	Muuli application Modes	94
8.4	Loading a Matrix from/Saving a Matrix to the Main Storage	94
8.5	Query/Set Properties and Values of Loaded Matrix	95
8.6	Executing Operations on a Loaded Matrix	97
8.7	Creating a Matrix	108
8.8	Matrix Functions	108
8.9	Muuli Examples	115
8.9.1	Creating separate Muuli COM Object	115
8.9.2	Creating Muuli COM Object embedded in VISUM	115
8.9.3	Destination-Coupled Gravitation Model	116
8.9.4	Calibri Method	116
8.9.5	Aggregating a Matrix without Optional Parameters	117
8.9.6	Aggregating a Matrix with Optional Parameters	117

Preamble

Starting with Visum 13, the reference documentation of the COM-interface of Visum is no longer included with this document. Instead, it is provided in a more accessible form as part of the Visum Online Help system. This document contains the conceptual parts of the former COM-documentation, some examples and a reference of the VisumPy library of Python modules shipped with Visum.

A table of all attributes and their properties is given in the ATTRIBUTE.XLS file and the attributes and relations of all object types are also listed in the 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. From Visum version 7.0 on data and algorithms contained in Visum can be accessed via the COM interface using Visum as automation server. The Visum COM interface is automatically included when the software Visum is installed.

COM does not depend on a certain programming language. COM Objects can be included in a wide range of programming and scripting languages reaching from VBA via Python and C++ to Delphi. In the following examples VBA is used most often. Exceptions using the programming language Python are marked explicitly.

Since Visum Version 9.3 scripts written in VBS can be called directly from Visum. For this purpose the script menu is supplemented by entries, each corresponding to a script and assigned to a shortcut.

Since Version 9.4 the possibilities of using the COM interface have been enlarged once more. Beside scripts written in VBS it is now possible to execute scripts written in the programming language Python directly from the Visum script menu. Since Python allows for creating graphical user interfaces in a simple way, it is now possible to add individual dialogues to the Visum software. Besides, script files can now be executed as a procedure step.

By introducing add-ins as an additional architecture element, Visum 11 now allows users to easily develop scripts that interact with Visum via a defined process. The advantage of these add-ins is that they can be transferred more easily, since to install and use them users do not have to know how to write script applications.

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 Visum, Excel or Access.
- allows less experienced users to set up a workplace that only contains the menus and commands they actually need.
- allows users to start and control applications as e.g. Visum 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, thus it should be available on almost any personal computer. This paper 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 storing 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 using Excel as Container for VBA-programs activating Visum 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 version file, VBA shows the error message “automation error”. If the same code is located in a module, VBA produces the correct error message „CVisum::LoadVersion failed! File...“.

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 please add the following line to your code:

```
Application.DisplayAlerts = False
```

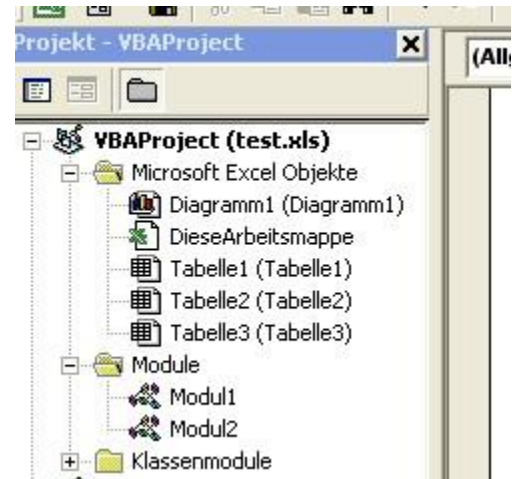


Figure 1: VBA project elements in Excel

1.1.4 Python

Python provides a second, very powerful programming language for direct execution in the Visum context. Users can execute Python scripts from the Visum script menu, or use the *Execute script* procedure that is performed within an automated run. It is also possible to start a Python script manually in the Python interpreter. This script can create instances of Visum and connect to them via COM.

In contrast to VBA forthcoming 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. Including such scripts into the script menu results in new, individual dialogues that appear as part of Visum. A simple, but useful example is shown in Appendix B (6.8.1).

Remind that Python supports properties only in case the Set variant of the property takes exactly one parameter. In all other cases, properties become separate Get and Set methods, where the latter receives an additional “Set” prefix. The most prominent example is the property AttValue which becomes in Python two separate methods:

```
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 A remark on object interface identifiers

Within different programming languages the object interfaces appear differently. The difference consists of the leading "I". Using any VB-dialect (VBA, VBS), the leading "I" has to be 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.

2 Extending PTV Visum through scripts and Add-Ins using the COM-Interface

2.1.1 Including scripts into the Visumscript menu

You can execute scripts written in VBS or Python directly from VisumVisum. To do so, click EDIT SCRIPT MENU... to enter a script menu in the corresponding box. Assign a script file (including path). Optionally, you can create a shortcut to this entry. Entries are optionally visible either for all users or for the current user only.

A predefined variable „Visum“ is passed to the script menu if started via the script menu. It points to the instance that calls VisumVisum. You do not have to start an additional Visum instance as a COM server. By this it is possible e.g. to access the currently loaded net and execute some recurring operations on it. Other status information as the graphic parameters can be changed by the script as well.

Executing a script via the script menu entry has the same effect as using the EXECUTE SCRIPTS+SCRIPT FILE menu or the *Run script* procedure. You cannot pass parameters to the script.

2.1.2 Calling scripts in procedure sequences

The procedure “Run script” allows to execute a script written in any of the supported scripting languages within a procedure sequence. The script code can either be provided as an external file, or for shorter scripts it can be directly entered in the procedure parameters. As with scripts executed from the script menu, these scripts get access to parenting Visum instance through a predefined variable “Visum”. It is not possible to pass any parameters to scripts invoked by this mechanism

2.1.3 Add-Ins

Add-ins are scripts that meet special conventions. The only script language allowed is Python. Contrary to scripts, add-ins are parametrizable and automatically show up in Visum after they have been saved to a defined location in the file system and Visum has been started. They thus appear as a user-defined, extended Visum functionality that you can use without having any programming knowledge.

Add-ins normally consist of two or several files. If it was copied to a sub-folder of %APPDATA%\ADDINS, Visum will find the add-in and read the information required to automatically show it in the script menu. Optionally, you can also show add-ins in the procedure dialog, as a possible procedure.

If the add-in is available in a parameter dialog, the latter is shown in the Visum procedure dialog as a parameter dialog of the procedure step. When executing add-ins from the script menu, you can also first set parameters in the dialog. If there are several procedure steps that execute the same add-in, every step has its own, independent set of parameters.

From Visum version 11, add-ins are shipped with the installation CD.

2.1.3.1 Add-in architecture

From a technical point of view, an add-in is a collection of files that has been saved to the same folder (and its sub-folders). A text file with the extension *.VAI is obligatory. It is an xml file that contains information about the add-in. The *.VAI file also contains the file names of the relevant Python files that contain the actual program code. It contains the following tags:

Tag	Data type	Description
AddInID	String	A unique ID that allows you to identify an add-in within an installation
AddInName	String	Name of add-in for output to the Visum program interface
AddInPath (optional)	String	Hierarchical path for placement of add-in in the script menu and in the procedure dialog
AddInParaDlgScript (optional)	Filename	Name of Python file that contains the code for display of a parameter dialog. The file must be in the same folder as the *.VAI file. If this parameter is missing, the display of a parameter dialog fails (as AddInAutoShowPara = False)
AddInBodyScript	Filename	Name of the Python file that contains the code for execution of the add-in. The file must be in the same folder as the *.VAI file.
AddInOperation	Bool	If true, the add-in is automatically inserted into the list of procedures.
AddInAutoMenu	Bool	If true, the add-in is automatically inserted into the script menu.
AddInAutoShowPara	Bool	If true, the add-in is executed from the script menu and first shown in the parameter dialog.

Table 1: Tags in the *.VAI file

The actual add-in code is in two script files that can integrate further modules as common in Python. When executed, one of the scripts only shows a parameter dialog. The values requested are saved in a special object called *Parameter* that is given to the script from outside, analogous to the Visum object. The parameter object has two members: *Parameter.OK* is a truth value that shows that the parameter object contains logical data. *Parameter.Data* contains a string that encodes the parameter settings. This may be a string created by the Python *pickle* module, or an xml or other representation of the current parameters.

The second script implements the functionality of the add-in. To do so, it can use the parameters it retrieves from the *Parameter* object. If *Parameter.OK* is true, the *Parameter.Data* value that contains the parameter settings may be retrieved and evaluated. It can influence the script execution.

If an add-in is integrated in a procedure, a parameter object is saved to the Visum network for each procedure step that is realized using the add-in.

2.1.3.2 Methods of the IAddInParameter objects

The IAddInParameter object is used to exchange the parameters of an add-in between Visum and the add-in code. This object has the following properties:

Data ([in] BSTR parameter)

Saves data as a string to the parameter object.

Data ([out, retval] BSTR *parameter)

Returns the data that was last saved as a string.

OK ([in] VARIANT_BOOL ok)

Sets the truth value that shows whether the parameter object data is valid, i.e. if it was written by the parameter dialog script of the add-in. If this value is true, the body script may retrieve and use the parameters.

OK ([out, retval] VARIANT_BOOL *ok)

Returns the truth value that shows whether the parameter object data is valid.

Example: 6.9

2.1.4 Important Change to Python Script Execution

With Visum 11.51 we introduced an important change to Python script execution from within Visum. This change breaks existing scripts which use wxPython as the graphical user interface, so you need to adapt them. We are forced to make this change, because an exotic, and so far overlooked, behavior of wxPython leads to instability in Visum when scripts are executed in the old way.

From Visum 11.51 onwards Visum itself takes care of creating the wxPython application object and running its main loop. This application object must remain the only wxPython application during the entire lifetime of the Visum session. Python scripts which use a wxPython GUI reuse this central application object and only create new windows within it.

To be compatible with this new philosophy you will need to modify your own existing Python scripts if they use a wxPython GUI. Parameter dialog scripts within an add-in written by yourself belong in this category. PTV already adapted all add-ins and other Python scripts which are distributed with the Visum 11.51 installation, so if you never wrote your own scripts, you can safely ignore this note and just continue using the pre-installed scripts.

Here is what you need to do:

- 1) Remove the instruction which creates the wxPython application object. Typically this instruction takes the form

```
app = wx.PySimpleApp(0)
```

The central PySimpleApp object created by Visum is available for subsequent calls in a predefined variable "app". So you can still use an instruction like

```
app.SetTopWindow(mydiag)
```

without creating app yourself.

- 2) Remove the instruction which runs the main loop. Typically this instruction takes the form

```
app.MainLoop()
```

- 3) Never use force to exit from your Python script. Remove all calls to sys.exit() and terminate normally.

- 4) **Only for parameter dialogs in add-ins:** There is no way to reliably detect from the outside whether you are through with getting parameter input from the user. Therefore in each program path that can end your parameter dialog, you should call

```
Terminated.set()
```

Typically, these are the OnOK and OnCancel event handler of the parameter dialog. The variable Terminated is predefined by Visum. If you fail to call Terminated.set(), the dialog may close, but Visum hangs instead of executing the add-in body script.

Example: The code snippet below, taken from the parameter dialog script of the BalanceMatrix add-in illustrates where you need to make the changes:

```
import wx
# ... other stuff

class MyDialog(wx.Dialog):
# ... other stuff

    def OnCancel(self, event): # wxGlade: MyDialog.<event_handler>
        Terminated.set() # <----- added to signal completion of dialog
        self.Destroy()

    def OnOK(self, event): # wxGlade: MyDialog.<event_handler>
        #... other stuff to save parameters
        Parameter.OK = True
        Terminated.set() # <----- added to signal completion of dialog
        self.Destroy()

# end of class MyDialog

wx.InitAllImageHandlers()
# <----- app = wx.PySimpleApp(0) removed
dialog_1 = MyDialog(None, -1, "")
app.SetTopWindow(dialog_1)
dialog_1.Show()
# <----- app.MainLoop() removed
```

2.1.5 Debug Python from Visum 12 onwards

From Visum 12 onwards, the libraries required to develop Python scripts with Visum are no longer located in the Python site packages folder, but in the Visum installation ...\\Visum125\\Exe\\PythonModules\\. This change enables versioning of the libraries. Each installed Visum version now has its own Python libraries. When updating your own Python packages, the Python packages installed for Visum remain unchanged. If you wish to use the Python libraries of a Visum installation in your script, you will have to adapt your Python script or add-in. To access the correct Python packages, you must complete the following steps:

- 1) Copy the SysPath.py file from the {VisumInstallation}\\PythonModules\\VisumPy directory to the folder of the script file requiring debugging.
- 2) Insert the following code into the first line of your Python script:

```
import SysPath
```



```
successful = SysPath.ChangePythonSysPath("125", "32")
```

The first parameter of the `ChangePythonSysPath` function specifies the desired Visum version. With the second parameter (32 or 64), you specify whether the Visum being used is a 32 or 64 bit installed version of Visum. The specifications are customized in the global Python variable `sys.path` with this call, enabling access to the correct Python libraries of Visum. The function returns 'true' if correctly customized. In the event of an error, it returns 'false'.

The following code is an example of the call described above in the `SetMatrixDiagonal` add-in:

```
# -*- coding: iso-8859-15 -*-
import SysPath
successful = SysPath.ChangePythonSysPath("125", "32")

import wx
import os
import sys
from VisumPy.wxHelpers import wxNetobjCombo, wxAttrIDButton
from VisumPy.AddIn import AddIn, AddInState, AddInParameter

def matrixFormatter(no, code):
    return "%d | %s" % (no, code)

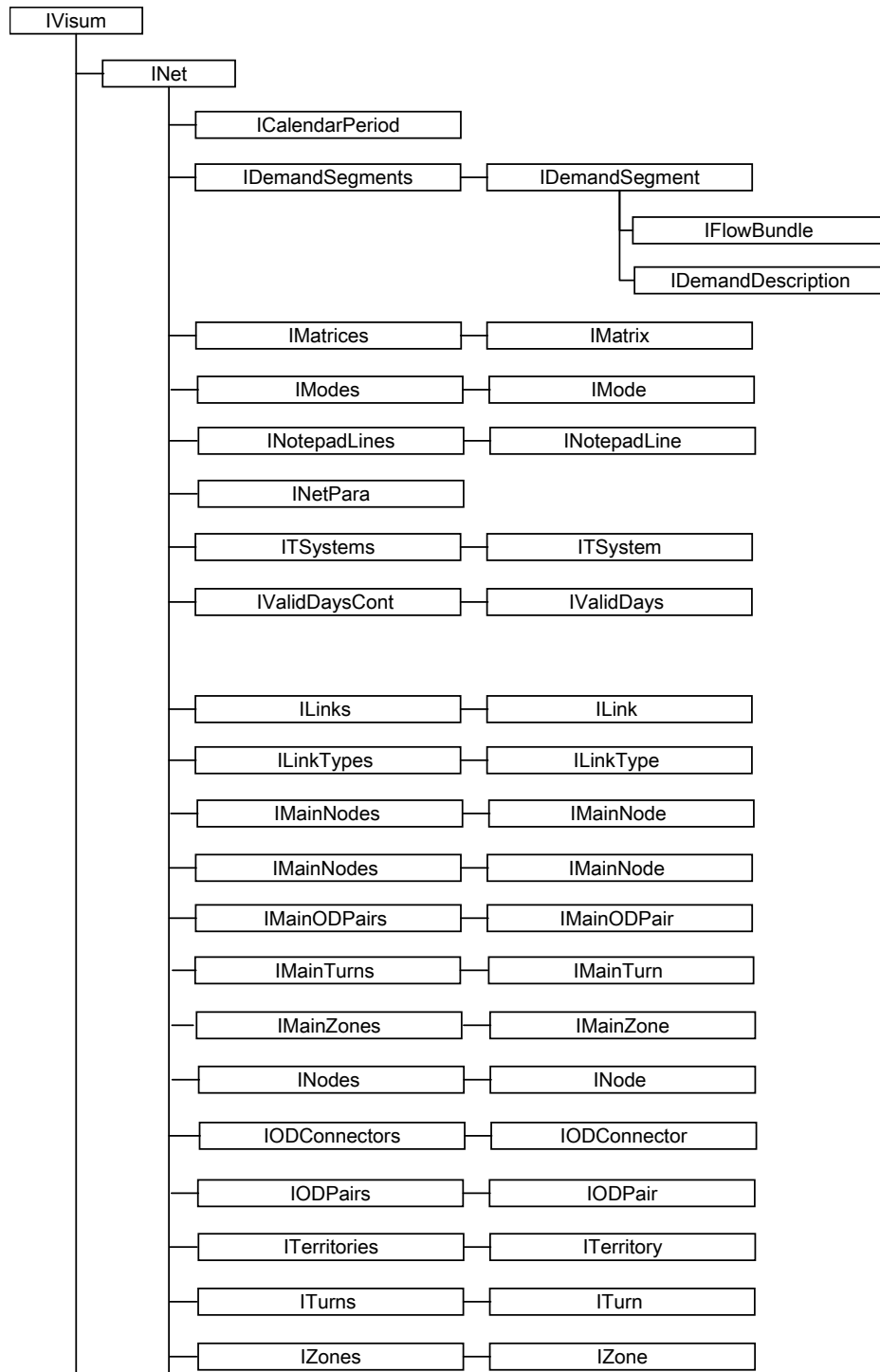
class MyDialog(wx.Dialog):
    def __init__(self, *args, **kwargs):
        # begin wxGlade: MyDialog.__init__
...

```

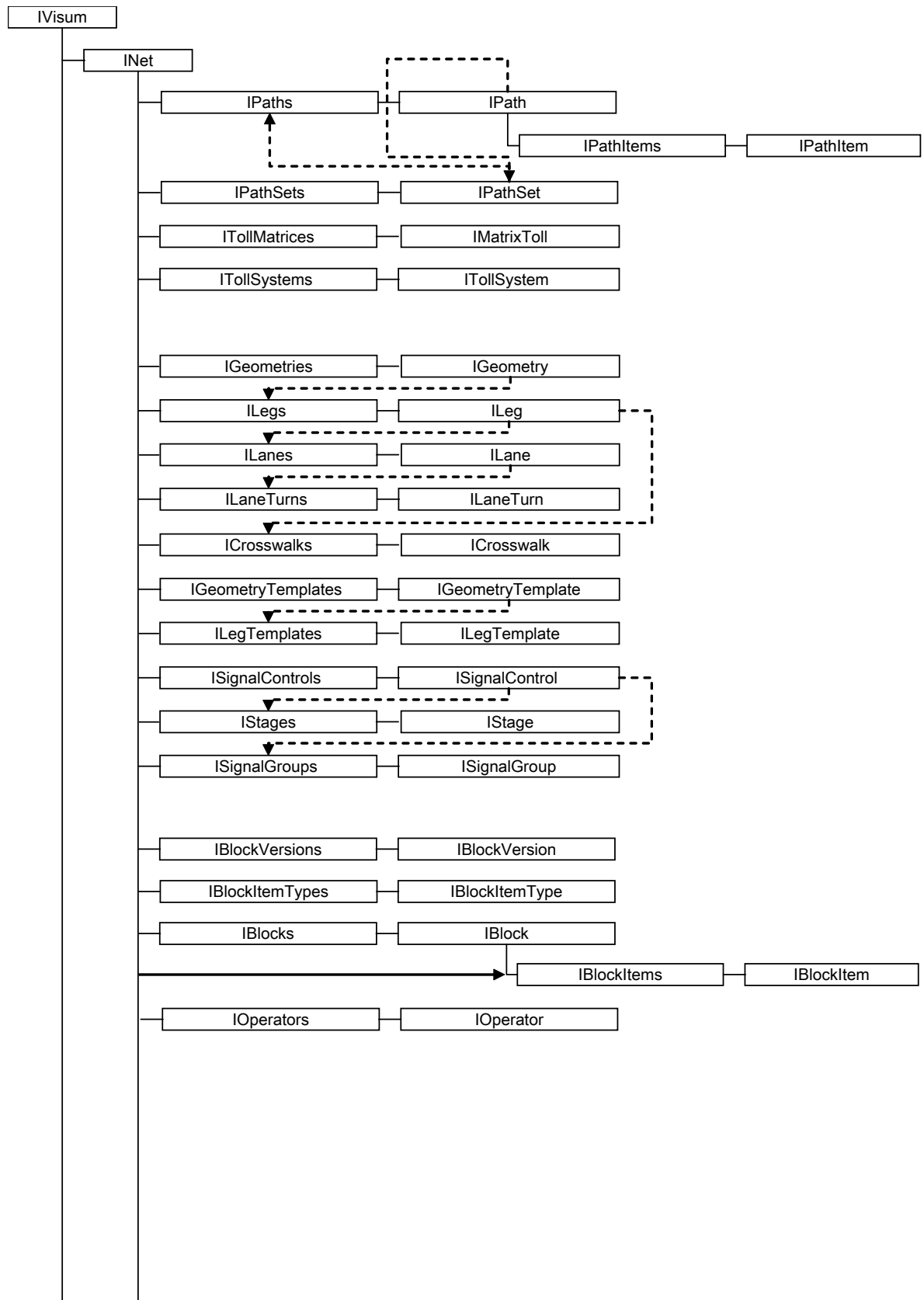
3 The Visum object model

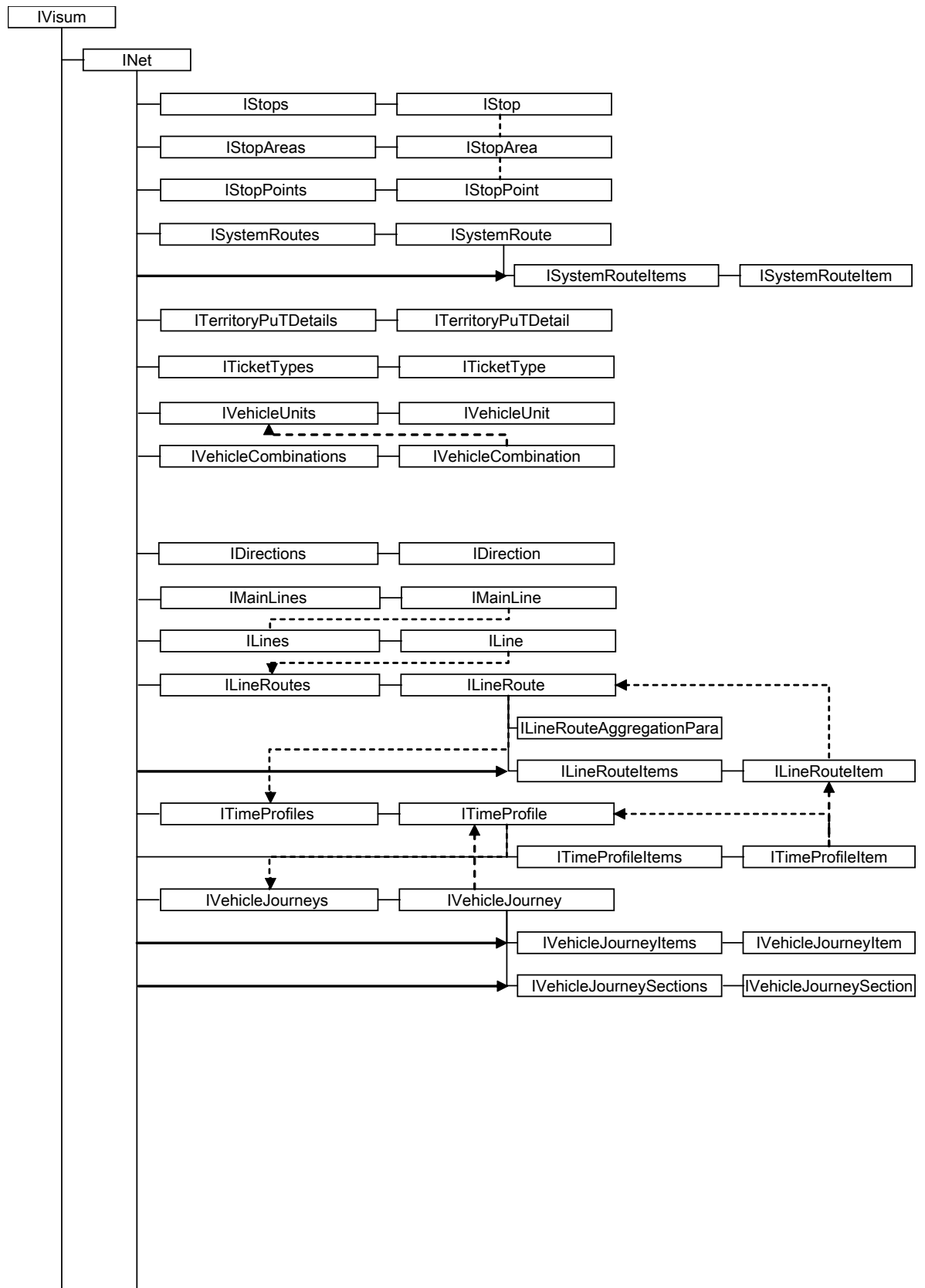
The Visum COM Model is subject to a strict object hierarchy (see Figure below). IVisum is the highest-ranking object. To access a sub-object, e.g. a link in the network, one must follow the hierarchy. Please see the COM-interface reference included with the Online Help of Visum for details about the various objects and their methods and properties.

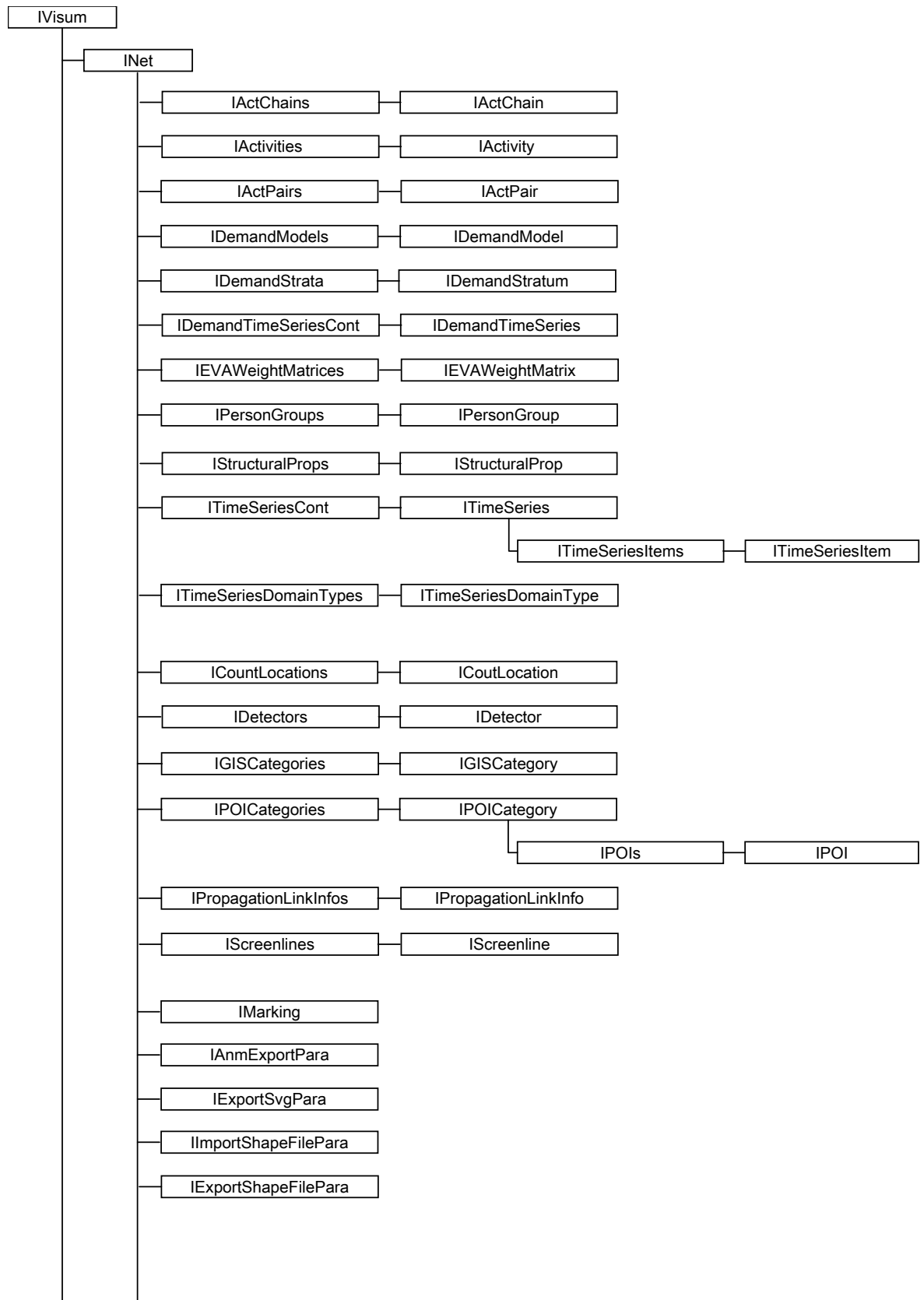
The Visum object model:

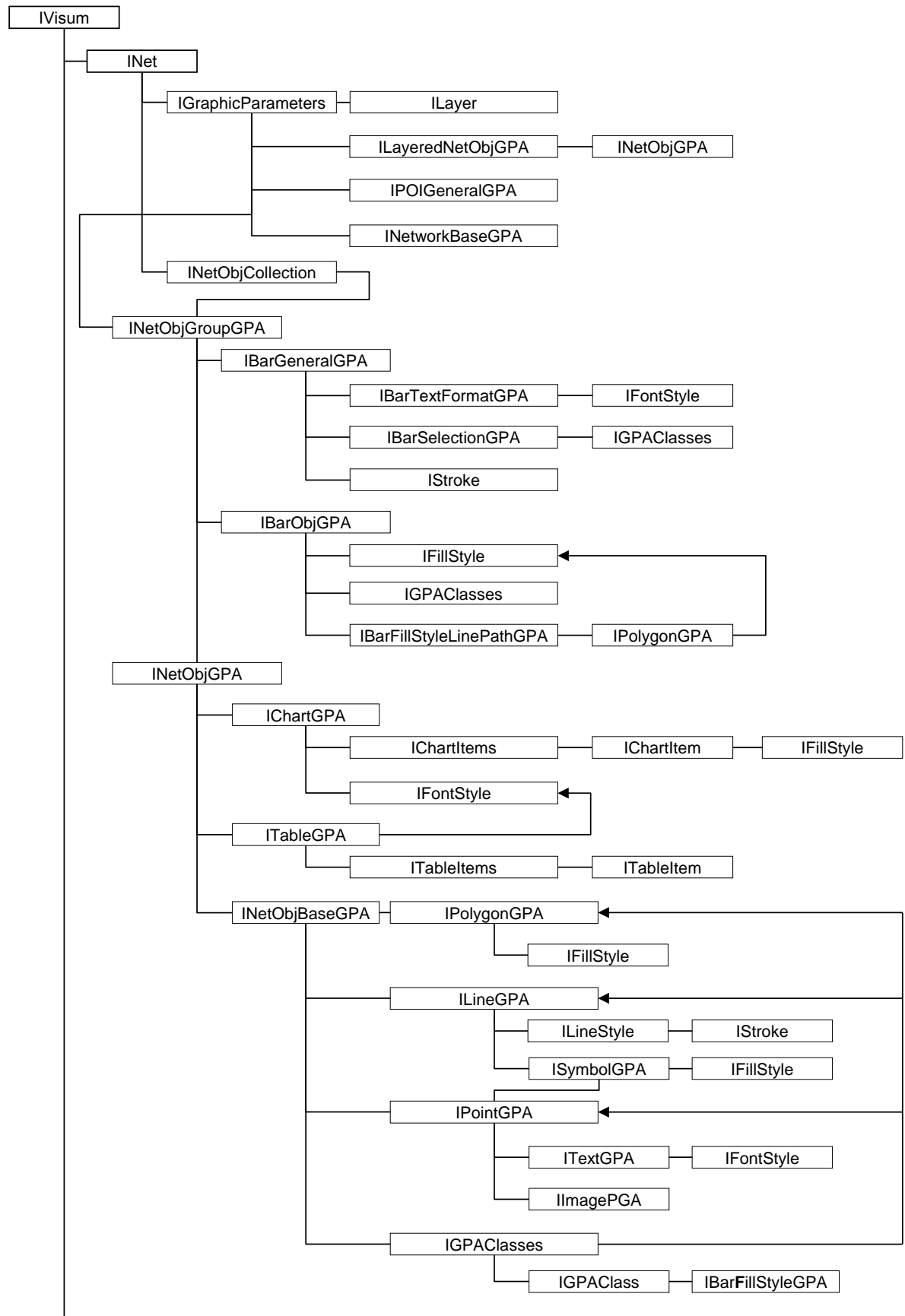


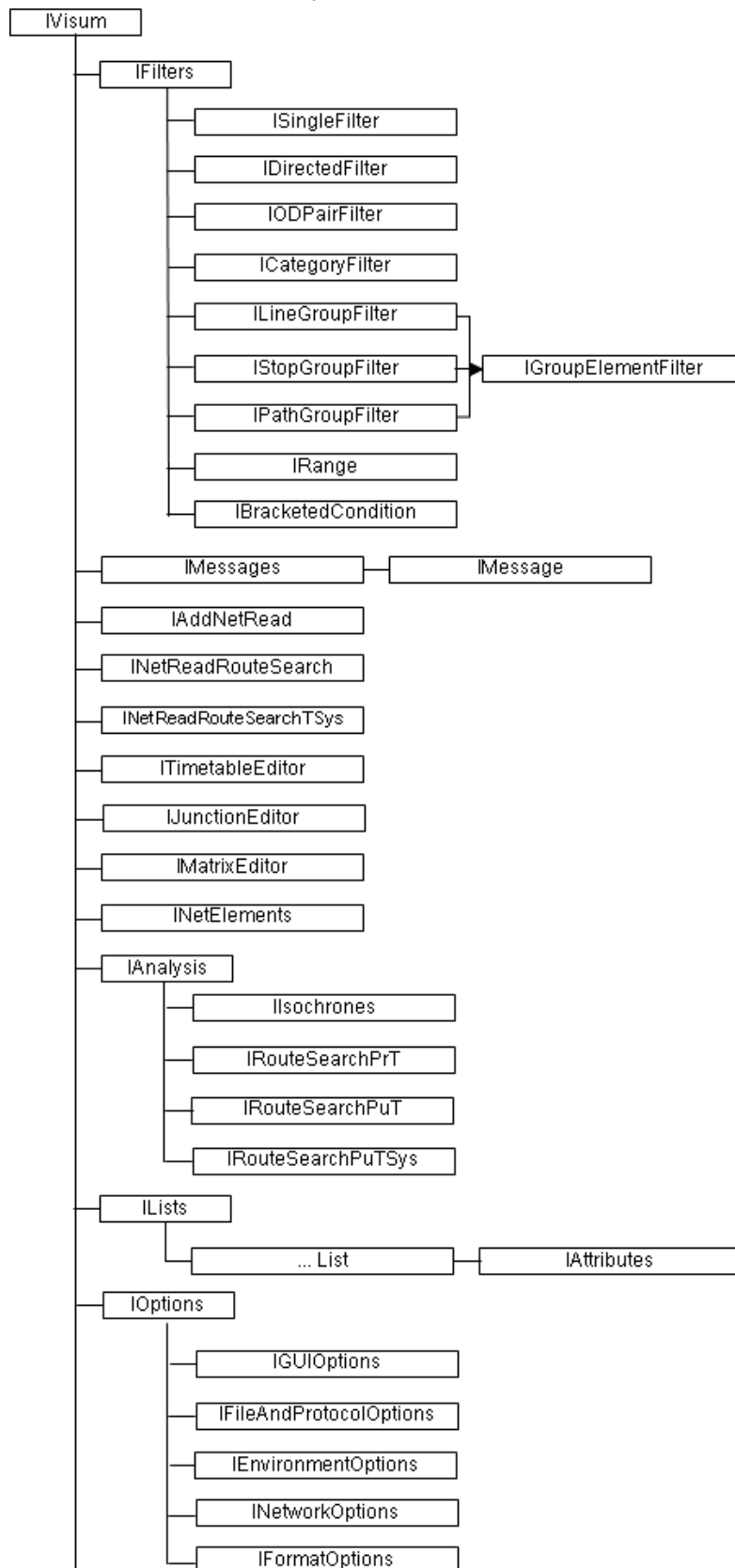
Continuation Visum object model:

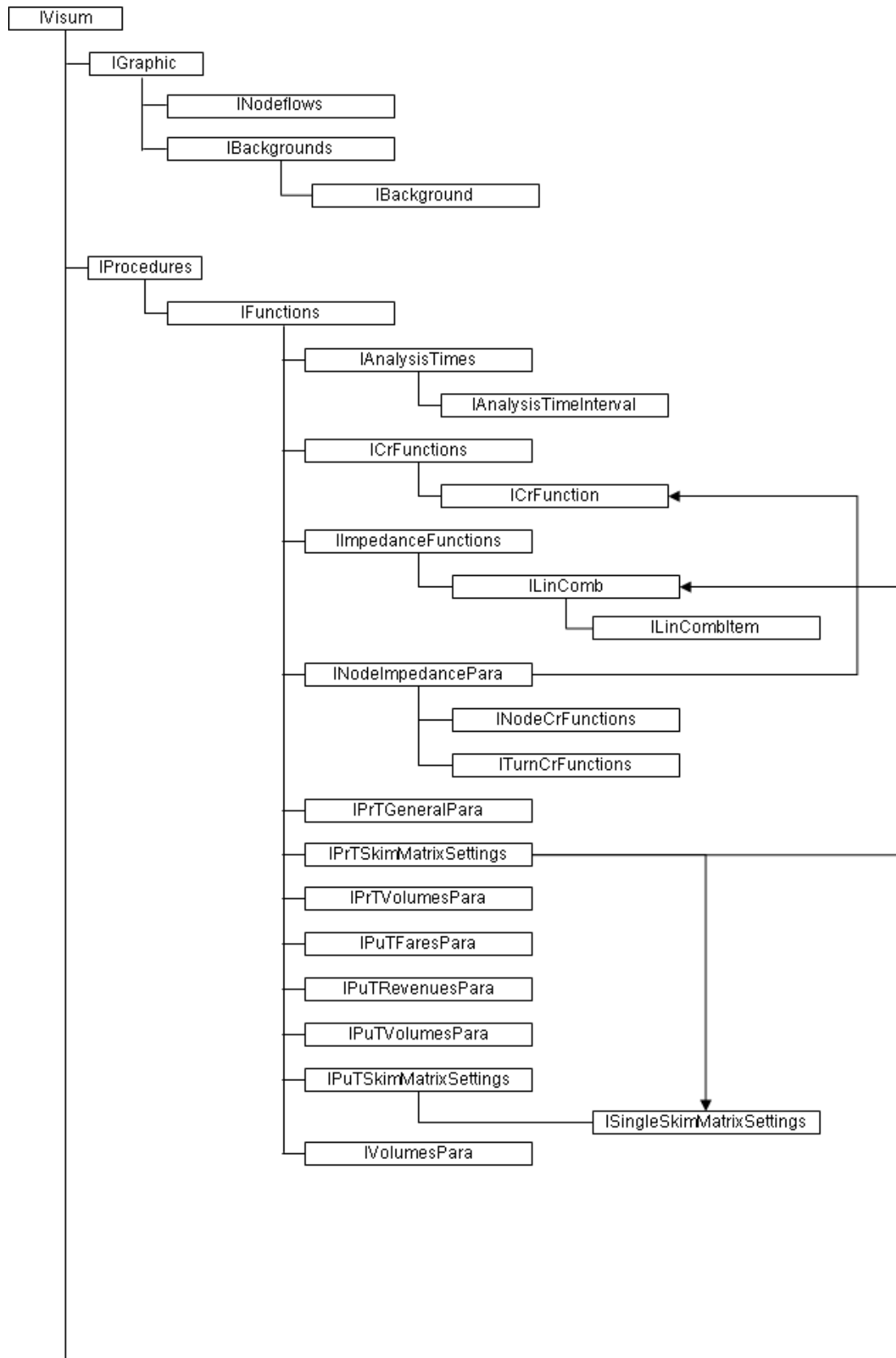


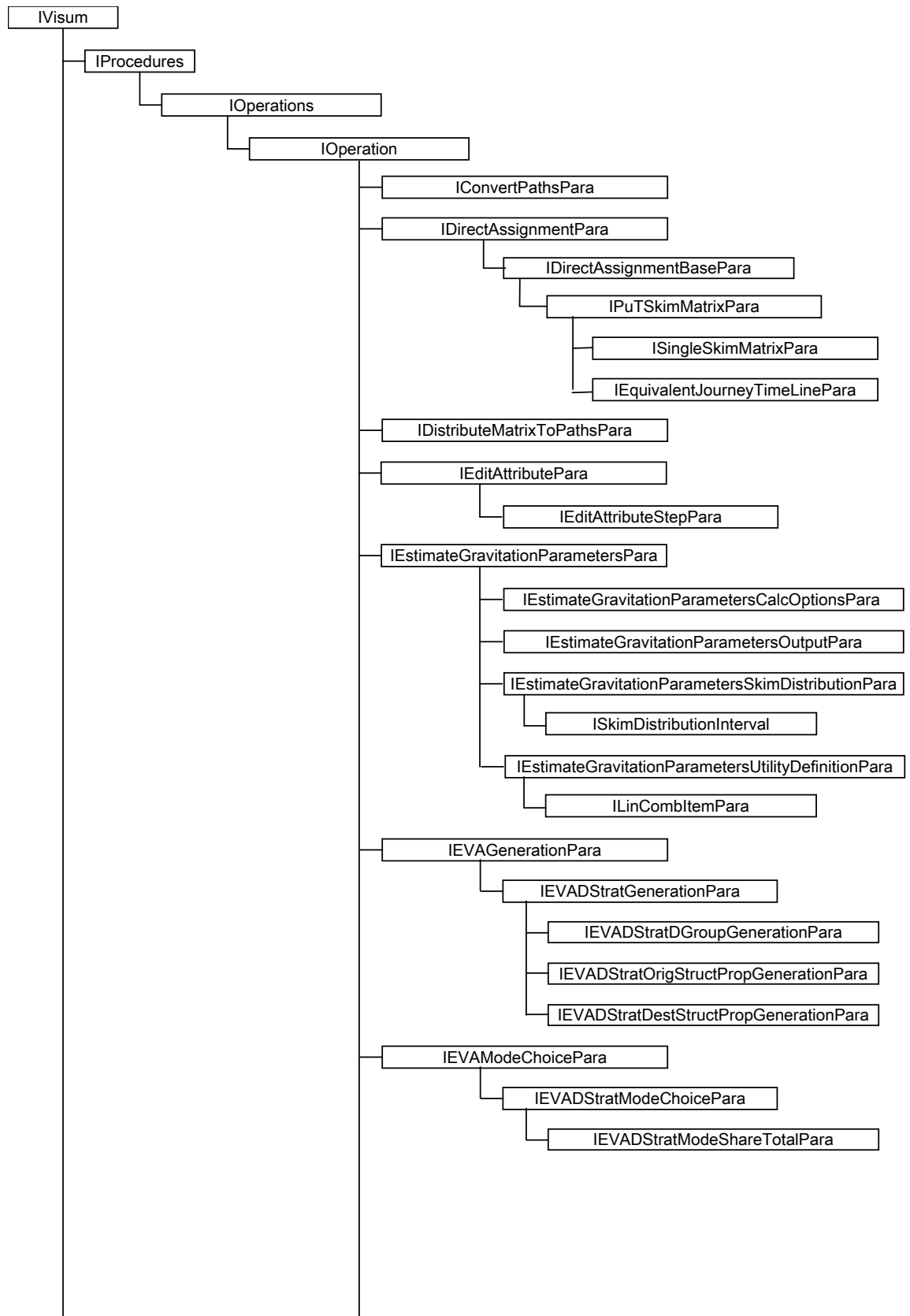
Continuation Visum object model:

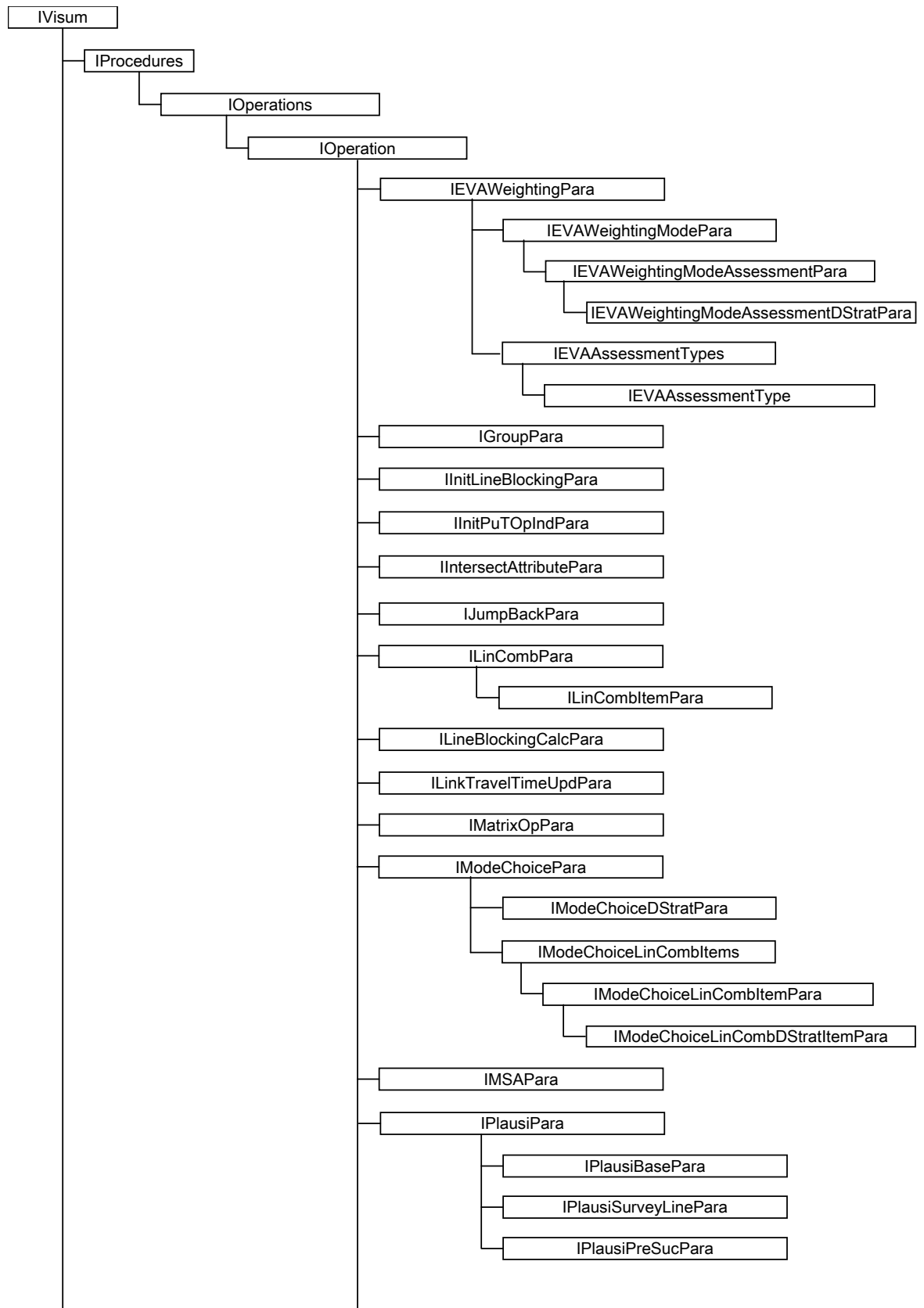
Continuation Visum object model:

Continuation Visum object model:

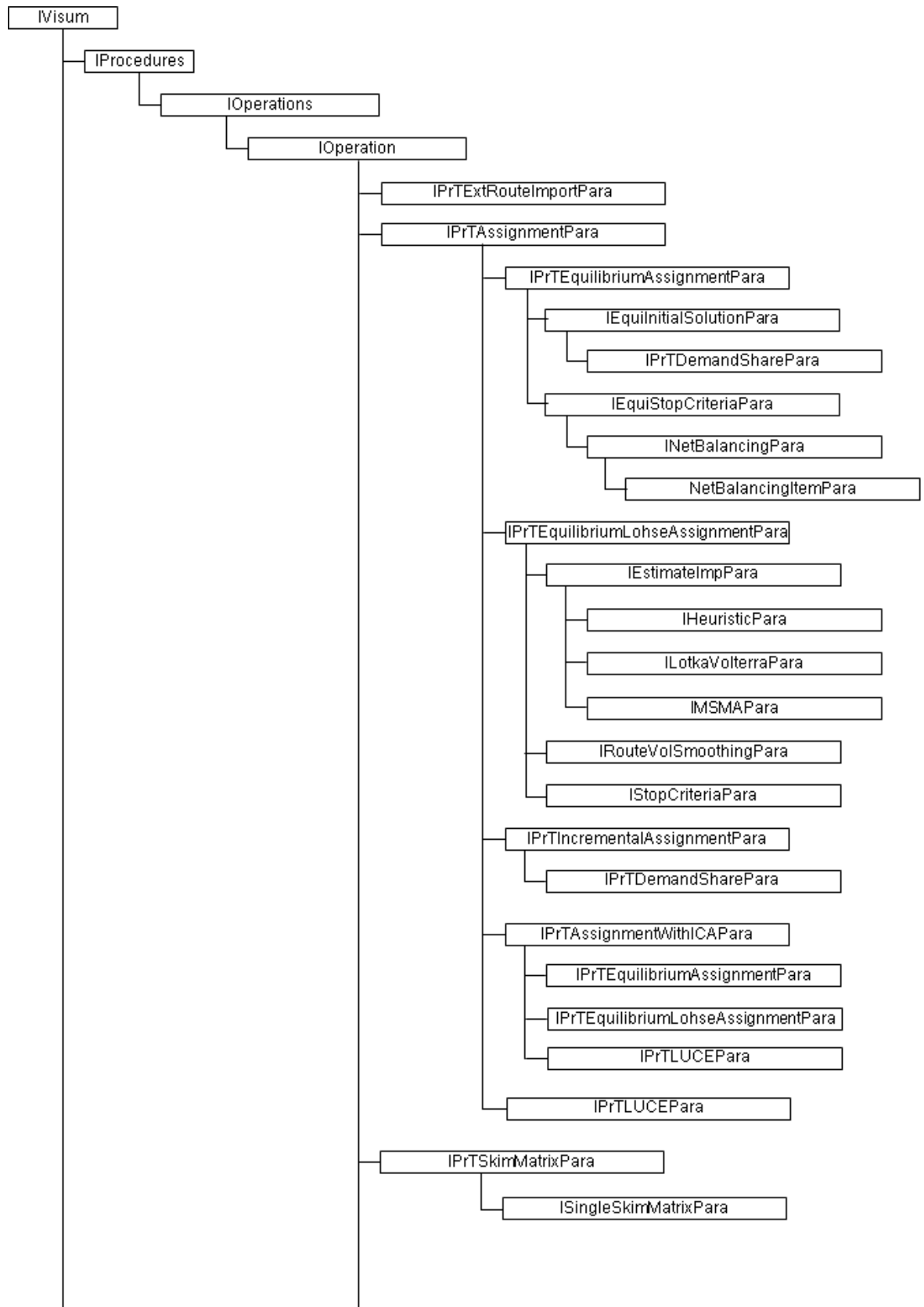
Continuation Visum object model:

Continuation Visum object model:

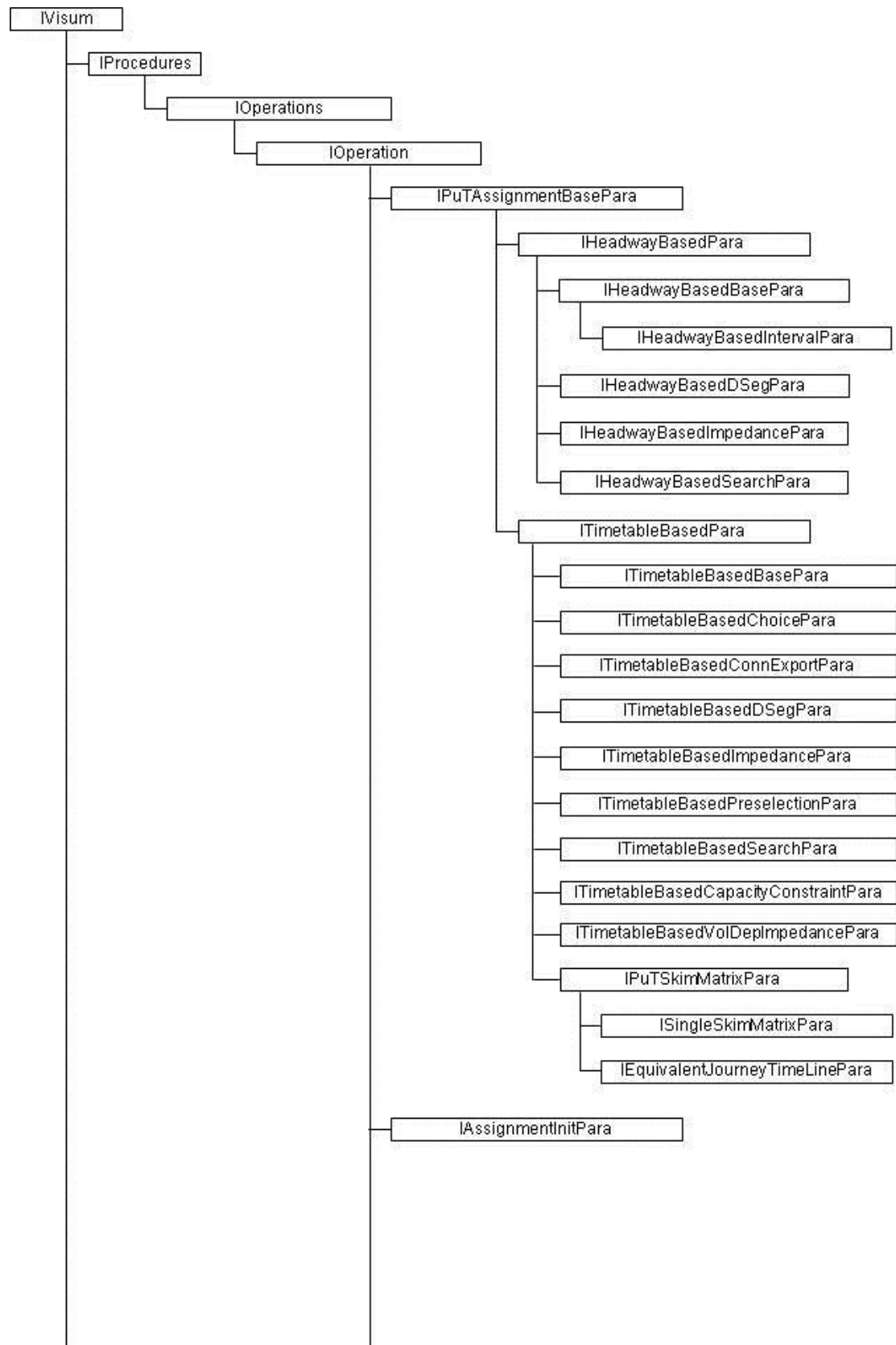
Continuation Visum object model:

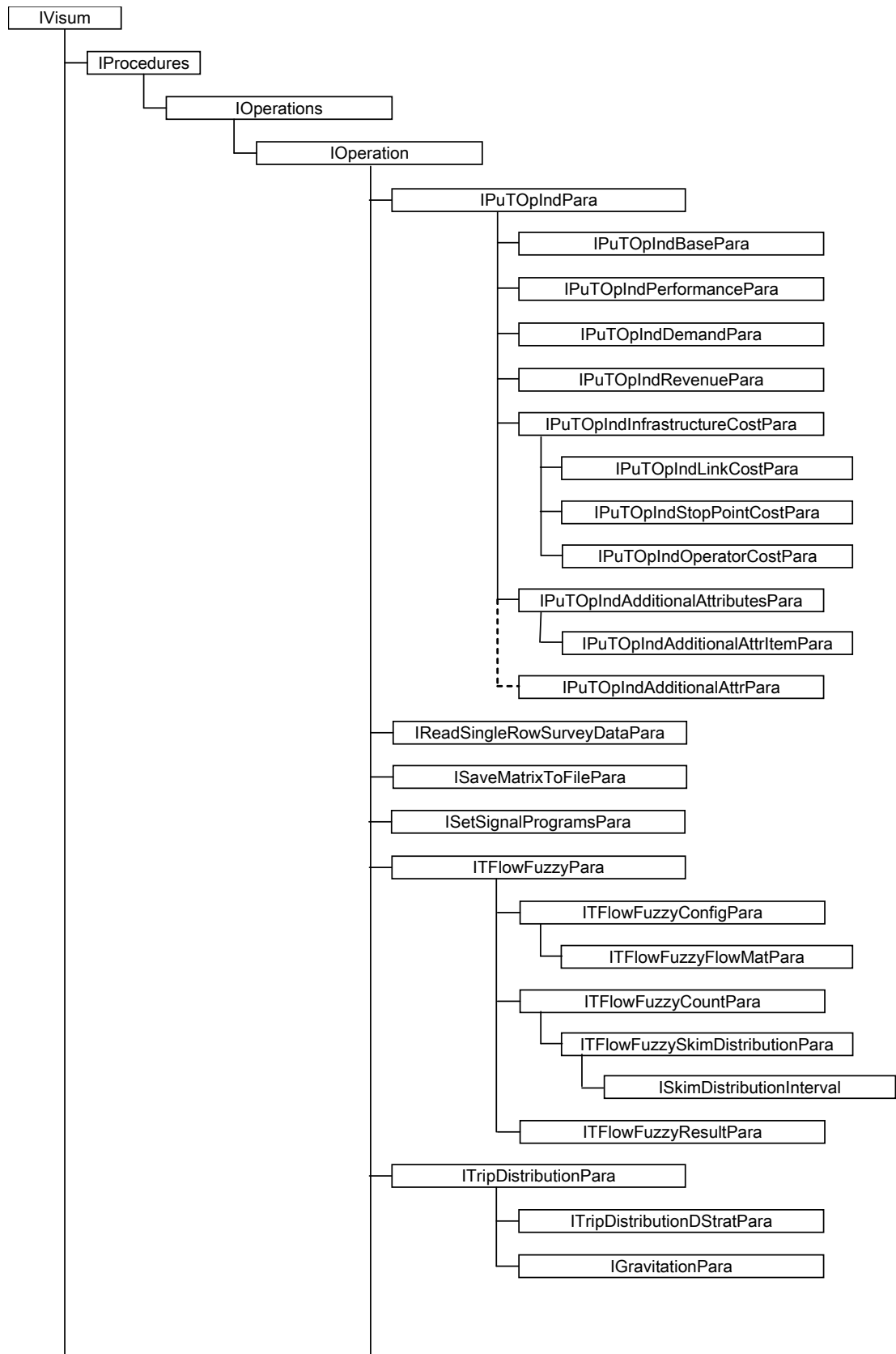
Continuation Visum object model:

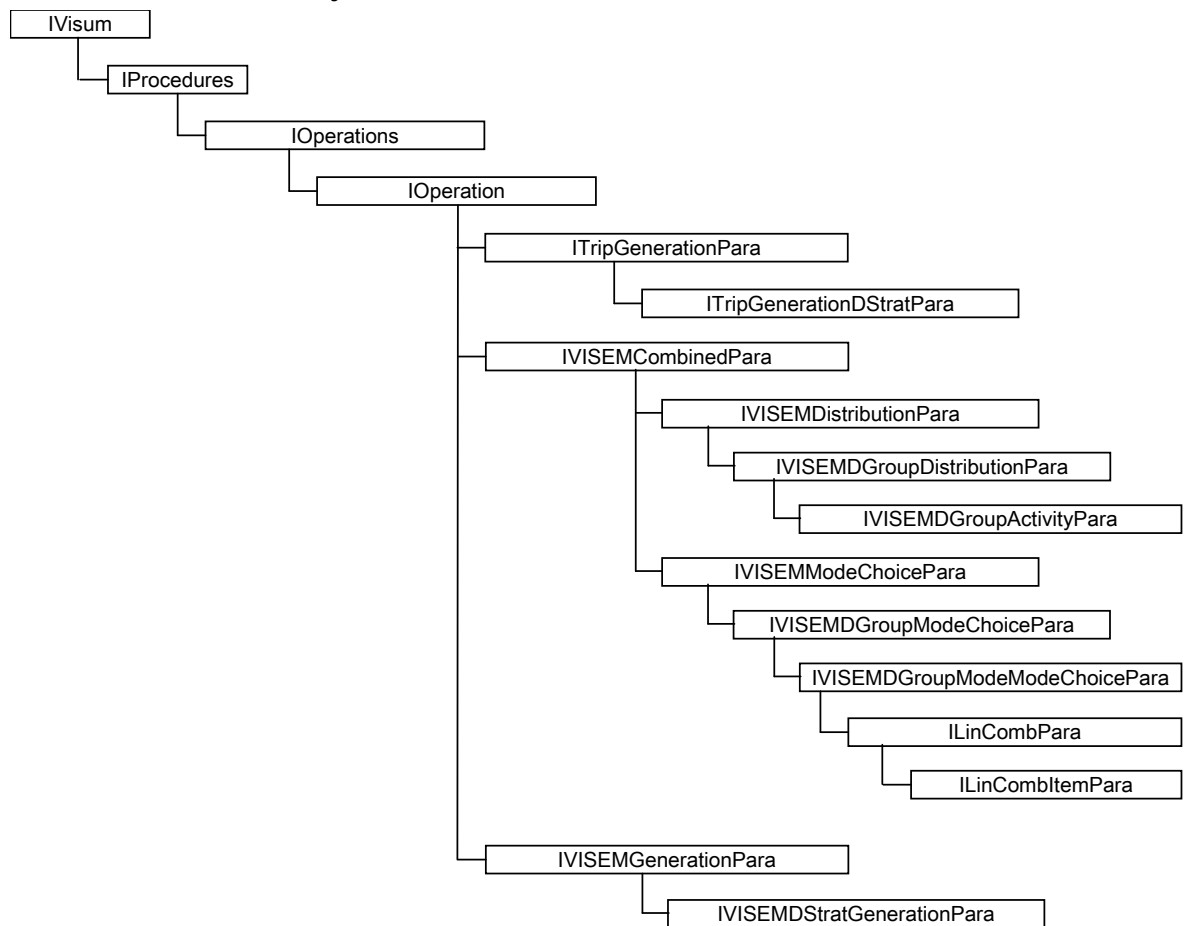
Continuation Visum object model:



Continuation Visum object model:



Continuation Visum object model:

Continuation Visum object model:

4 First Steps

4.1 Access to Visum objects and files

Visum Instances

When working with Visum manually it is possible to start several instances of Visum. You can also call several different versions (e.g. Visum 12.0 and Visum 12.5). The same applies when working with COM: Per default, the selected Visum version is started every time you call COM. Entering the main version number during the call, optionally followed by the one- or two-digit version number, allows you to distinguish between versions of Visum. If you enter the main version number (e.g. `CreateObject("Visum.Visum.11")` or `CreateObject("Visum.Visum.10")`), the version last installed will be started, in our case the latest release: 12.XY or 13.XY. Analogously, `CreateObject("Visum.Visum.115")` will start the most recent Visum 11.5X installed on this computer. If you have installed a 32-bit and 64-bit version, you can also enter "Visum.Visum-32.11XY" or "Visum.Visum-64.11XY" to call a version. A Visum instance started via COM is automatically closed as soon all COM objects are set to nothing.

If several COM programs are expected to work with the same Visum instance explicitly, two parameters can be used to control Visum:

- */Embedded*: This parameter starts a Visum instance that works as described above. Additionally, this instance is closed automatically, when the Visum-COM object has been released.
- */Automation*: A Visum instance started with this parameter serves as a server for all COM applications – until it is closed manually.

Please note: As long as a client is still running, no further COM applications can be started! Thus an automation server can be used only if several COM applications are not to be executed in parallel.

Creating/Deleting a Visum object

This example creates a Visum object. On screen, the Visum session window appears. Selected language and current Visum version No. are displayed in a message box. Afterwards, the Visum object will be deleted.

Example file: CreateVisum.xls

```
'This example shows how to create a Visum-Object. The current language
'and version number of Visum are shown in a message-box before the
'Visum-Object is deleted.
Sub FirstVisumExample()

    'variable declaration
    Dim Visum As Object                                ' Variable Visum
    'create the Visum-Object (connect the variable Visum to the software Visum)
    Set Visum = CreateObject("Visum.Visum.125")

    'get the current language and show it in a message-box
    MsgBox "Current language is: " & Visum.GetCurrentLanguage

    'get the current version number and show it in a message-box
    MsgBox "Current version number is: " & Visum.VersionNumber
```



```

        'delete the Visum object to close the Visum-software
        Set Visum = Nothing
    End Sub

```

Loading a versions or network file

To load Visum data from a version or a network file or from an Access database, the following methods are provided:

- Visum.LoadVersion (file name)
- Visum.LoadNet (file name)

Note! The complete path has to be specified for database, network or version files.

VB example: 6.1.1

Saving version, network and data base files

To save Visum data to file, the following methods are provided: Visum.SaveVersion (name of the version file)

- Visum.SaveVersion (file name)
- Visum.SaveNet (file name)
- Visum.SaveAccessDatabase (file name)

Note! The complete path has to be specified for database, network or version files.

VB example: 6.1.2

4.2 Access to objects and attributes

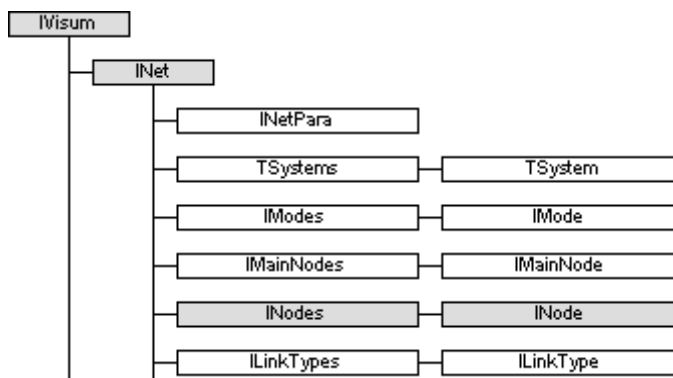
4.2.1 Introduction

For access to single Visum objects, the hierarchy of objects (see fig. Object Model) has to be followed. IVisum is the highest-ranking object of the model. IVisum allows access to the sub-objects INet (Network) and ILists (Lists) and further lower-ranking objects.

From the object INet, access is possible to the sub-objects ITSystems (transport systems), INodes (Nodes), ILinkTypes (link types) 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 Visum COM model are named using the plural form. The object ITSystems includes all transport systems, the object INodes includes all nodes, the object ILinks includes all links of the network.

Via the Collection Object, each individual object of the Collection can be accessed. Basically, random access to one specified object, access to a list of all objects, and loops over all objects of the container can be distinguished.

Below, these different alternatives are illustrated by examples using the Inode object. Here, the objects to be regarded for access to a single node are highlighted in grey.



4.2.2 Random access to a specified object using its key

Access via ItemByKey

The ItemByKey method guarantees unambiguous access to a single Visum object. To call a particular Visum object, the object's external key (identifier, consisting of one or several ID components) is used. Then access to the properties (attributes) of the particular Visum object is possible. Access to a network object depends on the network object type.

- Transport systems: ID = code

```
Set Obj = Visum.Net.TSystems.ItemByKey("P")
Name = Obj.AttValue("Name")
```

- Nodes: ID = number

```
Set Obj = Visum.Net.Nodes.ItemByKey(10)
Type = Obj.AttValue("TypeNo")
```

- Link: ID = FromNode, ToNode

```
Set Obj = Visum.Net.Links.ItemByKey(10, 11)
Length = Link.AttValue("Length")
```

VB example: 6.2.1

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 *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 visum.net.nodes
    node.AttValue("...") = ...
next
```

Python:

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

Loop using Item

This method also allows programming of loops:

VBA:

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

This method transfers all objects of the container into a list using GetAll. When accessing a single object (here INode) via Item, the object is selected by its Index number in the Collection Object (here INodes). Since the index of an object may change due to changes to the Visum network, an object cannot necessarily be determined unambiguously by its index number. That is why this type of access should be used in program loops only.

Usually the first 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 (see example 6.2.10).

VB example: 6.2.2**Loop using For-Each**

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

VB example: 6.2.4**Access using iterator object**

The collection object provides an iterator object that can be incremented. It can be used to access the current object. In contrast to the property GetAll that saves all objects of a collection within an array and that must be used before using Item or For...Each, the use of the Visum 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 = Visum.Net.Nodes.Iterator
While nodeIter.Valid
    Set curNode = nodeIter.Item
    curNode.AttValue("...") = ...
    nodeIter.Next
Wend
```

VB example: 6.2.5

4.2.5 Read and Save attributes

"Access for Reading data" is provided for all attributes, whereas "Access for Saving data" is limited to a number of attributes. Which attributes are available for which objects and what kind of access is possible is documented in the appended file `ATTRIBUTE.XLS`.

The attributes are always accessed by the `AttValue` method. The reading of data has already been used in the last example:

```
Cells(row, 2) = Node.AttValue("NO")
```

By accessing the attributes, only the language independent CodeIDs are allowed and they are used case insensitive.

Sub-attributes are added in parentheses to the attribute, e.g. `V0_PrTSys(C)` for the attribute speed at free flow of the private transport system car. Where there are more than one sub-attributes they are separated by a comma, e.g. `VolVeh_TSys(C,AP)` for the vehicle volume in the analysis period for cars. The feasible sub-attributes for each attribute are also listed in the file `ATTRIBUTE.XLS` (rows *SubAttr1* and *SubAttr2*).

If the value of a calculated numerical attribute is not defined, since e.g. the assignment has not been executed, the function `AttValue` returns values that are useless. That is why container classes like `ILinks` can be asked for the state of calculation using the `AttState` function. So defined values can be distinguished from undefined ones by the calling program.

VB example: 6.2.6

4.2.6 Execute Assignments

The COM object `IProcedures` permits the execution of procedures in several manners. In the most simple case, the operations defined in the Visum procedure dialog (menu `CALCULATE - PROCEDURES`) can be executed. It is also possible to read settings from an XML or procedure parameters file or to change parameters using the appropriate COM functions.

The operation specified as Procedure can be executed, effects to the Visum network can be analyzed. The operation and specified parameters cannot be saved to a Visum version file. Default values will be used for all parameters that have not been specified.

VB Examples: Section 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 Visum executable file installed (currently this is VISUM11.EXE) in order to use Visum COM objects. Then the objects and methods of the Visum 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 Visum objects set the clause „using VisumLIB“ to access the Visum COM object model. A Visum entity is created by calling the constructor using the expression „VisumClass Visum = new VisumClass()“.

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# optional parameters must be specified explicitly, which means there are no optional parameters. For example for loading a version file you have to specify the (in VBA optional) input parameter „convertFromImperial“.

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

C# example:

```

using System;
using VisumLIB;

namespace ConsoleApplication1
{
    /// <summary>
    /// Class NameReader reads the names of all StopPoints in Visum network.
    /// </summary>
    class NameReader
    {
        /// <summary>
        /// The main entry point for the application.
        /// </summary>
        [STAThread]
        static void Main(string[] args)
        {
            try
            {
                VisumClass Visum = new VisumClass();

                Visum.LoadVersion(@"C:\Program
Files\PTV_Vision\Visum120\Example\EXAMPLE_GER.ver", false);
                StopPoints stopPointList = (StopPoints) Visum.Net.StopPoints;
                IEnumerator iter = (IEnumerator) stopPointList.Iterator;

                while (iter.Valid)
                {
                    StopPoint stopPoint = (StopPoint) iter.Item;
                    string name = stopPoint.get_AttValue("Name").ToString();
                    Console.WriteLine(name);
                    iter.Next();
                }
                Visum = null;
            }
            catch (Exception ex)
            {
                Console.WriteLine("Error while reading stop point names via COM: " +
ex.ToString());
            }
        }
    }
}

```

This example loads a version file, iterates over all stop points and writes their names to the console.

5.2 COM in JAVA

Basically, there are two ways of using Visum via COM and each way requires different developer skills. One option is to use the Microsoft development environment (J#) that is simple and completely analogous to C#, but cannot be run on any JAVA virtual machine (JVM). The other option is to use a "pure" JAVA application.

5.2.1 Case 1: Microsoft Development Environment (J#)

You can use Visum via COM in a Microsoft JAVA framework in the same way as you do in a C# environment. To do so, in the development environment used, set a reference to the Visum executable file installed (currently this is VISUM100.EXE) in order to use Visum COM

objects. Then the objects and methods of the Visum COM object model can be used in the project like in the C# case. In the development framework Microsoft Visual C# .NET, you can browse the object model using the "Object Browser" similar to the object catalogue in VBA.

Classes using element of the Visum COM object model must contain the clause „import VisumLIB.*;“ in their definition. A Visum entity is created by calling the constructor using the expression „VisumClass Visum = new VisumClass()“.

As in C# the names of objects and methods differ from the notation used in VBA. Properties can not be used like attributes of the object – they can not be assigned any value. Properties occur explicitly as methods.. They are written with prefix „get_“ or „set_“ and must be completed using parentheses like any other procedure call. Access methods return objects of type „object“ that have to be casted explicitly to their „real“ type. Otherwise, their methods can not be used.

There are no optional parameters in the JAVA signatures of methods. Therefore all parameters have to be set explicitly, even if default parameters are defined within the VBA style method definition.

JAVA-(J#-)example:

```
package CallVisum;

import VisumLIB.*;

/**
 * Class CallVisum shows the principle of using Visum in a java environment via
 * COM. There are no members except the static function main which makes use of
 * the COM Interface. A COM reference to the local Visum installation must be set
 * in the project.
 */
public class CallVisum
{
    /** static function main */
    public static void main(String[] args)
    {
        try
        {
            VisumClass Visum = new VisumClass();
            Visum.LoadVersion("C:\\Program Files\\PTV_Vision\\Visum120\\
                               Example\\EXAMPLE_GER.ver", false);
            INet MyNet = (INet) Visum.get_Net();
            IStopPoints stopPointList = (IStopPoints) MyNet.get_StopPoints();
            IIterator iter = stopPointList.get_Iterator();
            while (iter.get_Valid())
            {
                IStopPoint stopPoint = (IStopPoint) iter.get_Item();
                String name = stopPoint.get_AttValue("Name").ToString();
                System.out.println(name);
                iter.Next();
            }
            Visum = null;
        }
        catch (java.lang.Exception e)
        {
            System.out.println(e);
        }
        System.exit(0);
    }
}
```

The program coincides functionally and merely literally with the C# example.

5.2.2 Case 2: "Pure" JAVA Application

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 in JAVA applications using so-called COM bridges. To do so, wrapper classes corresponding to the objects provided by the Visum 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 Visum COM objects in the JAVA application is 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 Visum 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 „VisumLib“ is assigned to it. After initializing the COM interface which is triggered in the Microsoft Visual C++ development environment by calling „`CoInitialize(NULL)`“ a Visum object can be created.

Now the objects of the Visum 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 Visum objects explicitly. This provides access to the Visum 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 "stdafx.h"

#include <iostream>
#include <string.h>

// import the *.exe file providing the COM interface
// use your Visum installation path here as the example shows
#import "C:\\Program Files\\PTV_Vision\\Visum120\\Exe\\Visum120.exe"
    rename_namespace ("VisumLib")

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()
{
    // initialize COM
    CoInitialize(NULL);

    // create Visum object
    VisumLib::IVisumPtr pVisum;
    HRESULT hr = pVisum.CreateInstance("Visum.Visum.11");
    if (hr != S_OK)
    {
        cout<<"COM connection to Visum cannot be established.\n";
    }

    try {
        pVisum->LoadVersion(bstr_t("C:\\Program Files\\PTV_Vision\\
            Visum100\\Example\\EXAMPLE_GER.ver"),false);
        VisumLib::INetPtr pNet;
        VisumLib::IStopPointsPtr pStopPointList;
        VisumLib::IIteratorPtr pIter;
        VisumLib::IStopPointPtr pStopPoint;
        bstr_t name;
        // attach pointer to Visum objects
        QueryAttach(pNet, pVisum -> GetNet());
        QueryAttach(pStopPointList, pNet -> GetStopPoints());
        QueryAttach(pIter, pStopPointList -> GetIterator());
        while (pIter -> GetValid()) {
            QueryAttach(pStopPoint, pIter -> GetItem());
            name = pStopPoint -> GetAttValue("Name");
            cout << name << "\n";
            pIter -> Next();
        }

        // release pointers
        if (pStopPoint.GetInterfacePtr() != NULL)
            pStopPoint.Detach() -> Release();

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

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

        if (pNet.GetInterfacePtr() != NULL)
            pNet.Detach() -> Release();
    }
    catch (...) {
        cout<<"Error during COM connection.\n";
    }
    // release Visum object
    if (pVisum.GetInterfacePtr() != NULL)
        pVisum.Detach() -> Release();
    return 0;
}

```

This example loads a version file, iterates over all stop points and writes their names to the console. This is exactly what the C# example does.

6 Visum 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 Visum object library. This is done by setting a reference to the Visum object library in the extras+references menu. The object library is part of the Visum program and is automatically available after the installation of Visum.

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 Visum 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 Visum 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 Visum Objects and Reading the Version File

This example creates a Visum object and loads a version file. The name of the version file is read from the Excel worksheet. The net is shown in the Visum window. Additionally the presentation of the Visum window is changed and the window is displayed in the minimized form.

Example file: LoadVersion.xls

```
'This example shows how to create a Visum-Object and load a
'version file from worksheet cell.
Sub LoadVersionExample()

    'variable declaration of variable Visum
    Dim Visum As Object

    'create the Visum object (connect the variable Visum to the software Visum)
    Set Visum = CreateObject("Visum.Visum.120")

    'Load version, file name from cell B3
    Visum.LoadVersion Cells(5, 2)

    'minimize Visum-Window
    Visum.Graphic.ShowMinimized

    'Stop to show you that Visum has loaded the version
    MsgBox "Stop"
```

```

        'delete the Visum-Object to close the Visum-software
        Set Visum = Nothing
    End Sub

```

6.1.2 Reading and Saving the Version File

This example creates a Visum object and loads a version file. The name of the version file is read from the Excel worksheet. The length of all links is to be modified. Then the result is saved to a new version file.

Example file: SaveVersion.xls

```

'This example create a Visum-Object and load a version file from worksheet cell.
'The length of all links , V0 for private transport and the time for the
'transport system bus are modified and the result will be saved in a new version
file.

Sub SaveVersionExample()

    'declare object Visum
    Dim Visum As Object

    'declare other variables
    Dim AllLinks() As Variant
    Dim Link As Variant
    Dim Length As Variant
    Dim VNull As Variant
    Dim TBus As Variant

    'create the Visum-Object (connect the variable Visum to the software Visum)
    Set Visum = CreateObject("Visum.Visum.120")
    'load version, filename from cell B5
    Visum.LoadVersion Cells(5, 2)

    'go over all links
    AllLinks = Visum.Net.Links.GetAll
    For Each Link In AllLinks
        'get length in meter
        'then write new length
        Length = Link.AttValue("LENGTH")
        Length = Length * 1.1
        Link.AttValue("Length") = Length
    Next Link

    'save version, filename from cell B6
    Visum.SaveVersion Cells(6, 2)

    'delete all objects to close Visum-software
    Set Visum = Nothing
End Sub

```

6.1.3 Reading Network File Additively

This example demonstrates how to control additive net reading using conflict avoiding strategies. The control is done using objects of type AddNetRead for single net object types and NetReadRouteSearchTSys or NetReadRouteSearch for shortest path searches when inserting line routes or system routes. The control parameters correspond exactly to the interactive case.

Example file: AddNetRead.xls

```

Sub AddNetReadExample()

    'declare object Visum and other objects
    Dim Visum As Object
    Dim AddNetReadController As Object
    Dim NetReadRouteSearchTSysController As Object
    Dim NetReadRouteSearchController As Object

    'create the Visum-Object and load version, filename from cell B3
    Set Visum = CreateObject("Visum.Visum.120")
    Visum.LoadVersion Cells(5, 2)

    'create AddNetRead-Object and specify desired conflict avoiding method
    Set AddNetReadController = Visum.CreateAddNetReadController
    AddNetReadController.SetConflictAvoidingForAll 10000, "tra_"

    'create NetRouteSearchTSys-Object and choose route search options
    'create one object per TSys if desired
    Set NetReadRouteSearchTSysController = Visum.CreateNetReadRouteSearchTSys
    NetReadRouteSearchTSysController.DontRead

    'create NetRouteSearch-Object and assign NetRouteSearchTSys-objects
    Set NetReadRouteSearchController = Visum.CreateNetReadRouteSearch
    NetReadRouteSearchController.SetForAllTSys NetReadRouteSearchTSysController

    'additively read the net file, filename from cell B4
    Visum.LoadNet Cells(6, 2), True, NetReadRouteSearchController,
    AddNetReadController

    'write version file, filename from cell B5
    Visum.SaveVersion Cells(7, 2)

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

End Sub

```

6.2 Access to Objects and Attributes

6.2.1 Finding nodes 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 file: GetItemByKey_Node.xls

```

'This example shows how to access single nodes of a net by using ItemByKey
Sub NodeAttributes()

    'declaration of variables
    Dim Visum As Object
    Dim aNode As Variant

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

    'Create the Visum-Object (connect the variable Visum to the software Visum)
    Set Visum = CreateObject("Visum.Visum.120")
    'load version, filename from cell B5
    Visum.LoadVersion Cells(5, 2)

```

```

'access node by number (key), the number is written in cell B6
Set aNode = Visum.Net.Nodes.ItemByKey(Cells(6, 2))

'read node attributes and write it in cells
Cells(13, 1) = aNode.AttValue("CODE")
Cells(13, 2) = aNode.AttValue("NAME")
Cells(13, 3) = aNode.AttValue("TYPEENO")

'delete all objects to close Visum-software
Set aNode = Nothing
Set Visum = Nothing

End Sub

```

6.2.2 Loop using Object Enumeration

The example demonstrates access to nodes inside a loop using object enumeration. It loops over all objects and writes some attribute values to an excel sheet.

Example file: Loop_Node.xls

```

'This example shows how to access single nodes of a net by using ItemByKey
Sub NodeAttributes()

    'declaration of variables
    Dim Visum As Object                                ' Variable Visum
    Dim aNode As Variant

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

    'Create the Visum-Object (connect the variable Visum to the software Visum)
    Set Visum = CreateObject("Visum.Visum.120")
    'load version, filename from cell B3
    Visum.LoadVersion Cells(5, 2)

    'access node by number (key), the number is written in cell B6
    Set aNode = Visum.Net.Nodes.ItemByKey(Cells(6, 2))

    'read node attributes and write it in cells
    Cells(13, 1) = aNode.AttValue("CODE")
    Cells(13, 2) = aNode.AttValue("NAME")
    Cells(13, 3) = aNode.AttValue("TYPEENO")

    'delete all objects to close Visum-software
    Set aNode = Nothing
    Set Visum = Nothing

End Sub

```

6.2.3 Loop using Item

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

Example file: GetItem_Node.xls

```

'This example shows how to access single nodes of a net by using Item.
'Attention: Item identifies a node by its index in the INodes-Container-Object.
'A Node cannot be uniquely identified through the index, the index is subject

```

```

'to change if the Visum network is altered.
'Accessing by Item should only be used with loops (i.e. processing all available
'nodes).
Sub NodeAttributItemExample()

    'declaration of variables
    Dim Visum As Object
    Dim aNode As Variant
    Dim AllNodes As Variant

    Dim row As Long

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

    'create the Visum-Object (connect the variable Visum to the software Visum)
    Set Visum = CreateObject("Visum.Visum.120")
    'Load version, filename from cell B5
    Visum.LoadVersion Cells(5, 2)

    'get all Nodes
    AllNodes = Visum.Net.Nodes.GetAll
    'access single node
    For i = 0 To UBound(AllNodes)
        row = row + 1

        Set aNode = AllNodes(i)

        'read node attributes and write it in cells
        Cells(row, 1) = aNode.AttValue("NO")
        Cells(row, 2) = aNode.AttValue("CODE")
        Cells(row, 3) = aNode.AttValue("NAME")
        Cells(row, 4) = aNode.AttValue("TYPENO")
    Next

    'delete all objects to close Visum-software
    Set aNode = Nothing
    Set AllNodes = Nothing
    Set Visum = Nothing

End Sub

```

6.2.4 Loop via For...Each

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

Example file: ForEach_Node.xls

```

'This example shows how to access single nodes with For-Each-Loop.
Sub NodeAttributForEachExample()

    'declaration of variables
    Dim Visum As Object
    Dim AllNodes() As Variant

    Dim row As Long

    'clear all rows in worksheet from number 12 to the end
    Rows("12:65536").ClearContents
    row = 11

    'create the Visum-Object (connect the variable Visum to the software Visum)
    Set Visum = CreateObject("Visum.Visum.120")

```

```

'Load version, filename from cell B5
Visum.LoadVersion Cells(5, 2)

'get all Nodes
AllNodes = Visum.Net.Nodes.GetAll
'access single node
For Each Node In AllNodes
    row = row + 1

    'read node attributes and write it in cells
    Cells(row, 1) = Node.AttValue("NO")
    Cells(row, 2) = Node.AttValue("CODE")
    Cells(row, 3) = Node.AttValue("NAME")
    Cells(row, 4) = Node.AttValue("TYPENO")
Next

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

End Sub

```

6.2.5 Loop using Iterator Object

The example demonstrates a loop that is set up using the Iterator object of the ILinks container object. The program multiplies the capacity of all links by 2.

Example file: Iterator.xls

```

Sub IteratorExample()

    'usual things to do
    Dim Visum As Object
    Set Visum = CreateObject("Visum.Visum.120")
    Visum.LoadVersion Cells(5, 2)

    'declare Variables
    Dim LinkList As Variant
    Dim LinkIter As Variant
    Dim CurLink As Variant

    'get ILinks object
    Set LinkList = Visum.Net.Links
    'create IIterator object
    Set LinkIter = LinkList.Iterator

    'loop using IIterator object
    While LinkIter.Valid
        Set CurLink = LinkIter.Item
        CurLink.AttValue("CAPPRT") = CurLink.AttValue("CAPPRT") * 2
        LinkIter.Next
    Wend

    'usual things to do
    Visum.SaveVersion Cells(6, 2)
    Set Visum = Nothing
End Sub

```

6.2.6 Reading and Saving Individual Attributes

The following example shows the access for reading and writing attributes. The running time of the buses are modified in order of the volumes of the cars.

Example file: WriteTBus.xls

```

Sub WriteTBusExample()

    'usual things to do
    Dim Visum As Object
    Set Visum = CreateObject("Visum.Visum.120")
    Visum.LoadVersion Cells(5, 2)

    'declare of variables
    Dim AllLinks() As Variant
    Dim vol As Double
    Dim length As Double
    Dim rtime As Integer

    'get all Links
    AllLinks = Visum.Net.Links.GetAll
    For Each Link In AllLinks
        'read link attributes
        vol = Link.AttValue("VolVeh_TSys(C,AP)")
        rtime = Link.AttValue("T_PUTSYS(B)")
        length = Link.AttValue("Length")

        'calculate new runtime
        rtime = rtime + (vol / 1800) * length

        'write new runtime
        Link.AttValue("T_PUTSYS(B)") = rtime
    Next

    'usual things to do
    Visum.SaveVersion Cells(6, 2)
    Set Visum = Nothing
End Sub

```

6.2.7 Reading and Saving Attributes of Several Network Objects

The program reads the values of AddValue1 for all zones using GetMultiAttValues, transforms then into relative values and writes back these new values to the zone attribute. The column types of the value array is printed to the excel worksheet.

Example file: SetRelativeZoneAddVal.xls

```

Sub SetRelativeZoneAddValExample()
    'usual things to do
    Dim Visum As Object
    Set Visum = CreateObject("Visum.Visum.120")
    Visum.LoadVersion Cells(5, 2)

    'declare variables
    Dim Result As Variant
    Dim i, Num As Long
    Dim Sum As Double
    Dim Type1, Type2 As String

    'get attribute array of AddVal1 values
    Result = Visum.Net.Zones.GetMultiAttValues("AddVal1", False)
    Num = UBound(Result)

    'get field types and print them on the Excel worksheet
    Type1 = TypeName(Result(0, 0))
    Type2 = TypeName(Result(0, 1))
    Cells(13, 1) = Type1
    Cells(13, 2) = Type2

    'read the attribute array

```



```

For i = 0 To Num
    Sum = Sum + Result(i, 1)
Next i
If (Sum = 0) Then
    Sum = 1
End If

'change values in the array to relative values
For i = 0 To Num
    Result(i, 1) = Result(i, 1) * 100 / Sum
Next i

'write attribute array
Visum.Net.Zones.SetMultiAttValues "AddVall", Result, False

'usual things to do
Visum.SaveVersion Cells(6, 2)
Set Visum = Nothing
End Sub

```

6.2.8 Invalid Attribute Values

The program shows the calculation status of some attributes before and after execution of an assignment. If there are no assignment results in the version file, the status of „volume“ is „undefined“ first and turns to defined after the assignment, whereas the status of „capacity“ is always true, since it does not depend on any calculation.

Example file: UndefinedValues.xls

```

Sub UndefinedValuesExample()
    'precondition: the version file does not contain an assignmet result
    'usual things to do
    Dim Visum As Object
    Set Visum = CreateObject("Visum.Visum.120")
    Visum.LoadVersion Cells(5, 2)

    'declare variables
    Dim AllLinks As Variant
    Dim Link As Variant
    Dim Length As Variant
    Dim VNull As Variant
    Dim TBus As Variant

    Set AllLinks = Visum.Net.Links

    'show attribute states before assignment
    Cells(12, 2) = AllLinks.AttState("CAPPRT")
    Cells(13, 2) = AllLinks.AttState("VOLVEHPRT (AP) ")

    Visum.Procedures.Execute

    'show attribute states after assignment
    Cells(12, 3) = AllLinks.AttState("CAPPRT")
    Cells(13, 3) = AllLinks.AttState("VOLVEHPRT (AP) ")

    'usual things to do
    Set Visum = Nothing
End Sub

```

6.2.9 Creating a Network

The following example demonstrates how to create a complete network using the add methods of the INet object. This principle can be used to insert data from other sources into a Visum network.

Example file: BuildNetwork.xls

```
Sub BuildNetwork()

    'usual things to do
    Dim Visum As Object
    Dim Net As Variant
    Set Visum = CreateObject("Visum.Visum.120")

    'clear network for safety
    Set Net = Visum.Net

    'transport systems
    Net.AddTSystem "Car", "PRT"
    Net.AddTSystem "Train", "PUT"
    Net.AddTSystem "Walk", "PUTWALK"

    'linktypes always exist, just set attribute values
    Dim linktype20 As Variant
    Dim linktype30 As Variant
    Set linktype20 = Net.LinkTypes.ItemByKey(20)
    Set linktype30 = Net.LinkTypes.ItemByKey(30)
    linktype20.AttValue("TSYSSET") = "Car"
    linktype20.AttValue("CAPPRT") = 800
    linktype20.AttValue("V0PRT") = 40
    linktype20.AttValue("VMAX_PRTSYS(Car)") = 40
    linktype30.AttValue("TSYSSET") = "Car,Train"
    linktype30.AttValue("CAPPRT") = 1000
    linktype30.AttValue("V0PRT") = 50
    linktype30.AttValue("VMAX_PRTSYS(Car)") = 50
    linktype30.AttValue("VDEF_PUTSYS(Train)") = 30

    'nodes
    Dim node1 As Variant
    Dim node2 As Variant
    Dim node3 As Variant
    Dim node4 As Variant
    Dim node5 As Variant
    Set node1 = Net.addnode(1)
    Set node2 = Net.addnode(2)
    Set node3 = Net.addnode(3)
    Set node4 = Net.addnode(4)
    Set node5 = Net.addnode(5)

    'set coordinates; default values remain at "0"
    node1.AttValue("XCOORD") = 1000
    node1.AttValue("YCOORD") = 1000
    node2.AttValue("YCOORD") = 1000
    node4.AttValue("XCOORD") = 1000
    node5.AttValue("XCOORD") = 500
    node5.AttValue("YCOORD") = 500

    'links
    Net.AddLink 1, 1, 2, 30
    Net.AddLink 2, 2, 3, 30
    Net.AddLink 3, 3, 4, 30
    Net.AddLink 4, 4, 1, 30
    Net.AddLink 5, 1, 5, 20
    Net.AddLink 6, 2, 5, 20
    Net.AddLink 7, 3, 5, 20
    Net.AddLink 8, 4, 5, 20
```

```

'zones and connectors
Dim zone1 As Variant
Dim zone2 As Variant
Set zone1 = Net.AddZone(1)
Set zone2 = Net.AddZone(2)
zone1.AttValue("NAME") = "zone 1"
zone1.AttValue("XCOORD") = 1100
zone1.AttValue("YCOORD") = 1100
zone2.AttValue("NAME") = "zone 2"
zone2.AttValue("XCOORD") = -100
zone2.AttValue("YCOORD") = -100
Net.AddConnector zone1, node1
Net.AddConnector zone2, node3

'stops, stopareas and stoppoints
Dim stop1 As Variant 'must be Variant because stop is a reserved word in VBA
Dim stop2 As Variant
Dim stoparea1 As Variant
Dim stoparea2 As Variant
Dim stoppoint1 As Variant 'must be Variant because NetElements.Add accepts
Dim stoppoint2 As Variant 'only Variants
Set stop1 = Net.AddStop(1)
Set stoparea1 = Net.AddStopArea(1, 1, 1)
Set stoppoint1 = Net.AddStopPointOnNode(1, 1, 1)
Set stop2 = Net.AddStop(2)
Set stoparea2 = Net.AddStopArea(2, 2, 3)
Set stoppoint2 = Net.AddStopPointOnNode(2, 2, 3)

'line and lineroutes
Dim line1 As Variant
Dim lineroute1 As Variant
Dim lineroute2 As Variant
Dim myrouteforward As Variant
Dim myroutebackward As Variant
Dim direction1 As Variant
Dim direction2 As Variant
Dim routesearchparameters As Variant

Set line1 = Net.AddLine("line 1", "Train")

Set direction1 = Net.Directions.ItemByKey(">")
Set direction2 = Net.Directions.ItemByKey("<")

Set myrouteforward = Visum.CreateNetElements
myrouteforward.Add (stoppoint1)
myrouteforward.Add (stoppoint2)

Set myroutebackward = Visum.CreateNetElements
myroutebackward.Add (stoppoint2)
myroutebackward.Add (stoppoint1)

Set routesearchparameters = Visum.CreateNetReadRouteSearchTSys
routesearchparameters.SearchShortestPath 1, False, False, 2#, 0, 1

Set lineroute1 = Net.AddLineRoute("lineroute 1", line1, direction1,
myrouteforward, routesearchparameters)
Set lineroute2 = Net.AddLineRoute("lineroute 2", line1, direction2,
myroutebackward, routesearchparameters)

'timeprofiles
Dim tp1 As Variant
Dim tp2 As Variant
Dim tpis As Variant
Dim tpi1 As Variant
Dim tpi2 As Variant

Set tp1 = Net.AddTimeProfile("timeprofile 1", lineroute1)
Set tp2 = Net.AddTimeProfile("timeprofile 2", lineroute2)

tpis = tp1.TimeProfileItems.GetAll

```

```

Set tpi1 = tpis(1)
tpi1.AttValue("Arr") = 120
tpi1.AttValue("Dep") = 120 'because Dep=Arr at the last TP item
tpis = tp2.TimeProfileItems.GetAll
Set tpi2 = tpis(1)
tpi2.AttValue("Arr") = 120
tpi2.AttValue("Dep") = 120

'save version file
Dim filename As String
filename = Cells(5, 2)
Visum.SaveVersion filename

Set linktype20 = Nothing
Set linktype30 = Nothing
Set node1 = Nothing
Set node2 = Nothing
Set node3 = Nothing
Set node4 = Nothing
Set node5 = Nothing
Set zone1 = Nothing
Set zone2 = Nothing
Set stop1 = Nothing
Set stop2 = Nothing
Set stoparea1 = Nothing
Set stoparea2 = Nothing
Set stoppoint1 = Nothing
Set stoppoint2 = Nothing
Set line1 = Nothing
Set lineroute1 = Nothing
Set lineroute2 = Nothing
Set myrouteforward = Nothing
Set myroutebackward = Nothing
Set direction1 = Nothing
Set direction2 = Nothing
Set routesearchparameters = Nothing
Set tp1 = Nothing
Set tp2 = Nothing
Set tpis = Nothing
Set tpi1 = Nothing
Set tpi2 = Nothing
Set Visum = Nothing
End Sub

```

6.2.10 Creating OD Connector Links

The example replaces every OD connector in the network by a short link between the connected node and a new, artificial node and a new OD connector connecting this new node to the zone. Using this construction direct connections to nodes with more than two legs can be avoided.

Example file: CreateConnectorLinks.xls

```

Sub createConnectorLinks()

'usual things to do
Dim i As Integer
Dim Visum As Object
Set Visum = CreateObject("Visum.Visum.120")
Visum.LoadVersion Cells(5, 2)

'get minimum link number for inserted links
Dim linkNos As Variant
Dim nextLinkNo As Integer
linkNos = Visum.Net.Links.GetMultiAttValues("NO", True)
nextLinkNo = 0

```

```

For i = 0 To UBound(linkNos)
    If linkNos(i, 1) > nextLinkNo Then
        nextLinkNo = linkNos(i, 1)
    End If
Next i
nextLinkNo = nextLinkNo + Cells(9, 2)

'get minimum node number for inserted links
Dim nodeNos As Variant
Dim nextNodeNo As Integer
nodeNos = Visum.Net.Nodes.GetMultiAttValues("NO", True)
nextNodeNo = 0
For i = 0 To UBound(nodeNos)
    If nodeNos(i, 1) > nextNodeNo Then
        nextNodeNo = nodeNos(i, 1)
    End If
Next i
nextNodeNo = nextNodeNo + Cells(8, 2)

'get linktype used for additional links
Dim linktype As Integer
linktype = Cells(7, 2)

'for each connector create a node, a link
'and a new connector
Dim oldConnectors As Variant
Dim oldConn As Variant
Dim zoneX, zoneY, nodeX, nodeY, viaX, viaY

oldConnectors = Visum.Net.Connectors.GetAll
For i = 0 To UBound(oldConnectors) Step 2
    Set oldConn = oldConnectors(i)

    'calculate coordinates for intermediate node
    zoneX = oldConn.AttValue("ZONE\XCOORD")
    zoneY = oldConn.AttValue("ZONE\YCOORD")
    nodeX = oldConn.AttValue("NODE\XCOORD")
    nodeY = oldConn.AttValue("NODE\YCOORD")
    viaX = (zoneX + nodeX) / 2
    viaY = (zoneY + nodeY) / 2

    'get referenced objects
    Dim thezone As Variant
    Dim thenode As Variant
    Set thezone = Visum.Net.Zones.ItemByKey(oldConn.AttValue("ZONE\NO"))
    Set thenode = Visum.Net.Nodes.ItemByKey(oldConn.AttValue("NODE\NO"))

    'create viaNode
    Dim viaNode As Variant
    Set viaNode = Visum.Net.AddNode(nextNodeNo)
    nextNodeNo = nextNodeNo + 1
    viaNode.AttValue("XCOORD") = viaX
    viaNode.AttValue("YCOORD") = viaY

    'create newConnector
    Visum.Net.AddConnector thezone, viaNode

    'create newLink
    Dim newLink As Variant
    Set newLink = Visum.Net.AddLink(nextLinkNo, viaNode, thenode)
    nextLinkNo = nextLinkNo + 1
    newLink.AttValue("TYPENO") = linktype

    'delete old connector
    Visum.Net.RemoveConnector (oldConn)
Next i

'save version and exit
Visum.SaveVersion Cells(6, 2)
Set oldConnectors = Nothing

```

```

Set thezone = Nothing
Set thenode = Nothing
Set newLink = Nothing
Set viaNode = Nothing
Set Visum = Nothing
End Sub

```

6.3 Assignment

6.3.1 Executing Assignments from the Assignment Parameters File

The example reads in an OD matrix from file. Then an assignment parameter file is read in and the assignment is executed.

Example file: StartAssignment.xls

```

Sub StartAssignment()

    'usual things to do
    Dim Visum As Object
    Dim Ass As Object
    Dim DSeg As Object
    Dim Proc As Object
    Dim filesys As Object

    Set Visum = CreateObject("Visum.Visum.120")
    Visum.LoadVersion Cells(6, 2)

    'initialize all filters
    Visum.Filters.InitAll

    'read O-D matrix
    Set DSeg = Visum.Net.DemandSegments.ItemByKey(CStr(Cells(8, 2)))
    DSeg.ODMatrix.Open Cells(5, 2) & "\PUT.fma"

    'read assignment parameters
    Set Proc = Visum.Procedures
    Proc.Open Cells(5, 2) & "\autoequi.par"
    Proc.Execute

    'save version, filename from cell B4
    Visum.SaveVersion Cells(7, 2)

    'stop to show you that Visum has loaded the version
    MsgBox "Stop"

    'delete the Visum-Objekt to close the Visum-software
    Set Ass = Nothing
    Set Visum = Nothing
End Sub

```

6.4 Lists

6.4.1 Creating a List and Exporting Data as an Attribute File

The example demonstrates the access to lists. An empty link list is created, filled with data from the selected objects, and the set of columns is determined. Then the data is exported as an attribute file.

Example file: LinkListToAttributeFile.xls

```
Sub LinkListToAttributeFile()
    'usual things to do
    Dim Visum As Object
    Dim LinkList As Object
    Set Visum = CreateObject("Visum.Visum.120")
    Visum.LoadVersion Cells(5, 2)

    'create empty link list
    Set LinkList = Visum.Lists.CreateLinkList

    'select objects to be shown in list. Here: only active links
    LinkList.SetObjects (True)

    'select columns to be shown in list
    'include all key columns to get a re-readable attribute file
    LinkList.AddKeyColumns
    LinkList.AddColumn "TSYSSET"

    'length in meters, 2 decimal places and unit
    LinkList.AddColumn "LENGTH", 13, 2, True

    'volume of DSeg X, transport system "Bus", analysis horizon
    LinkList.AddColumn ("VOLPERS_DSEG_TSYS(P-B,AH)")

    'save list to attribute file
    LinkList.SaveToAttributeFile Cells(6, 2), 59

    Set LinkList = Nothing
    Set Visum = Nothing
End Sub
```

6.4.2 Route Analysis

The example program reads the routes belonging to a certain demand segment from the assignment result and prints length, travel time and node chain to the excel worksheet for every route. There must be routes, i.e. an assignment must be calculated in the network.

Example file: RouteAnalysis.xls

```
Sub RouteAnalysis()

    'declaration of variables
    Dim Visum As Object
    Dim DSeg As Object
    Dim NodeList As Object
    Dim OZone As Object
    Dim Iter As Object

    'usual things to do
    Set Visum = CreateObject("Visum.Visum.120")
    Visum.LoadVersion Cells(5, 2)
```

```

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

'error handling when no valid demand segment is found
On Error GoTo No_DemandSegment

'get the demand segment code from cell B6
Code = Cells(6, 2).Value
Set DSeg = Visum.Net.DemandSegments.ItemByKey(Code)

'get first O-Zone for use in SetObjects-statement
Set Iter = Visum.Net.Zones.Iterator
Set OZone = Iter.Item

'create PrTPathList and add columns
Set PathList = Visum.Lists.CreatePrTPathList
PathList.SetObjects OZone, DSeg, routeFilter_filterAllRoutes
PathList.AddColumn ("OrigZoneNo")
PathList.AddColumn ("DestZoneNo")
PathList.AddColumn ("Index")
PathList.AddColumn "Vol(AP)", defaultfmt, 3
PathList.AddColumn ("tCur")
PathList.AddColumn "Length", longlength, 3

'get PrTPathList
List = PathList.SaveToArray()
For Row = 0 To UBound(List)
    Cells(Row + 13, 1) = List(Row, 0)
    Cells(Row + 13, 2) = List(Row, 1)
    Cells(Row + 13, 3) = List(Row, 2)
    Cells(Row + 13, 5) = List(Row, 5)
    Cells(Row + 13, 6) = List(Row, 4)
    Cells(Row + 13, 7) = List(Row, 3)

    'get nodes of current path and collect their numbers in string
    Set NodeList = DSeg.GetPathNodes(List(Row, 0), List(Row, 1), List(Row, 2))
    AllNodes = NodeList.GetAll
    Cells(Row + 13, 4) = UBound(AllNodes)
    FromNode = AllNodes(1).AttValue("No")
    TheNodes = Str(FromNode)
    For j = 2 To UBound(AllNodes)
        ToNode = AllNodes(j).AttValue("No")
        TheNodes = TheNodes & " -" & Str(ToNode)
    Next j
    Cells(Row + 13, 8) = TheNodes
Next Row

Set DSeg = Nothing
Set NodeList = Nothing
Set OZone = Nothing
Set Iter = Nothing
Set Visum = Nothing
Exit Sub

'error handling
No_DemandSegment:

MsgBox "Demand Segment does not exist!", vbCritical, "Error"
'delete object to close Visum-software
Set Visum = Nothing
End Sub

```


6.5 Filters

6.5.1 Defining Filters via COM

The following figure shows how filter criteria are defined via COM.

Example file: DefineFilters.xls

```
Sub DefineFilters()

    'variable declaration of variable Visum
    Dim Visum As Object

    'create the Visum-Object (connect the variable Visum to the software Visum)
    Set Visum = CreateObject("Visum.Visum.120")

    'load version, file name from cell E2
    Visum.LoadVersion Cells(5, 2)

    'initialize all filters
    Visum.Filters.InitAll

    'define zone filter (ISingleFilter)
    Set ZoneFilter = Visum.Filters.ZoneFilter
    ZoneFilter.UseFilter = True    'default
    ZoneFilter.Complement = False 'default
    ZoneFilter.AddCondition "OP_NONE", False, "XCOORD", "ContainedIn",
        Visum.Filters.Range(11.11, 22.22)
    ZoneFilter.AddCondition "OP_OR", False, "TYPENO", "ContainedIn", "3,4,5,6,7"
    ZoneFilter.AddCondition "OP_AND", True, "RELATIVESTATE", "ContainedIn",
        "RELSTATEINTERNAL"

    'define link filter (IDirectedFilter)
    Set LinkFilter = Visum.Filters.LinkFilter
    LinkFilter.UnDirected = False
    LinkFilter.UseFilter = True
    LinkFilter.AddCondition "OP_NONE", False, "NO", "ContainedIn",
        Visum.Filters.Range(100, 10000)

    'define POI filter for single POI category
    Set POI_one_Filter = Visum.Filters.POIFilter(1)
    POI_one_Filter.AddCondition "OP_NONE", False, "IMAGEHEIGHT", "ContainedIn",
        Visum.Filters.Range(100, 200)

    'define stop filter (hierarchical filter)
    Set StopGroupFilter = Visum.Filters.StopGroupFilter
    StopGroupFilter.UseFilterForStops = True
    StopGroupFilter.UseFilterForStopAreas = False
    StopGroupFilter.UseFilterForStopPoints = True
    StopGroupFilter.StopFilter.UseFilter = True
    StopGroupFilter.StopFilter.AddCondition "OP_NONE", False, "ADDVAL2",
        "ContainedIn", Visum.Filters.Range(-11, 11)
    StopGroupFilter.StopPointFilter.UseFilter = True
    StopGroupFilter.StopPointFilter.UseSelection = True

    'set selection for filter on stop point level
    Dim StopPointContainer As Object
    Set StopPointContainer = Visum.CreateNetElements
    Dim stopPoint1 As Object
    Set stopPoint1 = Visum.Net.StopPoints.ItemByKey(100030)
    Dim stopPoint2 As Object
    Set stopPoint2 = Visum.Net.StopPoints.ItemByKey(100031)
    Dim stopPoint3 As Object
    Set stopPoint3 = Visum.Net.StopPoints.ItemByKey(100032)
    StopPointContainer.Add stopPoint1
```

```

StopPointContainer.Add stopPoint2
StopPointContainer.Add stopPoint3
StopGroupFilter.StopPointFilter.SetSelection StopPointContainer

'define line group filter (hierarchical filter)
Set LineGroupFilter = Visum.Filters.LineGroupFilter
LineGroupFilter.UseFilterForLines = True
LineGroupFilter.UseFilterForLineRoutes = True
LineGroupFilter.UseFilterForLineRouteItems = True
LineGroupFilter.UseFilterForTimeProfiles = False
LineGroupFilter.UseFilterForTimeProfileItems = False
LineGroupFilter.UseFilterForVehJourneys = False
LineGroupFilter.UseFilterForVehJourneyItems = False
LineGroupFilter.UseFilterForVehJourneySections = False
LineGroupFilter.LineRouteFilter.UseFilter = True
LineGroupFilter.LineRouteFilter.UseSelection = True

'set selection for filter on line route level
Dim LineRouteContainer As Object
Set LineRouteContainer = Visum.CreateNetElements
Dim LineRoute1 As Object
Set LineRoute1 = Visum.Net.LineRoutes.ItemByKey("002", ">", "1_H")
Dim LineRoute2 As Object
Set LineRoute2 = Visum.Net.LineRoutes.ItemByKey("S2", "<", "1_R")
LineRouteContainer.Add LineRoute1
LineRouteContainer.Add LineRoute2
LineGroupFilter.LineRouteFilter.SetSelection LineRouteContainer

'save filter file
Visum.Filters.Save Cells(6, 2)

' say goodbye
Set Visum = Nothing
End Sub

```

6.6 Reports and Analyses

6.6.1 Flow Bundles

The example demonstrates the functionality of flow bundles for selected net elements. First, a net element container is created. Net objects are added to this container before it is passed to the flow bundle object as a parameter.

Example file: FlowbundleAnalysis.xls

```

Sub FlowbundleAnalysis()
'declare variables
Dim Visum As Object
Dim Link As Object
Dim Node As Object
Dim NetElementContainer As Object
Dim Flowbundle As Object

'usual things to do
Set Visum = CreateObject("Visum.Visum.120")
Visum.LoadVersion Cells(5, 2)

'create net element container and put net elements in
Set NetElementContainer = Visum.CreateNetElements
Set Flowbundle = Visum.Net.DemandSegments.ItemByKey(Cells(7, 2).Value).Flowbundle
Set Link = Visum.Net.Links.ItemByKey(11, 20)

```

```

Set Node = Visum.Net.Nodes.ItemByKey(12)
NetElementContainer.Add Link
NetElementContainer.Add Node

'execute flow bundle calculation
Flowbundle.Clear
Flowbundle.Execute NetElementContainer

Visum.SaveVersion Cells(6, 2)

Set Visum = Nothing
End Sub

```

6.6.2 A Slightly More Complicated Flow Bundle

The following code snippet shows how to calculate a flow bundle analysis filtering for all paths originating at zone 100, with destination zone 200 and passing over node 12.

```

... (define variables, open version file etc.)

Set MyFlowBundle = Visum.Net.DemandSegments.ItemByKey("C").Value.FlowBundle

' initialize MyFlowBundle
MyFlowBundle.Clear

' zone, origin traffic only
Set zone100 = visum.Net.Zones.ItemByKey(100)
Set MyActivityTypeSetZone100 = MyFlowBundle.CreateActivityTypeSet
MyActivityTypeSetZone100.Add(1)
MyFlowBundle.CreateCondition zone100, MyActivityTypeSetZone100

' node, any kind of traffic, no condition
Set node = Visum.Net.Nodes.ItemByKey(12)
Set MyActivityTypeSetNode= MyFlowBundle.CreateActivityTypeSet
MyActivityTypeSetNode.Add(0)
MyFlowBundle.CreateCondition node, MyActivityTypeSetNode

' zone, destination traffic only
Set zone200 = visum.Net.Zones.ItemByKey(200)
Set MyActivityTypeSetZone200 = MyFlowBundle.CreateActivityTypeSet
MyActivityTypeSetZone200.Add(2)
MyFlowBundle.CreateCondition zone200, MyActivityTypeSetZone200

' execute MyFlowBundle, conditions combined with the "and then" conjunction
MyFlowBundle.ExecuteCurrentConditions

```

6.7 Matrix Operations

6.7.1 Symmetrizing Matrices with Python

The following example is written in the Python programming language. Beside general access to the Visum object model which is done by importing the „win32com.client“ module, the module "numpy" is required. It provides a wide range of matrix operations. In the example we use the operation „transpose“ to symmetrize an OD matrix. The example demonstrates well the simplicity of the code.

Example file: MatrixOperations.py

```

import win32com.client
Visum = win32com.client.Dispatch("Visum.Visum.120")

```

```

Visum.LoadVersion("C:\\Program
Files\\PTV_Vision\\Visum120\\Examples\\Example_net\\EXAMPLE.ver")

from numpy import *

odMatrix = Visum.Net.ODMatrices.ItemByKey(1)
matrix = array(odMatrix.GetValues())
matrix = (matrix + transpose(matrix)) / 2
odMatrix.SetValues(matrix)

Visum.SaveVersion("C:\\Program
Files\\PTV_Vision\\Visum120\\Examples\\Example_net\\EXAMPLE_2.ver")

Visum = 0

```

6.7.2 Calculating Values for the Diagonal Entries of a Skim Matrix

The script calculates the intrazonal value (the value of the entries on the diagonal) of a skim matrix as a factor times the value for the “nearest” zone.

Example file: calculateIntrazonal.py

```

#Load library
import numpy

#Function to calculate intrazonal
def calcIntrazonal(mat, factor=0.5):
    """Set diagonal of matrix as factor * nearest destination zone
    """

    #Copy matrix
    mat = mat.copy()

    #Set intrazonal
    for i in range(0,mat.shape[0]):
        mat[i,i] = min(mat[i][mat[i]>0]) * factor
    return mat

#####

#Get skim matrix 1
mat = numpy.array(Visum.Net.SkimMatrices.ItemByKey(2).GetValues())

#Calculate intrazonal
mat = calcIntrazonal(mat, 0.5)

#Return skim to Visum
Visum.Net.SkimMatrices.ItemByKey(2).SetValues(mat)

```

6.8 Dialogs and Add-Ins

6.8.1 Selecting Graphic Parameters

This example demonstrates how to add new dialogues to Visum using the Python programming language. A class GPASelector is defined which is derived from the calls Frame with is part of the Tkinter module. Objects of this class contain a list box filled with the file names of all GPA files that can be found in the current directory. The constructor sets this current directory to the Visum project directory for GPA files.

Once started, the dialogue waits in a loop that is interrupted by the WaitForIdle method to allow Visum for drawing. If a file name is chosen from the list box, Visum loads the GPA file and redraws the network window.

Example file: GParFileSelector.py

```
from Tkinter import *
import os

class GPASelector(Frame):

    def createWidgets(self):
        self.LB = Listbox(self, selectmode=SINGLE, width=50)
        count = 0
        for fname in os.listdir(self.Dir):
            if os.path.splitext(fname)[1].upper() == self.Ext:
                self.LB.insert(END, os.path.basename(fname))
                count = count+1
        self.LB.configure(height=count)
        self.LB.pack()

    def listEntryClicked(self, *ignore):
        fullpath=os.path.join(self.Dir, self.LB.get(self.LB.curselection()[0]))
        self.Visum.Graphic.Parameter().Open(fullpath)
        self.Visum.Graphic.WaitForIdle()

    def waitForVisum(self, *ignore):
        self.Visum.Graphic.WaitForIdle()
        self.after(100,self.waitForVisum)

    def __init__(self, visum, master=None):
        Frame.__init__(self, master)
        self.pack()
        self.master.title("Graphic parameters:")
        self.Visum = visum
        self.Dir = visum.GetPath(8) # GPA project directory
        self.Ext = ".GPA"
        self.createWidgets()
        self.LB.bind("<ButtonRelease-1>", self.listEntryClicked)
        self.after(100,self.waitForVisum)

dlg = GPASelector(Visum)
dlg.mainloop()
```

6.8.2 Access to GPA via COM

The following excel VBA code examples show the usage of the GPA interface

Direct access to nodes GPA

```
Visum.Net.GraphicParameters.Nodes.AttValue("DRAW") = True
Visum.Net.GraphicParameters.Nodes.Active.AttValue("ClassificationAttrID") = "TypeNo"
Visum.Net.GraphicParameters.Nodes.Active.AttValue("UseClassifiedMode") = True
Set nodeClasses = Visum.Net.GraphicParameters.Nodes.Active.Classes
For Each class In classes
    class.Point.Symbol.Filling.AttValue("SomeAttr") = 2.0
Next class
```

Access to nodes GPA via container

```
Set allNodes = Visum.Net.Nodes
allNodes.GPA.Passive.Point.AttValue("Draw") = False
```

Access to nodes via generic method

```
Visum.Net.GraphicParameters.NetObjGPA („Nodes“).Active.AttValue(“UseClassifiedMode”) = True
```

Direct access to POI GPA

```
Set Category = Visum.Net.POICategories.ItemByKey(1)
Visum.Net.GraphicParameters.POIs(Category).SetDrawingOrder(3)
Visum.Net.GraphicParameters.POIs(Category).GPA.AttValue(“DRAW”) = True
Visum.Net.GraphicParameters.POIGeneralGPA.AttValue(“POI2StopAreaDisplayType”) = MarginDisplayType_continuousLine
```

Access to POI GPA via container

```
Set Category = Visum.Net.POICategories.ItemByKey(1)
Category.GPA.AttValue(“Draw”) = True
```

Access to link GPA direct and via group

```
Set linkPassiveGPA = Visum.Net.GraphicParameters.Links.Passive
linkPassiveGPA.Line.AttValue(“Draw”) = False
Set linkActiveGPA = Visum.Net.GraphicParameters.LinkGroup.ObjectGPA.Active
linkActiveGPA.AttValue(„UseClassifiedMode“) = True
```

Access to link label GPA

```
Set linkActiveLabelGPA = Visum.Net.GraphicParameters.LinkGroup.LabelGPA.Active
linkActiveLabelGPA.AttValue(“UseClassifiedMode”) = False
linkActiveLabelGPA.Point.Text.AttValue(“AttrID”) = “Name”
```

6.9 Add-in components

The following example refers to the Multi-Edit attribute add-in which comes with Visum and is meant to show you the elements of an add-in. For this purpose, only the program code used to create an add-in is listed. The actual functionality is not of interest in this case; you can find the entire code under the Exe/AddIn directory of Visum.

References: Fehler! Verweisquelle konnte nicht gefunden werden.

The add-in description file (*.VAI) is used to add the add-in to the Visum script menu:

```
<?xml version="1.0" encoding="UTF-8" ?>
<VisumAddInCollection Format="1.0.0">
  <VisumAddIn>
    <AddInID>PTV_MULTIEDITATTR</AddInID>
    <AddInName>Multi-Edit Attribute</AddInName>
    <AddInPath></AddInPath>
    <AddInParaDlgScript>multieditdlg.py</AddInParaDlgScript>
    <AddInBodyScript>multieditBody.py</AddInBodyScript>
    <AddInOperation>true</AddInOperation>
    <AddInAutoMenu>true</AddInAutoMenu>
    <AddInAutoShowPara>true</AddInAutoShowPara>
  </VisumAddIn>
</VisumAddInCollection>
```

Parameter script: extract from multieditdlg.py

```
#import modules
# ... other stuff

class MyDialog(wx.Dialog): # Dialog class
# ... other stuff

    def on_OK(self, event): # Event handler which is called when OK button is
clicked
```

```

param = dict()
param["ObjType"] = self.cboType.GetName()
param["Active"] = self.chkActive.GetValue()

if self.rboxEditorOptions.GetSelection() == 0:
    if self.expr_text_inInfoMode == False:
        param["Expr"] = self.expr_text.GetValue()
    else:
        param["Expr"] = ""
else:
    if self.expr_text_inInfoMode == False:
        param["Expr"] = self.expr
    else:
        param["Expr"] = ""

if self.code_text_inInfoMode == False:
    param["Context"] = self.code_text.GetValue()
else:
    param["Context"] = ""
param["TargetAttrID"] = self.btnTarget.GetAttrID()

if not multieditHelper.CheckParam(param, addIn):
    return

result = multieditHelper.CheckExpression(param, addIn, self.container,
param["Active"], param["TargetAttrID"], Visum)
if result == None or result == []:
    return
addInParam.SaveParameter(param)
if not addIn.IsInDebugMode:
    Terminated.set()
    self.Destroy()

def OnCancel(self, event): # Event handler which is called when Cancel
button is clicked
    if not addIn.IsInDebugMode:
        Terminated.set()
        self.Destroy()
# ... other stuff

```

Every dialog script should contain the following lines at the end of the dialog script in order to add functionality for treating the add-in in debug mode, handling the internationalization settings, handling an optional progress dialog and implements logging functionality, i.e. writing into Visum trace/error file and/or displaying a message box, for more information see 7.11.1 AddIn Class and 7.11.2 AddInParameter Class.

```

if len(sys.argv) > 1:
    addIn = AddIn()
else:
    addIn = AddIn(Visum)

visumParameter = None
if addIn.IsInDebugMode:
    app = wx.PySimpleApp(0)
    Visum = addIn.Visum
else:
    visumParameter = Parameter
addInParam = AddInParameter(addIn, visumParameter)

if addIn.State != AddInState.OK:
    # for further information: loop over list to get all error objects and use
    # AddIn.ErrorObjects[i].ErrorType to determine which error occurred
    # The AddIn.ErrorObjects[i].ErrorMessage contains a message for user
    addIn.ReportMessage(addIn.ErrorObjects[0].ErrorMessage)
else:
    try:
        wx.InitAllImageHandlers()
        dialog_1 = MyDialog(None, -1, "")

```

```

app.SetTopWindow(dialog_1)
dialog_1.Show()

if addIn.IsInDebugMode:
    app.MainLoop()
except:
    addIn.HandleException(addIn.TemplateText.MainApplicationError)
if not addIn.IsInDebugMode:
    Terminated.set()

# end of file

```

Body script: extract from multieditBody.py

Every body script should contain the following lines. At the end of the body script there should be functionality for treating the add-in in debug mode, handling the internationalisation settings, handling an optional progress dialog and implements logging functionality, i.e. writing into Visum trace/error file and/or displaying a message box, for more information see 7.11.1 AddIn Class and 7.11.2 AddInParameter Class.

```

#import modules

# ... other stuff

def Run(param):
    # do something

if len(sys.argv) > 1:
    """ Set translation functionality, because body script is in debug mode and
    the dialog script got not called before
        The Visum object will be created by functionality of class 'AddIn' ( the
    debug sys.argv[2] - parameter is used for this) """
    addIn = AddIn()
else:
    addIn = AddIn(Visum)
    """In else condition (the AddIn was called out of Visum) the AddIn class
    object is needed only to get logging functionality. The passed Visum object
    is needed for logging functionality. Translation functionality must not be
    set because it already got set in dialog script """

if addIn.IsInDebugMode:
    app = wx.PySimpleApp(0)
    Visum = addIn.Visum

if addIn.State != AddInState.OK:
    addIn.ReportMessage(addIn.ErrorObjects[0].ErrorMessage)
else:
    try:
        defaultParam = multieditHelper.GetDefaultParams()
        visumParameter = None
        if not addIn.IsInDebugMode:
            visumParameter = Parameter

        addInParameter = AddInParameter(addIn, visumParameter)
        addInParameter.AddDependency("Context", "Context", '')
        param = addInParameter.Check(True, defaultParam)
        Run(param)
    except:
        addIn.HandleException(addIn.TemplateText.MainApplicationError)

# end of file

```


7 The Python Library VisumPy

7.1 Introduction

Python is quickly becoming more popular as a scripting language for Visum, because third-party libraries, like *numpy*, *matplotlib* and *tkinter*, simplify matrix manipulation and the construction of user-defined GUIs.

Script developers will notice that the same kind of “glue” code is repeatedly needed in Python scripts for Visum. These standard building blocks are being collected in a library of helper functions called VisumPy, facilitating reuse across scripts.

This appendix describes the modules into which VisumPy is structured and the functions contained in these modules. In order to use one of the functions in your script, include the statement

```
from VisumPy.<module name> import <function>
```

in your script, before calling the function. Example: in order to use the functions *GetMulti* and *SetMulti*, write

```
from VisumPy.helpers import GetMulti, SetMulti
```

COMPATIBILITY OF DIFFERENT MATRIX LIBRARIES IN PYTHON

In Python there are several libraries that are used for matrix operations. Two very popular libraries are *numarray* and *numpy*. In the last years *numarray* was the standard, but today is no longer developed. Its successor is *numpy* and only this module will be adapted to changes in Python in the future. The program interfaces of the two modules are similar, but not identical. You cannot use objects from both modules together.

We thus recommend using *numpy* if you want to develop new scripts. At the same time, however, scripts that already exist in *numarray* are still supported to make the changeover smoother. This is why we adapted the functions in VisumPy so that they could optionally be used with either of the matrix libraries.

In this case, the functions that used a *numarray* data structure for parameters (input or output) are problematic.

Example: If in your script you are using the expressions

```
from VisumPy.helpers import GetODMatrix
mat = GetODMatrix(Visum, 1) # supplies numarray array
```

the function *GetODMatrix* creates a *numarray* array as a result. This result is assigned the variable *mat* and can then be edited using other *numarray* methods. If, however, you would like to use it in your *numpy* script, just tell VisumPy that *GetODMatrix* should create a *numpy* array. To do so, import the *numpy* module before you import the VisumPy.helpers:

```
import numpy # Important: before exporting VisumPy
from VisumPy.helpers import GetODMatrix
mat = GetODMatrix(Visum, 1) # supplies numpy array
```

VisumPy notices during its own import that *numpy* has already been imported. In this case, VisumPy uses variants of the functions that *numpy* objects use for input and output.

7.2 Helpers Module

This module contains a collection of miscellaneous helper functions which abbreviate more general Visum COM functions for their most typical use cases.

def GetMulti(container, attribute, activeOnly=False)

Returns the values of an attribute for all objects of a network object type. They are returned as a list with one element per object (in standard key order). Compared to the built-in method GetMultiAttValues, there is no additional index column which is more convenient when using the vector in numpy/numarray.

Parameters

container	The network object container for which attribute values are retrieved, e.g. Visum.Net.Links.
attribute	attribute ID.
activeOnly	If true, only active net objects will be returned (default False).
return value	attribute values listed in key order (type double for numeric attributes, strings for all other attributes).

Example: Assume that nodes 1,2,3 exist, and that the node UDA MyAtt contains the square of the node number. Then

```
GetMulti(Visum.Net.Nodes, "MyAtt")
will return
[1, 4, 9].
```

def SetMulti(container, attribute, values, activeOnly=False)

Sets the values of an attribute for all objects of a network object type. They are passed as a list with one element per object (in standard key order).

Parameters

container	The network object container for which attribute values are retrieved, e.g. Visum.Net.Links.
attribute	string - the attribute ID.
values	list - new values, as many as there are objects in the collection.
activeOnly	If true, only active net objects will be returned (default False).
return value	none.

Example: Assume that nodes 1,2,3 exist, and have a node UDA MyAtt. Then

```
SetMulti(Visum.Net.Nodes, "MyAtt", [-1, -2, -3])
will cause UDA MyAtt to contain the negative of the node number.
```

def GetAttShortName(container, attrID)

Returns the short name of a given net object attribute in the selected Visum language. If the installed Visum-Version does not provide the needed COM-function 'GetShortName' the return value will be the passed attribute ID instead of raising an error.

Parameters

container	The network object container for which attribute values are retrieved, e.g. Visum.Net.Links.
attrID	The ID of the attribute whose short name will be read and returned.
return value	The short name of the attribute with attrID.

Example: Assume that for a GUI internationalization in German you want to display the translation of the attribute "No" of the container Nodes. Then

```
VisumPy.helpers.GetAttShortName(Visum.Net.Nodes, "No")
will return: u'Nr'
```

def GetTableShortName(container, tableName)

Returns the short name of a given net object in the selected Visum language. If the installed Visum-Version does not provide the needed COM-function 'GetTableShortName' the return value will be the passed tableName instead of raising an error.

Parameters

container	The network object container for which attribute values are retrieved, e.g. Visum.Net.Links.
tableName	The name of the net object whose short name will be read and returned.
return value	The short name of the net object.

Example: Assume that for a GUI internationalization in German you want to display the translation of the net object "Nodes". Then

```
VisumPy.helpers.GetTableShortName(Visum.Net.Nodes, "Nodes")
```

will return: u'Knoten'

def GetODMatrix(visum, which)

Gets the values (contents) of an OD matrix as a numpy array.

Parameters

visum	The Visum object (must be passed explicitly)
which	Either a string containing a demand segment code, or the numerical ID of the OD matrix
return value	If N zones exist, the function will return an <i>N-tuple</i> of N-tuples, representing the matrix contents in row-major order.

Example: Assume that the zones1,2,3 exist, then

```
GetODMatrix(Visum, "P")
```

will return the contents of the OD matrix currently associated with demand segment "P" as a tuple like

```
((1,2,3),
 (4,5,6),
 (7,8,9)).
```

def SetODMatrix(visum, which, values, additive=False)

Sets the values (contents) of an OD matrix from a two-dimensional Python sequence type.

Parameters

visum	The Visum object (must be passed explicitly)
which	Either a string containing a demand segment code, or the numerical ID of the OD matrix
values	A square matrix of values in any Python sequence type, including two-dimensional numpy/numarray arrays.
additive	If true, adds values to present contents of the matrix, otherwise replaces present contents (default: False).
return value	none.

Example: Assume that the zones1,2,3 exist, then

```
SetODMatrix(Visum, 2, ((1,2,3),
 (4,5,6),
 (7,8,9)).
```

will replace the contents of the OD matrix with ID 2 by the given values.

def GetSkimMatrix(visum, which, dseg=None)

Gets the values (contents) of a skim matrix as a two-dimensional tuple which can be passed directly to the array constructor of numpy/*numarray*.

Parameters

visum	The Visum object (must be passed explicitly)
which	The numerical ID or code of a skim matrix (e.g. "TT0")
dseg	Demand segment code (default None).
return value	If N zones exist, the function will return an <i>N-tuple</i> of N-tuples, representing the matrix contents in row-major order.

Example: see GetODMatrix.

def SetSkimMatrix(visum, which, values, additive=False)

Sets the values (contents) of a skim matrix from a two-dimensional Python sequence type.

Parameters

visum	The Visum object (must be passed explicitly)
which	The numerical ID of a skim matrix
values	A square matrix of values in any Python sequence type, including two-dimensional numpy/ <i>numarray</i> arrays.
additive	If true, adds values to present contents of the matrix, otherwise replaces present contents (default: False).
return value	none.

Example: see GetODMatrix.

def GetMatrix(Visum, which)

Get a matrix as a numpy array.

Parameters

visum	The Visum object (must be passed explicitly)
which	The numerical ID of a matrix OR the matrix code
return value	numpy.array - matrix contents

Example: Assume that the matrix 42 with code "C" exist, then

```
mat = GetMatrix(Visum, 42)    OR    mat = GetMatrix(Visum, "C")
```

will return the OD matrix as a numpy array.

def SetMatrix(Visum, which, values, additive=False)

Set an OD matrix from a two-dimensional sequence.

Parameters

Visum	The Visum object (must be passed explicitly)
-------	--

which	The numerical ID of a matrix OR the matrix code
values	list(list) or 2D array - matrix contents
additive	If true, adds values to present contents of the matrix, otherwise replaces present contents (default: False).
return value	none.

Example: Assume that the zones1,2,3 and the matrix 2 with the code "C" exist, then

```
SetMatrix(Visum, 2, ((1,2,3),
                     (4,5,6),
                     (7,8,9))) OR
SetMatrix(Visum, "C", ((1,2,3),
                       (4,5,6),
                       (7,8,9)))
```

will replace the contents of the OD matrix with ID 2 by the given values.

def GetMatrixRaw(Visum, which, intoMat=None)

Get an OD matrix as a numpy array.

CAUTION: This function only works, if the script is executed within Visum (from the script menu or within Calculate - Procedures). In scripts executed outside Visum, it will crash Visum!

Parameters

Visum	The Visum object (must be passed explicitly)
which	The numerical ID of a matrix OR the matrix code
intoMat	optionally, an already existing numpy matrix of the right dimensions, which will be overwritten with the contents of the matrix
return value	numpy.array - matrix contents

Example: Assume that the matrix 42 with code "C" exist, then

```
mat = GetMatrixRaw(Visum, 42) OR mat = GetMatrixRaw(Visum, "C")
```

will return the OD matrix as a numpy array.

def SetMatrixRaw(Visum, which, values)

Set an OD matrix from a two-dimensional sequence.

CAUTION: This function only works, if the script is executed within Visum (from the script menu or within Calculate - Procedures). In scripts executed outside Visum, it will crash Visum!

Parameters

Visum	The Visum object (must be passed explicitly)
which	The numerical ID of a matrix OR the matrix code
values	numpy.array - matrix contents
return value	none.

Example: Assume that the matrix 2 with code “C” exist, then

```
A = numpy.array(((1,2,3),
                 (4,5,6),
                 (7,8,9)), dtype=numpy.float64)
SetMatrixRaw(Visum, 42, A)    OR    SetMatrixRaw(Visum, "C", A)
```

will replace the contents of the OD matrix with ID 2 by the given values.

def GetODMatrixRaw(Visum, which, intoMat=None)

Get an OD matrix as a numpy array.

CAUTION: This function only works, if the script is executed within Visum (from the script menu or within Calculate - Procedures). In scripts executed outside Visum, it will crash Visum!

Parameters

Visum	The Visum object (must be passed explicitly)
which	Demand segment code OR matrix number
intoMat	optionally, an already existing numpy matrix of the right dimensions, which will be overwritten with the contents of the matrix
return value	numpy.array - matrix contents

Example: Assume that the matrix with demand segment C exists, then

```
mat = GetODMatrixRaw(Visum, "C")
```

will return the corresponding OD matrix.

def SetODMatrixRaw(Visum, which, values)

Set an OD matrix from a numpy array.

CAUTION: This function only works, if the script is executed within Visum (from the script menu or within Calculate - Procedures). In scripts executed outside Visum, it will crash Visum!

Parameters

Visum	The Visum object (must be passed explicitly)
which	Demand segment code OR matrix number
values	numpy array - matrix contents
return value	none.

Example: see SetMatrixRaw.

def GetSkimMatrixRaw(Visum, which, dseg=None, intoMat=None)

Get a skim matrix as a numpy array.

CAUTION: This function only works, if the script is executed within Visum (from the script menu or within Calculate - Procedures). In scripts executed outside Visum, it will crash Visum!

Parameters

Visum	The Visum object (must be passed explicitly)
-------	--

which	Matrix number or matrix code string
dseg	Demand segment string (default: None)
intoMat	Optionally, an already existing numpy matrix of the right dimensions, which will be overwritten with the contents of the matrix
return value	numpy.array - matrix contents

Example: Assume that the skim matrix with number 3 and skim matrix with matrix code "TT0" and demand segment string "C" exist, then

```
mat1 = GetSkimMatrixRaw(Visum, 3)
mat2 = GetSkimMatrixRaw(Visum, "TT0", "C")
```

will return the corresponding skim matrices.

def SetSkimMatrixRaw(Visum, which, values)

Set a skim matrix from a two-dimensional sequence.

CAUTION: This function only works, if the script is executed within Visum (from the script menu or within Calculate - Procedures). In scripts executed outside Visum, it will crash Visum!

Parameters

Visum	The Visum object (must be passed explicitly)
which	Demand segment code OR matrix number
values	numpy array - matrix contents
return value	none.

Example: see SetMatrixRaw.

def CreateVisum(release=None)

Creates a COM instance of Visum as a Python object. If release is passed, an instance of this release is created.

Parameters

release	An integer number describing the release to be used. Examples: 9 = latest release installed 9.??; 94 = the latest installed Visum 9.4?; 942 = Visum 9.42.
return value	The COM object.

Example: Assume you need an instance of the latest installed Visum 9.4? then

```
visum = CreateVisum(94)
```

will create the corresponding instance.

def CreateObject(progID)

Creates an instance of any COM server as a Python object.

Parameters

progID	The prog ID of the application as a string.
return value	The COM object.

Example: CreateObject("Excel.Application") will start Excel and return a reference to it.

def secs2HHMMSS(secs)

Returns a string representation for a time value given as seconds from midnight.

Parameters

secs	The time value in seconds.
return value	The formatted string.

Example: secs2HHMMSS(3660) will return "01:01 AM:00".

def HHMMSS2secs(hhmmss)

Parses a formatted string and returns the value in seconds from midnight.

Parameters

hhmmss	The formatted string.
Return value	The time value in seconds after midnight.

Example: secs = HHMMSS2secs("10:01 AM:00") returns 36060.

def skimLookup(Visum, code, dseg)

Searches for skim matrix number using skim matrix code and demand segment code.

Parameters

Visum (COM object)	Visum object
code (String)	Matrix code, e.g. "TT0"
dseg (String)	Demand segment code
return value (int or None)	Matrix number (if available)

Example:

```
skimLookup(Visum, "TT0", "C")
```

def attributeExists(container, attribute):

Checks whether there is an attribute with this name for the network object.

Parameters

container (Visum container object)	Container object (e.g. ILinks)
attribute (String)	Attribute name
return value (bool)	True or False

Example:

```
attributeExists(Visum.Net.Zones, "TEST")
```

7.3 TISupport Module

This module contains a family of functions which emulates user-defined attributes (UDAs) with analysis time intervals as subattributes. As long as these are not available as a built-in feature of Visum, they can be emulated by several user-defined attributes, one for each time interval. The IDs (and names) of the attributes are regularly formed from a common prefix (baseattrID) and the code of the time interval. Example: If three time intervals with codes 0700, 0800, 0900 exist, then a baseattrID of "MyAtt" will generate proxy UDAs MyAtt_0700, MyAtt_0800, MyAtt_0900.

Convenience functions are provided for creating, getting and setting such proxy UDAs for all existing time intervals.

def CreateUDAPerTI(visum, container, baseattrID, **kwargs)

Will generate a set of UDAs according to the naming scheme described above. It is an error, if attributes of the generated names already exist.

Parameters

visum	The Visum object (must be passed explicitly)
container	Network object container for which the UDAs shall be created, e.g. Visum.Net.Links
baseattrID	Prefix for generation of attribute IDs
**kwargs	Any number of named parameters of <i>AddUserDefinedAttribute</i> , as e.g. value type, decimal places, etc.
return value	none.

Example: If intervals with the codes 0700, 0800, 0900 exist,

```
CreateUDAPerTI(Visum, Visum.Net.Links, "CO2", vt=2)
```

will create link UDAs CO2_0700, CO2_0800, CO2_0900, of value type double (vt = 2).

def AttrIDsPerTI(visum, baseattrID, predefined=False)

Returns a list of attribute IDs according to the naming scheme described above. Works with both UDAs created by CreateUDAPerTI and predefined attributes with subattribute AHPI.

Parameters

visum	The Visum object (must be passed explicitly)
baseattrID	Prefix for generation of attribute IDs. If the attribute is predefined by Visum and possesses subattributes other than AHPI, then they are passed in parentheses after the attribute ID.
predefined	True means that the attribute is predefined by Visum, False indicates that the attribute was created using CreateUDAPerTI
return value	List of attribute IDs (strings)

Example: If intervals with the codes 0700, 0800, 0900 exist,

```
AttrIDsPerTI(Visum, "CO2")
```

will return the list

```
["CO2_0700", "CO2_0800", "CO2_0900"].
```

The call

```
AttrIDsPerTI(Visum, "IMP_PRTSYS(P)", True)
```

will return the list

```
["IMP_PRTSYS(P,0700)", "IMP_PRTSYS(P,0800)", "IMP_PRTSYS(P,0900)"].
```

def GetMultiPerTI(visum, container, baseattrID, predefined=False)

Returns the values of attributes per time interval. The values are retrieved for all objects of a network object type and all time intervals. They are returned as a matrix with one row per object (in standard key order) and one column per time interval (ascending).

Parameters

visum	The Visum object (must be passed explicitly)
container	The network object container for which attribute values are retrieved, e.g. Visum.Net.Links.
baseattrID	Prefix for generation of attribute IDs. If the attribute is predefined by Visum and possesses subattributes other than AHPI, then they are passed in parentheses after the attribute ID.

predefined	True means that the attribute is predefined by Visum, False indicates that the attribute was created using CreateUDAPerTI
return value	Matrix of attribute values (type double for numeric attributes, strings for all other attributes).

Example: Assume that the intervals with codes 0700, 0800, 0900 and the nodes 1,2,3 exist, and that the node UDA MyAtt_0700 contains the node number itself, MyAtt_0800 contains the square and MyAtt_0900 contains the negative of the node number. Then

```
GetMultiPerTI(Visum, Visum.Net.Nodes, "MyAtt")
```

will return

```
[ [1, 1, -1], [2, 4, -2], [3, 9, -3] ].
```

def SetMultiPerTI(visum, container, baseattrID, values, predefined=False)

Sets the values of attributes per time interval. The values are set for all objects of a network object type and all time intervals. They are passed as a matrix with one row per object (in standard key order) and one column per time interval (ascending).

Parameters

visum	The Visum object (must be passed explicitly)
container	The network object container for which attribute values are retrieved, e.g. Visum.Net.Links.
baseattrID	Prefix for generation of attribute IDs. If the attribute is predefined by Visum and possesses subattributes other than AHPI, then they are passed in parentheses after the attribute ID.
values	Matrix with the new values.
predefined	True means that the attribute is predefined by Visum, False indicates that the attribute was created using CreateUDAPerTI
return value	none.

Example: Assume that the intervals with codes 0700, 0800, 0900 and the nodes 1,2,3 exist, then

```
GetMultiPerTI(Visum, Visum.Net.Nodes, "MyAtt", [ [1, 1, -1], [2, 4, -2], [3, 9, -3] ])
```

will cause UDA MyAtt_0700 to contain the node number itself, MyAtt_0800 the square, and MyAtt_0900 the negative of the node number.

7.4 Depreciated: Tk Module

With Visum 11.51 we introduce an important change to Python script execution using a graphical user interface from within Visum. From Visum 11.51 onwards Visum itself will take care of creating the wxPython application object and running its main loop. For this reason the graphical user interface library Tk becomes deprecated. We recommend the usage of wxPython instead.

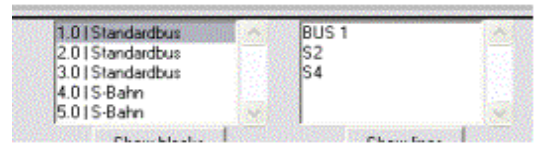
Anyhow, for existing scripts using the Tk module we explain the content of the Tk Module in the following chapters.

This module contains user interface classes for frequent tasks such as prompting the user for a network object selection or displaying a progress dialog for a long task. They are meant to be used within a *Tkinter*-based application. A good knowledge of *Tkinter* programming is assumed.

7.4.1 NetobjSelector Class

This class implements a *Tkinter* widget for a listbox (with scroll bar) that contains all instances of a certain network object type and allows the user to select them.

It supports the following methods:



def __init__(self, master, container, displayattrs, resultattr=None, **kwargs)

The constructor for the widget.

Parameters

master	The parent widget.
container	The network object container for which attribute values are retrieved, e.g. Visum.Net.Links.
displayattrs	A list of attributes of the network object type whose values shall be displayed as a list entry for the network object. A typical choice is the key attribute(s) of the network object type. Attribute values are separated by vertical bars.
resultattr	If a string is passed, it contains the ID of the attribute whose value is returned for each selected object. If the parameter is omitted, the object itself is returned instead of an attribute value.
**kwargs	All keyword parameters of Tkinter.Listbox can be used, in particular for setting the selection mode and the size of the list box.
return value	Object or its attribute value

Example:

```
NetobjSelector(frame, Visum.Net.Blocks,
               displayattrs=["ID", "VEHCOMBNAME"],
               height=5,
               selectmode=Tk.EXTENDED)
```

will insert into frame a listbox with all Block objects. List entries look like "17 | Bus" where 17 is the block ID and "Bus" is the name of the vehicle combination. Multiple selection is allowed and the selection will be returned as a list of Block objects.

```
NetobjSelector(frame, Visum.Net.Lines,
               displayattrs=["NAME"],
               resultattr="NAME",
               height=5,
               selectmode=Tk.SINGLE)
```

will insert into frame a listbox with all Line objects. The line name is used as the list entry and the selection is also returned as the line name. Only single selection is allowed.

def getFrame(self)

The widget is internally implemented as a frame that contains a list box and a scroll bar. getFrame returns the frame object to give it a specific format or layout. Normally not needed.

def getListBox(self)

Ditto for the internal Listbox object.

def getSelection(self)

Returns the current selection as a list. The contents of the list is determined by the value of resultattr in the constructor. A single selection will be returned as a list of length one.

def setSelectionInd(self, newsel)

Sets the (single) selection to newsel which must be one of the strings in the listbox.

Parameters

newsel	One of the strings in the list box.
--------	-------------------------------------

def grid(self, **kwargs) / def pack(self, **kwargs)

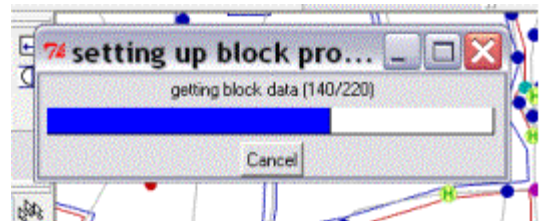
These shortcuts apply the Tkinter layout functions to the Frame of the widget.

Parameters

**kwargs	Keyword parameter of Tkinter.Frame.
----------	-------------------------------------

7.4.2 ProgressDlg Class

This class implements a progress dialog for tasks that may take a long time. It contains a text field for the description of the current activity, a progress bar and a cancel button.



Construct the progress dialog at the beginning of a long task, then repeatedly call the setMessage method and finally destroy the dialog. Optionally, you can call isAborted to check whether the user has clicked the Cancel button.

def __init__(self, title="Progress", **kwparams):

The constructor for the dialog.

Parameters

Title	Window title of the progress dialog
**kwargs	All keyword parameters of Tkinter.Toplevel can be used, in particular to set the size, border and colors of the dialog.
return value	The object.

def close(self)

Closes the dialog.

def isAborted(self)

Returns a Boolean value. True means that the user has clicked the cancel button. As a side effect this will close the dialog. All other reactions, like quitting the long task, must be programmed individually.

def setMessage(self, msg, progress = 0, total = 100)

Sets the current progress message in the dialog.

Parameters

msg	Text of message
progress, total	If progress = x and total = y are passed, i.e. the progress is at "x out of y". The progress bar is drawn

accordingly, and the string "(x/y)" is appended to the message text.

return value

none.

Example:

```
allblocks = Visum.Net.Blocks.GetAll
numblocks = len(allblocks)
pd = ProgressDialog(title="setting up block profile")
for i,block in enumerate(allblocks):
    if (i % 10 == 0):
        pd.setMessage("working on block", i, numblocks)
        if pd.isAborted():
            # quit the function
        # do something long with the block
pd.close()
```

7.4.3 Other Functions

def fileChooser(title, fileTypes=[("","*.*)], startDir=os.getcwd())

File chooser dialog.

Parameters

Title (String)	Window title
fileTypes (list(tuple))	Pairs for file types, see filetypes argument of tkFileDialog. Default fileTypes argument is no file type filter (*.*)
startDir (String)	Name of directory in which the dialog is started
return value (String)	File name as a string.

Example:

Create a file chooser dialog that asks for a filter file:

```
fileChooser("Choose filter file", [('Filter File','*.fil')])
```

Dito, but the dialog starts in C:/project:

```
fileChooser("Choose filter file", [('Filter File','*.fil')], "C:/project")
```

def messageBox (message, title="Info")

Message box.

Parameters

message	Text of message
title	Window title, default is "Info"
return value	none.

Example:

Notify user that the script is complete:

```
messageBox("Script Complete")
```

7.5 wxHelpers Module

The wxHelpers module contains helper classes for creating widgets in wxPython dialogs that allow you to access certain functionalities from Visum.

7.5.1 wxAttrIDButton Class

This class implements a wx widget for a button that allows the user to select an attribute of a specific network object. Multiple selection is not supported. The class is derived from the wx.Button class.

def __init__(self, parent, *args, **kwargs)

The constructor for the class. Since most popular GUI images, as e.g. wxGlade, do not support any additional arguments that allow you to adapt widgets, these are adapted via the methods described in the following and not directly via the constructor.

Parameters

parent	The parent widget
--------	-------------------

def SetContainer(self, container)

Sets the network element collection the user selected an attribute for. The method must be performed before the user can click the button.

Parameters

container (object)	Object of Visum network element collection
--------------------	--

Example:

Creating a button that allows you to show all link attributes:

```
btnAttr.SetContainer(Visum.Net.Links)
```

def SetAttrType(self, editableOnly = False, numericOnly = False)

Filters the attributes to be shown. You can call this method optionally.

Parameters

editableOnly (bool)	If True, only input attributes are displayed.
numericOnly (bool)	If True, only numeric attributes are displayed.

Example:

Creating a button that allows you to show numeric attributes (input attributes and calculated attributes) only:

```
btnAttr.SetAttrType(numericOnly=True)
```

def SetAttrID(self, attrID)

Sets the current selection to the attribute ID. To specify indirect attributes, use the common syntax "\\" (backslash). You can call this method optionally.

Parameters

attrID (string)	ID of attribute that is set as a new selection
-----------------	--

def GetAttrID(self)

Gets the current selection of the attribute ID.

Parameters

return value (string)	Attribute ID of current selection
-----------------------	-----------------------------------

def OnChoose(self, event)

This event is fired when the selection is changed.

7.5.2 wxNetobjtypeCombo Class

This class implements a widget for a combo box that contains all network object types. These support all container methods (e.g. GetMulti/SetMulti) and allow the user to make his/her selection. The class is derived from the wx.ComboBox class.

def __init__(self, *args, **kwargs)

The constructor for the class.

def InitChoices(self, Visum)

Initialize combobox values with all net objects.

Parameters

Visum	Visum COM object
return value	None

def GetName(self)

Gets the name of the current selection.

Parameters

return value (string)	Name of the current selection
-----------------------	-------------------------------

def SetName(self, name)

Sets the current selection to the value of the name specified.

Parameters

name (string)	Name of network object that shall become the new selection
---------------	--

def SetIndex(self, name)

Set the index of the value(name). If an exception occurs return None.

Parameters

name (string)	Name of network object which index shall be set
---------------	---

def GetContainer(self, Visum)

Gets the object of the network object collection for the current selection.

Parameters

return value (Visum network container object)	Object of Visum network object collection for selected network object type
---	--

Example:

If the current selection is "Links", the method will return a Visum.Net.Links object.

7.5.3 wxNetobjCombo Class

This class implements a wx widget for a combo box that contains all instances of a network object type and allows the user to make a selection. Multiple selection is not supported. The class is derived from the wx.ComboBox class.

def __init__(self, *args, **kwargs)

The constructor for the class. Since most popular GUI images, as e.g. wxGlade, do not support any additional arguments that allow you to adapt widgets, these are adapted via the methods described in the following and not directly via the constructor.

def InitChoices(self, container, displayattrs, resultattr=None, filtervector=None, displayformatter=None)

This method is used to specify the attributes that shall be displayed in the selection list and the format in which the selection shall be displayed. This method must be performed before the user can make a selection.

Parameters

container (object)	The network object collection for which attribute values are queried (e.g. Visum.Net.Links).
displayattrs (list(string))	Attribute IDs that refer to the attributes of a network object type. They are displayed on the selection list. These are typically the key attributes of the type. Attribute values are separated by vertical lines.
resultattr (string)	If specified, refers to the ID of an attribute whose value is returned as a selection. Otherwise, the object itself is returned (default: none).
filtervector (list(bool))	If specified, refers to a list with length == number of objects in container - if passed, then only the objects for which the corresponding list item is True are added to the combobox. If filtervector is omitted, all objects are added. (default: none)
displayformatter (function)	if specified, refers to a function which takes as many parameters as there are displayattrs and maps them to a string. In effect, the function is called for each combobox entry with the displayattrs values for that entry. The function should return a human-readable string that will be added as the combobox item. If displayformatter is not passed, the values are all concatenated, separated by vertical bars (default: none).

Example:

Creating a selection list with block objects. The individual list entries appear as "17 | Bus", whereas 17 is the block object ID and "Bus" is the name of the vehicle combination. The selection is returned as block object:

```
cboBlocks.InitChoices(Visum.Net.Blocks, displayattrs=["ID", "VEHCOMBNAME"])
```

Using optional parameter resultattr:

Creating a selection list for all lines. The line name is used as a list entry and also returned as a result of the selection:

```
cboBlocks.InitChoices(Visum.Net.Lines, displayattrs=["NAME"], resultattr="NAME")
```


Using optional parameter displayattrs and displayformatter:

Creating a selection list for all links. Link entries are displayed in the following shape:

```
LinkNo (FromNode -> ToNode)
```

Therefore, we need to add the format function as follows:

```
def linkformatter(no, fromnode, tonode):
    return "%d (%d -> %d)" % (no, fromnode, tonode)
```

Then

```
cboLinks.InitChoices(Visum.Net.Links,
                    displayattrs=["NO", "FROMNODENO", "TONODENO"],
                    displayformatter=linkformatter)
```

will set up a combobox with all Link objects.

Using optional parameter displayattrs and filtervector:

Assume we want to set up a combobox with all Zone objects with TYPENO == 1, then we have to write the following lines:

```
zonetype = numpy.array(VisumPy.helpers.GetMulti(Visum.Net.Zones, "TYPENO"))
cboZones.InitChoices(Visum.Net.Zones,
                    displayattrs=["NO"],
                    filtervector=(zonetype==1))
```

def GetSelectionData(self)

This method returns the selection in the form specified using the resultattr parameter of the InitChoices() method. If instead, you want the index of the entry, you can use the GetSelection() method inherited from wx.ComboBox. To set a selection, apply the SetSelection or SetStringSelection methods inherited from the base class.

def SetSelectionByName(self, entryName)

This method returns the index of a combobox string entry. The index has the form specified by the resultattr parameter of InitChoices().

Parameters

entryName (string)

The combobox entry name which shall be selected.

7.6 Excelplot Module

This module contains a helper class for creating charts in Excel.

7.6.1 Chart Class

The class represents an Excel diagram with one or several data series. Data are added one series at a time and finally show is called to display the diagram.

def __init__(self, title="", xtitle="", ytitle="")

The constructor for the class.

Parameters

title	Diagram title
xtitle	x-axis title
ytitle	y-axis title
return value	The object.

Example:

```
Chart("My diagram", "My x values", "My y values")
```

will create a diagram with these title strings.

def addSeries(self, x, y, legend="")

Adds one data series to the diagram.

Parameters

x	List of x values. If several data series are added, only the x values of the first call are accepted as x axis labels, the others are ignored.
y	List of y values
legend	Title of data series in the legend
return value	The Excel Series object

Example:

```
chart.addSeries([1,2,3], [10,20,30], "tenfold")
```

adds a series with the data points (1, 10), (2,20), (3,30) to the diagram and labels it "tenfold" in the legend.

def show(self)

Makes the diagram visible.

attribute chart

Default values apply to all diagram properties not set by the methods above. In particular the diagram will appear as a column chart. If more control over the appearance is needed, then the underlying Excel chart object may be accessed through the "chart" attribute of the class.

Example: The following statement (immediately before calling show()) will switch the chart type to a surface chart:

```
chart.chart.ChartType = 85 # Surface top-view
```

7.7 Excel Module

This module contains functions for writing listing data to Excel.

def writeTableToExcelFile(listOfColumns, attributeNames, fileName, sheetName=None)

Writes a table to Excel.

Parameters

listOfColumns	List of data columns
attributeNames	List of column names (as strings)
fileName	Name of Excel file generated
sheetName	Name of worksheet that data is written to. Default is ActiveSheet
return value	none.

Example:

Create a listing of turns by node by orientation and then write results to an Excel file (for the function `turnValueByOrientation` see 7.8):

```
turnVols = turnValueByOrientation(Visum, "VolVehPrT(AP)")
writeTableToExcelFile(turnVols[0], turnVols[1], "c:/turnVolume.xls")
```

def writeTablesToExcelFile(listOfTables, fileName)

Writes tables to an Excel sheet.

Parameters

listOfTables	Table data as list with entries in table format (liste(list)), as attributes (list(string)), table sheet (string)
fileName	Name of Excel file generated
return value	none.

Example:

This is an example of writing node, link and zone attributes:

```
attributesNode = ["No", "Name"]
nodeData = networkObjectAttributeList(Visum.Net.Nodes, attributesNode)
nodeTable = [nodeData[0], attributesNode, "Node"]
attributesLink = ["FromNodeNo", "ToNodeNo", "VolVehPrT(AP)"]
linkData = networkObjectAttributeList(Visum.Net.Links, attributesLink)
linkTable = [linkData[0], attributesLink, "Link"]
attributesZone = ["No", "Name", "XCOORD", "YCOORD"]
zoneData = networkObjectAttributeList(Visum.Net.Zones, attributesZone)
zoneTable = [zoneData[0], attributesZone, "Zone"]
listOfTables = [nodeTable, linkTable, zoneTable]
writeTablesToExcelFile(listOfTables, "c:/report.xls")
```

7.8 Reports Module

This module contains functions for creating Visum list-like objects.

def turnValueByOrientation(Visum, attribute)

This function creates a table of turn attributes by orientation (SBL, SBT, SBR, etc) for each active node. The result is for each node a row with the node number and a column for each turn at that node. It is limited to up to four legs. The results of this function can be written to Excel with `writeTableToExcelFile()`.

Parameters

Visum	The Visum object (must be passed explicitly)
attribute	Name of the turn attribute
return value	Column array and columns names list

Example:

```
turnVols = turnValueByOrientation(Visum, "VolVehPrT(AP)")
```

def networkObjectAttributeList(networkObjectCollection, attributesList)

Creates a list of columns of attributes for a network object.

Parameters

networkObjectCollection	Network object container, such as nodes or zones
attributesList	List of attribute names
return value	List of columns and column names

Example:

This example creates a list of the specified territory attributes and writes the results to Excel.

```
attributes = ["Name","VehHourTravPrT(AP)","VehMiTravPrT(AP)", "PassMiTrav(AP)",
             "PassHourTrav(AP)"]
result = networkObjectAttributeList(Visum.Net.Territories, attributes)
writeTableToExcelFile(result[0], result[1], "c:/terrIndicators.xls")
```

def readTurnValueByOrientationFromCSV(Visum, fileName, targetAttribute)

Loads a numeric turn attribute from a csv file that contains attribute values based on node orientation. Works for nodes with up to four legs. The csv file should look as follows, whereas "No" stands for the node number:

```
No,EBU,EBL,EBT,EBR,NBU,NBL,NBT,NBR,WBU,WBL,WBT,WBR,SBU,SBL,SBT,SBR
10200,0,0,503.112,0,0,0,0,0,0,0,616.258,0,0,0,0,0
10202,0,0,462.941,62.537,0,0.958,0,36.502,0,360.703,252.087,0,0,0,0,0
10203,0,183.95,243.2,5.147,0,5.737,61.442,21.8,0,0,240.8,6.6,0,55.067,660.363,0
12329,0,0,0,0,0,152.507,183.032,0,0,182.756,1120.336,296.722,0,0,410.495,16.527
```

Parameters

Visum (COM object)	Visum object
fileName (String)	Name of csv file
targetAttribute (String)	Name of numeric turn attribute as target for data
return value	none

7.9 Matrices Module

This module contains functions for matrix manipulation. See the helpers module for functions to get and set matrices from and to Visum (see 7.2). See 7.1 on details about using either numarray or numpy matrices.

def rowSums(mat)

Returns the row sums of the matrix.

Parameters

mat	Matrix
return value	Vector of row sums.

Example:

```
import numpy
from VisumPy.matrices import *

x = numpy.array([1,2,3,4])
x = x.reshape((2,2))
print x
print rowSums(x)
```

def colSums(mat)

Returns the column sums of the matrix.

Parameters

mat	Matrix
return value	Vector of column sums.

Example:

```
import numpy
from VisumPy.matrices import *

x = numpy.array([1,2,3,4])
```

```
x = x.reshape((2,2))
print x
print colSums(x)
```

def calcIntrazonal(mat, factor=0.5, numNeighbors=1)

Sets the diagonal to the average value of the destination zone closest to numNeighbor multiplied by the "factor" factor. The value for zone "i" is calculated using the smallest entries in row "i" and column "i".

Parameters

mat	Matrix
factor	Scaling factor for value of nearest zone
numNeighbors	Number of neighbors that must be taken into account
return value	Adjusted matrix.

Example:

```
import numpy
from VisumPy.matrices import *

x = numpy.array([1,2,3,4])
x = x.reshape((2,2))
print x
x = calcIntrazonal(x, 0.5)
print x
```

def addVector(mat, vector, byrow=True)

Adds vector by row or by column to matrix.

Parameters

mat	Matrix
vector	Row or column vector
byrow	Replicate by row or by column, default is True = by row
return value	Adjusted matrix.

Example:

```
import numpy
from VisumPy.matrices import *

x = numpy.array([1,2,3,4])
x = x.reshape((2,2))
print x
y = numpy.array([1,2])
z = addVector(x, y)
print z
```

def unique(vector)

Unique items in a vector.

Parameters

vector	Array of items
return value	Array of individual items

Example:

```
import numpy
from VisumPy.matrices import *

print unique(numpy.array([1,2,2,3,4,5,5]))
```

def rmse(arrayOne, arrayTwo)

Root Mean Square Error of two arrays.

Parameters

arrayOne	Array one
arrayTwo	Array two
return value	Mean root square error between the two arrays.

Example:

```
import numpy
from VisumPy.matrices import *

x = numpy.array([1,2,2,3,4,5,5])
y = numpy.array([10,5,3,6,2,5,5])
print rmse(x,y)
```

def prmse(arrayOne, arrayTwo)

Percent Root Mean Square Error of two arrays.

Parameters

arrayOne	Array one
arrayTwo	Array two
return value	Mean root square error between the two arrays.

Example:

```
import numpy
from VisumPy.matrices import *

x = numpy.array([1,2,2,3,4,5,5])
y = numpy.array([10,5,3,6,2,5,5])
print prmse(x,y)
```

def aggregateMatrix(mat, mainZones, function=sum)

Aggregates a matrix by mainZones by the input function.

Parameters

mat	Matrix
mainZones	A main zone number for each matrix row
function	Python function, such as min, max, sum, len, default is sum
return value	Aggregated matrix.

Example:

```
import numpy
from VisumPy.matrices import *

x = numpy.array([1,2,3.,4,6,7,8,9,10])
x = x.reshape((3,3))
print x
y = [1,2,2] #main zone for each zone
print aggregateMatrix(x,y,sum)
print aggregateMatrix(x,y,min)
```

def balanceMatrix(mat, rowTargets, colTargets, iterations=25, closePctDiff=0.0001)

Balances a matrix to row and column totals. This procedure is also known as IPF (iterative proportional fitting) and Furness.

Parameters

mat	Matrix
-----	--------

rowTargets	Vector of row targets
colTargets	Vector of column targets
iterations	number of iterations, default 25
closePctDiff	Deviation in percent in percent for cancel criterion, default 0.0001
return value	(Un)balanced matrix

Example:

```
import numpy
from VisumPy.matrices import *

mat = numpy.array([0,80.,120,200,60,0,100,140,110,40,0,50,80,270,250,0])
mat = mat.reshape((4,4))
print mat
r = numpy.array([500,450,250,800])
c = numpy.array([300,500,600,600])
print balanceMatrix(mat,r,c)
```

def balance3DMatrix(mat, rowTargets, colTargets, depTargets, iterations=25, closePctDiff=0.0001)

Balance a matrix to row and column totals and to totals in a third dimension. This procedure is also known as IPF (iterative proportional fitting) and Furness.

Parameters

mat	Matrix
rowTargets	Vector of row targets
colTargets	Vector of column targets
depTargets	Vector of depth targets (third dimension)
Iterations	Number of iterations, default 25
closePctDiff	Deviation in percent in percent for cancel criterion, default 0.0001
return value	(Un)balanced matrix

Example:

```
import numpy
from VisumPy.matrices import *

mat = numpy.array([0.0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,
                  21,22,23])
mat = mat.reshape((4,3,2))
print mat
r = numpy.array([40,120,180,260])
c = numpy.array([160,200,240])
d = numpy.array([280,320])
print balance3DMatrix(mat,r,c,d,25,0.05)
```

def setDiagonalFromZoneAttribute(Visum, mat, zoneAttr)

Sets the diagonal of a matrix according to a zone attribute.

Parameters

Visum (COM object)	Visum object
mat (numpy.array)	Matrix
zoneAttr (String)	Name of zone attribute
return value (numpy.array)	Changed matrix

Example:

```
import numpy
import VisumPy.helpers
x = setDiagonalFromZoneAttribute(Visum, VisumPy.helpers.GetSkimMatrix(Visum, 1),
"ADDVAL1")
```

```
VisumPy.helpers.SetSkimMatrix(Visum, 1, x)
```

def setZoneAttributeFromDiagonal(Visum, mat, zoneAttr)

Sets a zone attribute using values on the diagonal of a matrix.

Parameters

Visum (COM object)	Visum object
mat (numpy.array)	Matrix
zoneAttr (String)	Name of zone attribute
return value	none

Example:

```
import numpy
import VisumPy.helpers
x = VisumPy.helpers.GetODMatrix(Visum, 1)
setZoneAttributeFromDiagonal(Visum, x, "ADDVAL1")
```

def balanceMatrixFromZoneAttributes(Visum, matNum, rowAttName, colAttName, iterations=25, closePctDiff=0.0001)

Balances a matrix according to zone attributes for row and column sums. Wrapper function for the balanceMatrix function. Since this function changes the matrix in Visum, first make a copy of it.

Parameters

Visum (COM object)	Visum object
matNum (int)	Number of OD-matrix in Visum
rowAttName (String)	Name of zone attribute for use as row sum
colAttName (String)	Name of zone attribute for use as column sum
iterations (int)	Number of iterations, default 25
closePctDiff (float)	Difference in percent used as a cancel criterion, default 0.0001
return value	none

Example:

```
balanceMatrixFromZoneAttributes(Visum, 1, "ADDVAL1", "ADDVAL2")
```

def readBIMatrixWithHeader(fileName)

Reads a binary packed matrix including titles.

Parameters

fileName (String)	File name of matrix
return value (dict)	dictionary with matrix data and metadata

def writeBIMatrixWithHeader(input, fileName)

Writes a binary packed matrix including titles.

Parameters

Input (list)	Matrix data and titles
fileName (String)	File name of matrix
return value	none

def toNumArray(matList)

Internal function for converting a BI list into a numpy/numarray 2D-array object.

Parameters

matList (list)	Matrix data and titles
return value (numpy.ndarray)	Matrix

def fromNumArray(nArray, zoneNums)

Internal function for converting a numpy/numarray 2D-array object into a BI list.

Parameters

nArray (2D-array)	Matrix
zoneNums (list)	List of zone numbers
return value (list)	Matrix data and titles

def readBIMatrix(fileName)

Reads a matrix that is binary packed in a numpy/numarray 2D-array.

Parameters

fileName (String)	File name of matrix
return value (numpy.ndarray)	Matrix

Example:

```
x = readBIMatrix("c:/test.mtx")
writeBIMatrix(x, [1,2,3], "c:/testout.mtx")
```

def writeBIMatrix(nArray, zoneNums, fileName)

Creates a binary matrix using a numpy matrix.

Parameters

nArray (numpy.matrix)	Matrix
zoneNums (list)	List of zone numbers
fileName (String)	File name of matrix
return value	none

Example:

```
x = readBIMatrix("c:/test.mtx")
writeBIMatrix(x, [1,2,3], "c:/testout.mtx")
```

7.10 csvHelpers Module

This module provides functions for getting and setting comma separated text files (csv). The separator can be changed by the developer. This is useful for treating Visum net files that use the semicolon (";") as separator.

def readCSV(fileName, numarrayRec=False, delimiter=";")

Reads a comma separated text file (csv file) into an array of arrays, each corresponding to one row. The separator can be specified to enable reading of Visum net files (separator ";"). Usually the first row of a csv file contains the column names but this is not a condition. All rows are interpreted as strings. Thus it may be necessary to cast the values to the desired type after using this function.

If the optional parameter numarrayRec is 'true', the function returns a numpy/numarray.records object (like a spreadsheet). In this case the first row must contain the column names. See 7.1 on details about using either numarray or numpy records.

Parameters:

filename	Name of csv file
numarrayRec	if "True", a numarray/numpy.records object is returned. Default is false
delimiter	Separator. Default is ",", use ";" for Visum net files
return value	List of row lists or numarray.records object.

Example:

```
inFile = readCSV("c:/ids.csv")
inFile = readCSV("c:/nodes.net", ";")
```

def writeCSV(rowList, fileName, delimiter=",")

Writes a list of row lists to a comma separated text file. The separator can be specified to enable writing of Visum net files (separator ";"). Usually the first row of a csv file contains the column names but this is not a condition.

Parameters

rowList	List of table row lists
filename	Name of csv file
delimiter	Separator. Default is ",", use ";" for Visum net files
return value	Null.

Example:

```
inFile = readCSV("c:/ids.csv")
writeCSV(inFile, "c:/idsOut.csv")
```

7.11 AddIn Module

This module exists from Visum version 12.0 and contains methods and functionality used for Visum add-ins. For information about add-in components please see 6.9.

7.11.1 AddIn Class

This class contains functionality for treating the add-in in debug mode, handling the internationalization settings, handling an optional progress dialog and implements logging functionality, i.e. writing into Visum trace/error file and/or displaying a message box.

The following public members are provided:

member	explanation
self.State	Returns the state of the add-in, if it was well defined and no error occurred.
self.LanguageCode	Language code of the current Visum language.
self.Language	Current Visum language.
self.Name	The name of the add-in.
self.ErrorObjects	A list containing all current add-in errors.
self.Visum	The Visum pointer.
self.InstallTranslation	If true, set internationalization according to Visum language.
self.IsInDebugMode	Is set in the add-in constructor and can be used to differ between debug and release mode. Defines if the add-in is in debug mode or

	not.
self.TemplateText.MainApplicationError	Provides internationalized standard error text "An Error in the main application of the AddIn occurred: ".

def __init__(self, Visum = None, installTranslation = True, name = "")

The constructor of the AddIn class.

Parameters

Visum [optional, default = None]	The Visum pointer, parameter is needed when script is in release mode.
installTranslation [optional, default = true]	If true, set internationalization according to Visum language.
name [optional, default = ""]	The add-in name.
return value	None.

Example to create a new add-in:

```
from VisumPy.AddIn import AddIn
```

Calling AddIn constructor in release mode: `addIn = AddIn(Visum)`

Calling AddIn constructor in debug mode: `addIn = AddIn()`

def __del__(self)

The destructor of the AddIn class.

def __IsInDebugMode(self)

Checks if the AddIn is in debug mode. This is done via command line arguments which are stored in sys.argv.

In the debug mode the following command line arguments are necessary:

1. 'Visum.Visum' (or any other Visum installation e.g. 'Visum.Visum.120')
2. Full file path to a Visum version file *.ver (this parameter is optional)

def ReportMessage(self, message, messageType = 1, captionText = None)

Writes an error message into Visum log or trace file if the add-in is not executed from the script menu otherwise a message box is shown.

Parameters

message	The message which shall be displayed.
messageType [optional, default = Error]	The type of the messages are, see table below.
captionText [optional, default = None]	The caption text of the message box.
return value	None.

identifier	value	comment
MessageType.Warning	0	Add-in executed as procedure: Writes the message to the Visum trace file. Add-in executed from script menu: Display message box with warning icon.
MessageType.Error	1	Add-in executed as procedure: Writes the message to the Visum error file. Add-in executed from script menu: Display message box with error icon.
MessageType.Info	2	Add-in executed as procedure: Writes the message to the Visum trace file. Add-in executed from script menu: Display message box with information icon.

Table 2: Instances of the parameter messageType**def GetAddInLanguage(self)**

Return internationalization language of the add-in.

def HandleException(self, prefixText = "")

This method is called if an exception during the add-in execution occurs. The error text which is retrieved from the error stack is either written to the Visum error file or displayed by a message box depending if the add-in is executed from script menu or as procedure. Optionally a prefix text which is put before the error text can be set as parameter.

Parameters

prefixText [optional, default = ""]	A text which is put before the error text.
return value	None.

def WriteToTraceFile(self, message)

Write a message into the Visum trace file.

Parameters

message	The message which shall be written.
return value	None.

def WriteToErrorFile(self, message)

Write a message into the Visum error file.

Parameters

message	The message which shall be written.
return value	None.

def ShowProgressDialog(self, captionText, infoText, maxCounter, setTimeMode = False)

Show a progress dialog and set progress bar settings.

Parameters

captionText	The caption text of the dialog.
infoText	The information text of the dialog.
maxCounter	The caption text of the dialog.
setTimeMode [optional, default = False]	If true, elapsed and remaining time is displayed additionally.
	If false, only caption and information text is displayed.
return value	None.

Example:

The following progress dialog of the add-in '*PuT > Buffers around PuT lines*' shows the elapsed and remaining time and is shown until value 100 unless the Cancel button was pressed.

```
addIn.ShowProgressDialog(_("Calculating buffers"), _("Time remaining"), 100,
setTimeMode = True)
```

def UpdateProgressDialog(self, counter, messageText = None)

This method is used to update a defined progress dialog, see method *ShowProgressDialog()*.

Parameters

messageText [optional, default = None]	Message text which shall be displayed.
return value	None.

Example:

Set the progress bar to value 50 and display the new text, see the add-in '*Line Blocking > Vehicle Usage Profile*':

```
addIn.UpdateProgressDialog(50, _("Creating chart"))
```

def CloseProgressDialog(self)

This method is used to close the progress dialog if it was defined with the method *ShowProgressDialog()*.

Example:

Here you see the usage of a progress dialog in the add-in '*GIS > Buffers around PuT Lines*':

```
def StartBuffering(self):
    Visum.Graphic.StopDrawing = True
    self.CreatePOICat()
    self.scale = Visum.Net.AttValue("SCALE")

    if self.param["ObjType"] == _("Lines"):
        allLines = Visum.Net.Lines.GetAllActive
        self.numobj = len(allLines)
        addIn.ShowProgressDialog(_("Calculating buffers"), _("Time
remaining"), self.numobj, setTimeMode = True)
        try:
            self.BufferLines(allLines)
        except:
            addIn.HandleException()
    else:
        self.numobj = Visum.Net.LineRoutes.CountActive
        addIn.ShowProgressDialog(_("Calculating buffers"), _("Time
remaining"), self.numobj, setTimeMode = True)
        try:
            self.BufferLineRoutes()
        except:
            addIn.HandleException()

    # redraw graphic
    addIn.CloseProgressDialog()
    Visum.Graphic.StopDrawing = False
    Visum.Graphic.Redraw()
```

7.11.2 AddInParameter Class

This class handles Visum parameters of the `Parameter.Data` object. It offers functionality of checking parameters and defining dependencies of parameters.

def __init__(self, addIn, visumParameter)

The constructor of the `AddInParameter` class.

Parameters

<code>addIn</code>	The <code>AddIn</code> object which must be created in each add-in script.
<code>visumParameter</code>	The Visum <i>Parameter</i> container object which stores all add-in parameters (only available in release mode, else pass <code>None</code>).
return value	<code>None</code> .

Example to create a new add-in parameter object:

```
from VisumPy.AddIn import AddInParameter, AddIn
addIn = AddIn(Visum)
```

Calling constructor in release mode: `addInParam = AddInParameter(addIn, None)`

Calling constructor in debug mode: `addInParam = AddInParameter(addIn, Parameter)`

def AddDependency(self, originAttribute, dependentAttribute, checkValue)

This method defines if a Visum parameter is optional or if there are dependencies between parameters which will be checked in the `AddInParameter.Check()` method.

Therefore, the method checks if the *originAttribute* is not valid, i.e. `None` or an empty string. If so, the dependent attribute must have the value which is defined in *checkValue*.

Parameters

<code>originAttribute</code>	The name of the parameter which is tested to be valid.
<code>dependentAttribute</code>	The name of the parameter which is dependent on the <i>originAttribute</i> .
<code>checkValue</code>	If the <i>originAttribute</i> is invalid the <i>dependentAttribute</i> must have this <i>checkValue</i> .
return value	<code>None</code> .

Examples:

1. Add-in Matrix Convolution:

The add-in contains a filter checkbox and a corresponding filter attribute which shall be set when the filter checkbox is checked, see Figure 2: Matrix Convolution add-in dialog with filter and filter attribute in rectangular.

Therefore, we have to define a dependency between the filter attribute "*FilterAttr*" parameter and the filter checkbox "*Filter*" parameter. In this case, we would define that if the filter attribute is not set in the Visum parameters the dependent parameter "*Filter*" (checkbox) must not be checked, i.e. the "*Filter*" attribute must have the value `False`. The following lines were added to the dialog script:

```
...
addInParam.AddDependency("FilterAttr", "Filter", False)
```

```
param = addInParameter.Check(True, defaultParam)
```

```
...
```

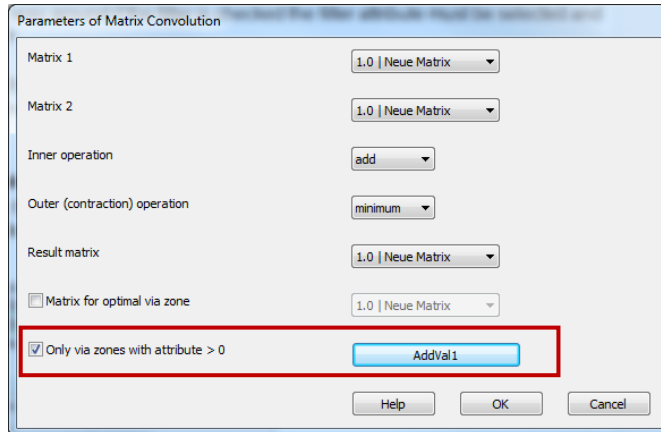


Figure 2: Matrix Convolution add-in dialog with filter and filter attribute in rectangular

2. Add-in Calculate Matrix:

In order to define that the marked text boxes **1** and **2** are optional in the add-in dialog (see Figure 3: Calculate Matrix add-in dialog) we define two dependencies for each text box. In this case, the *originAttribute* and the *dependentAttribute* are the same. The *checkValue* is an empty string. This in turn means if the *originAttribute* (the text box attribute) is empty its dependent attribute (which is the same) must be empty. This means the text box may be empty and is an optional parameter.

The following lines were added to the dialog script:

```
...
```

```
addInParameter = AddInParameter(addIn, visumParameter)
```

```
addInParameter.AddDependency("Filter", "Filter", '')
```

If the filter parameter is empty the dependent attribute Filter must be empty.

```
addInParameter.AddDependency("Context", "Context", '')
```

```
param = addInParameter.Check(True, defaultParam)
```

```
...
```

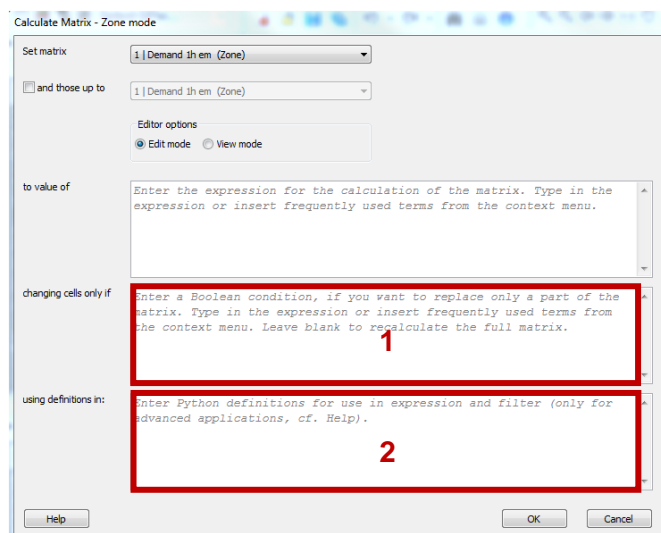


Figure 3: Calculate Matrix add-in dialog

def SaveParameter(self, param)

This method stores the addIn parameter object into the Visum Parameter.Data container. It sets `Parameter.OK = True`. Usually, it is called in the dialog script in the OK button event handler, see 6.9.

Parameters

param	The add-in parameter container.
return value	None.

def Check(self, isBody, defaultParameter = None)

This method checks if the add-in parameter are valid. Usually, it is called both in the dialog and body script, see 6.9. If the parameters are incorrect an exception is thrown.

Parameters

isBody	Boolean if the method call was from body script.
defaultParameter [optional, default = None]	Optional, the dictionary of default parameters.
return value	The parameter dictionary.

8 Matrix Editor Interface Script Muuli

With the new Matrix Editor in VISUM there is an alternative way to use Script Muuli beside the Matrix Editor Interface Script (see Example 8.9.1). The matrix editor object can be derived from an instantiated IVisum object using the MatrixEditor access method. In this case an IMatrixEditor object is created which is equivalent to the COM object generated by CreateObject (see Example 8.9.2).

The advantage of using the IMatrixEditor access method is that an instance of the matrix editor version is created that always corresponds to the currently running VISUM version in the case that several versions are installed. Please note that each call to the IVisum.MatrixEditor method returns a reference to the same matrix editor object. Therefore, it is possible to write down several matrix operations like Visum.MatrixEditor.MLoad..., but it is not possible to open several distinct matrix editors to be used for distinct operations at the same time. To do so, the CreateObject method has to be used.

Anyhow for completeness this chapter describes the old interface:

The COM interface ScriptMuuli is available after installation of VISUM if the module Muuli+ is licensed. An additional installation is not necessary. Please be sure that the component is switched on in the 'Licence' dialog of the 'Extra' menu. The basic use of the interface is the same as that of the VISUM COM interface, but a separate COM object has to be created using the CreateObject command. The ScriptMuuli instance can be called for each installed VISUM instance.

Important Notice: The calling convention for ScriptMuuli has changed to enable users using several versions of ScriptMuuli to differentiate between these versions. The hitherto existing call *CreateObject("VISION.ScriptMuuli.1")* is now only valid for ScriptMuuli (32bit) shipped with VISEM/Muuli 8.1. The ScriptMuuli versions shipped with VISUM 12.0 are called as follows, see example 8.9.1:

- *CreateObject("VISUM120.ScriptMuuli")* creates an instance of the ScriptMuuli version shipped with VISUM 12.0 (32bit),
- *CreateObject("VISUM120-64.ScriptMuuli")* creates an instance of the ScriptMuuli version shipped with VISUM 12.0 (64bit).

While using Excel (32bit) and VBA please install VISUM 12.0 (32bit) with 32bit ScriptMuuli-Interface. If you have got installed Excel (64bit) then please install VISUM 12.0 (64bit) containing 64bit ScriptMuuli-Interface. In general it is not possible to use a 32bit application (such as Excel) with a 64bit interface.

8.1 Hints for the use of the matrix editor COM interface

- Specify absolute paths only, i.e. no relative paths.
- Indices: In MuuliScript all indices start with 1.
- Precision can be specified through the format string ranging from 0 (for integers) to 6.

Example:

\$V saves the matrix in the \$V format with integers,

\$V;D2 saves the matrix in the \$V format with 2 decimal places.

8.2 General Muuli Methods

GetVersionNumber ([out, retval] VARIANT *VersionNumber)

Returns the version number.

GetInfoString ([out, retval] BSTR *VersionNumber)

Returns an information about the program.

8.3 Muuli application Modes

For access to the matrix editor Muuli via COM two different application modes are provided.

Mode 1

To call the matrix operation, specify input matrix, output matrix and matrix format of the output matrix and also code file, if applicable.

For each method call

- input matrices are loaded into the computer memory,
- the corresponding calculation is performed and
- the output matrices are saved to file.

Afterwards matrices are not longer saved to the main storage of the computer.

Example:

```
muuli.Add ("Matrix1.fma", "Matrix2.fma", "OutputMat1.fma", "$V;D2")
muuli.AddValue ("OutputMat1.fma", 3.0, "OutputMat1.fma", "$V;D2")
muuli.Aggregate ("OutputMat1.fma", "agg.cod", "OutputMat1.fma", "$V;D2"))
```

Mode 2

The second mode is characterized by

- loading a matrix into the main storage as a separate operation,
- applying one or more operations to the saved matrix,
- saving the matrix as separate operation.

Example:

```
muuli.MLoad ("Matrix1.fma")      'Loads matrix 1 into main storage
muuli.MAdd ("Matrix2.fma")      'Adds Matrix1 and Matrix2
muuli.MAddValue (3.0)           'Adds Value 3 to the result
muuli.MAggregate ("agg.cod")    'Aggregates
muuli.MSave ("OutputMat.fma", "$V;D2") 'Saves results matrix
```

8.4 Loading a Matrix from/Saving a Matrix to the Main Storage

MLoad ([in] VARIANT Mat)

Loads the matrix into computer memory. The matrix remains in the computer memory until a new matrix is loaded into the computer memory or until the script is finished.

MSave ([in] VARIANT Mat, [in] BSTR FormatString)

Saves the matrix stored in the computer memory to hard disk, with the specified file name and the matrix format specified. The matrix remains in the computer memory and is available for further calculations.

MSaveOptions ([in] VARIANT Mat, [in] BSTR FormatString, [in] BSTR Time)

Saves the matrix stored in the computer memory to hard disk, with the specified file name, matrix format and time interval.

The time interval is specified as a string according to the following rules: from time; to time. If minutes are specified their number is separated using a point in the time strings as well as in the matrix. Examples: "6;7" for 6:00h to 7:00h, "6.30;7.30" for 6:30h to 7:30h.

8.5 Query/Set Properties and Values of Loaded Matrix

MGetColCount ([out, retval] long *colCount)

Determines the number of columns of a matrix that has been loaded into the computer memory via MLoad.

MGetRowCount ([out, retval] long *rowCount)

Determines the number of rows of a matrix that has been loaded into the computer memory via MLoad.

MGetTotalSum ([out, retval] VARIANT *Sum)

Determines the total sum of a matrix that has been loaded into the computer memory via MLoad.

MGetOriginSum ([in] VARIANT FromZoneNo, [out, retval] VARIANT *Sum)

Determines the row sum (origin sum) for a zone of a matrix that has been loaded into the computer memory via MLoad. As parameter the zone number (external ID) has to be specified.

MGetDestinationSum ([in] VARIANT ToZoneNo, [out, retval] VARIANT *Sum)

Determines the column sum (destination sum) for a zone. As parameter the zone number (external ID) has to be specified.

MGetOriginSumByIndex ([in] VARIANT fromIndex, [out, retval] VARIANT *Sum)

Determines the row sum (origin sum) of the i-th row in a matrix. As parameter the index of the row has to be specified. The first row has index 1.

MGetDestinationSumByIndex ([in] VARIANT toIndex, [out, retval] VARIANT *Sum)

Determines the column sum (destination sum) of the i-th column in a matrix. As parameter the index of the column has to be specified. The first column has index 1.

**MGetValue ([in] VARIANT FromZoneNo,
[in] VARIANT ToZoneNo,
[out, retval] double *pVal)**

Determines the value of the element which is referenced by Row-ZoneNo, Column-ZoneNo in the matrix using the Zone IDs, i.e. the external ZoneNumbers.

**MPutValue ([in] VARIANT FromZoneNo,
[in] VARIANT ToZoneNo,
[in] double newVal)**

Sets the value of the element which is referenced by Row-ZoneNo, Column-ZoneNo in the matrix using the Zone IDs, i.e. the external ZoneNumbers.

**MGetValueByIndex ([in] VARIANT fromIndex,
[in] VARIANT toIndex,
[out, retval] double *pVal)**

Returns the value of the element which is determined by Row-IndexNo, Column-IndexNo in the matrix which is stored in the memory. As parameters the indices of the row and column have to be specified. The first row / column has index 1.

**MPutValueByIndex ([in] VARIANT fromIndex,
[in] VARIANT toIndex,
[in] double newVal)**

Sets the value of the element which is determined by Row-IndexNo, Column-IndexNo in the matrix which is stored in the memory. As parameters the indices of the row and column have to be specified. The first row / column has index 1.

MGetRowZoneNr ([in] long index, [out, retval] BSTR *zoneNr)

Returns the name of the zone.

MGetColZoneNr ([in]long index, [out, retval] BSTR *zoneNr)

Returns the name of the zone.

MGetVarType ([out, retval] VARIANT *Type)

Returns the matrix data type. Important: MGetVarType is only provided with the Muuli matrix editor in VISUM.

Feasible return values are:

- 2 represents matrix type "short",
- 3 represents matrix type "int",
- 4 represents matrix type "float",
- 5 represents matrix type "double",

MChangeVarType ([in] VARIANT Type, [in] VARIANT RoundingProcedure)

Changes the variable type of matrix values and the rounding procedure for calculations, if applicable.

As input parameters are required:

Type: An integer in the range [2, 5] with

- 2 represents matrix type "short",
- 3 represents matrix type "int",
- 4 represents matrix type "float",
- 5 represents matrix type "double" (default value)

respectively.

Rounding procedure: either 0 or 1, i.e.

- 0 for random rounding and
- 1 for arithmetic rounding (default value).

Important: MChangeVarType is only provided with the Muuli matrix editor in VISUM.

For type 4 („float“) and 5 („double“) rounding is not applicable. In this case the corresponding parameter is ignored. If values exceeding the range [2, 5] for type and/or [0, 1] for rounding method are specified, the default values will be used.

8.6 Executing Operations on a Loaded Matrix

MAdd ([in] VARIANT Mat)

Adds *Mat* to a matrix that has been loaded to the computer memory using MLoad. Values are added by element. Different matrix dimensions permitted.

Example:

```
retVal = Muuli.MAdd ("Matrix2.fma")
```

or

```
Muuli.MAdd "Matrix2.fma"
```

MSubtract ([in] VARIANT Mat)

Subtracts *Mat* from a matrix that has been loaded to the computer memory using MLoad. Values are subtracted by element. Different matrix dimensions permitted.

MMultiply ([in] VARIANT Mat)

Multiplies *Mat* by a matrix that has been loaded to the computer memory using MLoad. Values are multiplied by element. The matrices must have the same dimensions (number of zones and zone numbers).

MDivide ([in] VARIANT Mat)

Divides a matrix that has been loaded to the computer memory using MLoad by the matrix *Mat*. Values are divided by element. The matrices must have the same dimensions (number of zones and zone numbers).

MMax ([in] VARIANT Mat)

Compares a matrix that has been loaded to the computer memory using MLoad with *Mat*. The result is the maximum of the two matrix element values. The matrices must have the same dimensions (number of zones and zone numbers).

MMin ([in] VARIANT Mat)

Compares a matrix that has been loaded to the computer memory using MLoad with *Mat*. The result is the minimum of the two matrix element values. The matrices must have the same dimensions (number of zones and zone numbers).

MAddValue ([in] VARIANT Value)

Adds the specified value to all elements of the matrix that have been loaded to memory using MLoad.

MSubtractValue ([in] VARIANT Value)

Subtracts the specified value from all elements of the matrix that has been loaded to memory using MLoad.

MMultiplyValue ([in] VARIANT Value)

Multiplies all elements of the matrix that has been loaded to memory using MLoad by the specified value.

MDivideValue ([in] VARIANT Value)

Divides all elements of the matrix that has been loaded into memory before using MLoad by the specified value.

MMaxValue ([in] VARIANT Value)

Compares all elements of the matrix that has been loaded to memory using MLoad with the specified value. The resulting value of a matrix element in row i and column j is

$$\text{Matrix}[i,,j] = \text{MAX}(\text{Matrix}[i,,j], \text{Value}).$$
MMinValue ([in] VARIANT Value)

Compares all elements of the matrix that have been loaded to memory using MLoad with the specified value. The resulting value of a matrix element in row i and column j is

$$\text{Matrix}[i,,j] = \text{MIN}(\text{Matrix}[i,,j], \text{Value}).$$
MSetConstantValue ([in] VARIANT Value)

Sets all elements of the matrix that has been loaded to memory using MLoad to the specified value.

MReciprocalValue ()

Sets all elements of the matrix that has been loaded to memory using MLoad to their reciprocal value.

MSetDiagonal ([in] VARIANT Value)

Sets all elements of the (main) diagonal of the matrix that has been loaded to the computer memory using MLoad to the specified value. Other matrix elements remain unchanged.

MExtractDiagonal ()

Sets the elements of the matrix that has been loaded to the computer memory using MLoad to zero, except values of the (main) diagonal. These matrix elements remain unchanged.

MLn ()

Applies logarithm to all elements in the matrix that has been loaded to the memory using MLoad (natural logarithm).

MExp ()

Applies exponential function to all elements in the matrix that has been loaded to the computer memory using MLoad.

MPow ([in] VARIANT Value)

Raises each element in the matrix that has been loaded to the memory using MLoad to the power of the specified value.

MMirrorUpperTriangle ()

The part of the matrix which is situated above the diagonal (the upper triangle) is mirrored to the lower triangle. This function is applied to the matrix that has been loaded into the computer memory using MLoad before.

MMirrorLowerTriangle ()

The part of the matrix which is situated below the diagonal (the lower triangle) is mirrored to the upper triangle. This function is applied to the matrix that has been loaded into the computer memory using MLoad before.

MTranspose ()

This function is applied to the matrix that has been loaded into the computer memory using MLoad before. Columns become rows and vice versa.

MAggregate ([in] BSTR CodeFileName)

This function is applied to the matrix that has been loaded into the computer memory using MLoad before. Aggregation forms one or several zones from multiple zones of the matrix. Zones can be renamed and / or renumbered or removed from the matrix. A code file (ASCII format) is required for aggregation.

MSplit ([in] BSTR CodeFileName)

This function is applied to the matrix that has been loaded into the computer memory using MLoad before. Using the split function zones can be subdivided into smaller units. A code file (ASCII format) is required for splitting.

MGravitate ([in] BSTR CodeFileName)

This function calculates the gravitation model on the matrix that has been loaded using MLoad. A code file (ASCII format) is required for calculating the gravitation model.

MProject ([in] BSTR CodeFileName)

This function is applied to the matrix that has been loaded into the computer memory and calculates a projection. A code file (ASCII format) is required for calculating a projection model.

MReplaceValue ([in] VARIANT PreviousValue, [in] VARIANT NewValue)

In the matrix loaded using ML load, all elements with the value *PreviousValue* are replaced by elements with the *NewValue*.

**MFilterAndSet ([in] VARIANT LoLimit,
[in] VARIANT LoIncl,
[in] VARIANT UpLimit,
[in] VARIANT UpIncl,
[in] VARIANT NewValueWithin,
[in] VARIANT NewValueOutside)**

This function is applied to the matrix that has been loaded into the computer memory using MLoad before. The elements within the interval are set to *NewValueWithin*; the elements outside the interval are set to *NewValueOutside*.

If the value "" is specified for *NewValueWithin* or *NewValueOutside*, the elements within and/or outside the interval will remain unchanged.

The values for *LoIncl* and *UpIncl* can be specified as follows:

- Value shall belong to interval: 1 or "X" or "x"
- Value shall not belong to interval: 0 or ""

Parameters

[in] VARIANT LoLimit	Lower limit of interval
[in] VARIANT LoIncl	Lower limit belongs to interval yes/no
[in] VARIANT UpLimit	Upper limit of interval
[in] VARIANT UpIncl	Upper limit belongs to interval yes/no
[in] VARIANT NewValueWithin	Value for element within the interval
[in] VARIANT NewValueOutside	Value for element outside the interval

Examples:

```
retVal = Muuli.MFilterAndSet (1, 1, 100, 1, 2, 0)
```

(values in the interval from 1(incl.) to 100(incl.) are set to 2, all others to 0)

```
retVal = Muuli.MFilterAndSet (1, 0, 100, 1, 2, 0)
```

(values in the interval from 1(excl.) to 100(incl.) are set to 2, all others to 0)

```
retVal = Muuli.MFilterAndSet (1, "", 100, "X", 2, 0)
```

(values in the interval from 1(excl.) to 100(incl.) are set to 2, all others to 0)

```
retVal = Muuli.MFilterAndSet (1, "", 100, "X", 2, "")
```

(values in the interval from 1(excl.) to 100(incl.) are set to 2, all others remain unchanged)

```
retVal = Muuli.MFilterAndSet (1, "", 100, "X", "", "0")
```

(values in the interval from 1(excl.) to 100(incl.) remain unchanged, all others are set to 0)

MAdd_aXb ([in] VARIANT Mat, [in] VARIANT Factor, [in] VARIANT Exponent)

Adds an extended utility term ($U = Factor * Mat[i,j] ^ Exponent$) to a previously loaded matrix. Addition is executed for each element. Different matrix dimensions permitted.

MAdd_BoxCox ([in] VARIANT Mat, [in] VARIANT Factor, [in] VARIANT Exponent)

Adds an extended utility term:

for $Exponent > 0$: $BC = Factor * (Mat[i,j] ^ Exponent - 1) / Exponent$

for $Exponent = 0$: $BC = Ln (Mat[i,j])$

Addition is executed for each element. Different matrix dimensions permitted.

**MClassifyUsingIntervals ([in] VARIANT LoLimit,
[in] VARIANT UpLimit,
[in] VARIANT Width,
[in] BSTR ResultFileName)**

Counts the number of relations for each interval. The result is stored into the result file. The intervals are defined as follows:

```
MIN; LoLimit
LoLimit; LoLimit + Width
...
UpLimit - Width; UpLimit
UpLimit; MAX
```

All intervals are of the same length. The lower limit is always included in the interval, whereas the upper limit is excluded. The calculated upper limit will be ignored, if the specified upper limit differs from the calculated one.

The result file has the following structure:

```
$VISION
$VERSION: VersNr;FileType;Language
1.0;MUULI;D
$Interval:From;To;Number;Share;Accum.Number;Accum.Share
MIN;1;0.00;0.0000;0.00;0.0000
1;501;5.00;0.2000;5.00;0.2000
501;1001;10.00;0.4000;15.00;0.6000
1001;1501;0.00;0.0000;15.00;0.6000
1501;2001;5.00;0.2000;20.00;0.8000
2001;2500;0.00;0.0000;20.00;0.8000
2500;MAX;5.00;0.2000;25.00;1.0000
```

MClassifyUsingCodeFile ([in] BSTR CodeFile, [in] BSTR ResultFileName)

Counts the number of relations for each interval. The result is stored into the result file. The definition of the intervals is contained in a code file and can differ from one interval to another. The lower limit is always included in the interval, whereas the upper limit is excluded.

Example of an interval code file:


```
$Interval:From;To;
MIN;1;
1;501;
501;2001;
2001;2501;
2501;MAX;
```

The result file has the following structure:

```
$VISION
$VERSION: VersNr;FileType;Language
1.0;MUULI;D
$Interval:From;To;Number;Share;Accum.Number;Accum.Share
MIN;1;0.00;0.0000;0.00;0.0000
1;501;5.00;0.2000;5.00;0.2000
501;2001;15.00;0.6000;20.00;0.8000
2001;2501;0.00;0.0000;20.00;0.8000
2501;MAX;5.00;0.2000;25.00;1.0000
```

**MClassifyWithMatrixUsingIntervals ([in] VARIANT ClassMat,
[in] VARIANT LoLimit,
[in] VARIANT UpLimit,
[in] VARIANT Width,
[in] BSTR ResultFileName)**

The classification matrix *ClassMat* and the previously loaded matrix must have the same dimensions (number of zones and zone numbers). For each element of the classification matrix (ClassMat) the interval is determined. For each interval the belonging values of the matrix are summed up. The result is stored into the result file.

The intervals are defined as follows:

```
MIN; LoLimit
LoLimit; LoLimit + Width
...
UpLimit - Width; UpLimit
UpLimit; MAX
```

All intervals are of the same length. The lower limit is always included in the interval, whereas the upper limit is excluded. The calculated upper limit will be ignored, if the specified upper limit differs from the calculated one.

The result file has the following structure:

```
$VISION
$VERSION: VersNr;FileType;Language
1.0;MUULI;D
$Interval:From;To;Number;Share;Accum.Number;Accum.Share
MIN;1;230.00;0.1620;230.00;0.1620
1;501;0.00;0.0000;230.00;0.1620
501;1001;460.00;0.3239;690.00;0.4859
1001;1501;250.00;0.1761;940.00;0.6620
1501;2001;90.00;0.0634;1030.00;0.7254
2001;2500;150.00;0.1056;1180.00;0.8310
2500;MAX;240.00;0.1690;1420.00;1.0000
```

**MClassifyWithMatrixUsingCodeFile ([in] VARIANT ClassMat,
[in] BSTR CodeFileName,
[in] BSTR ResultFileName)**

The classification matrix *ClassMat* and the previously loaded matrix must have the same dimensions (number of zones and zone numbers). For each element of the classification matrix (ClassMat) the interval is determined. For each interval the belonging values of the matrix are summed up. The result is stored into the result file.

The definition of the intervals is contained in a code file. The width of the intervals are set individually. The lower limit is always included in the interval, whereas the upper limit is excluded.

Example of an interval code file:

```
$Interval:From;To;
MIN;1;
```

```
1;501;
501;1001;
1001;1501;
1501;2001;
2001;2501;
2501;MAX;
```

The result file has the following structure:

```
$VISION
$VERSION: VersNr;FileType;Language
1.0;MUULI;D
$Interval:From;To;Number;Share;Accum.Number;Accum.Share
MIN;1;0.00;0.0000;0.00;0.0000
1;501;5.00;0.2000;5.00;0.2000
501;1001;10.00;0.4000;15.00;0.6000
1001;1501;0.00;0.0000;15.00;0.6000
1501;2001;5.00;0.2000;20.00;0.8000
2001;2501;0.00;0.0000;20.00;0.8000
2501;MAX;5.00;0.2000;25.00;1.0000
```

MClassifyWithMatrixUsingIntervals2 ([in] VARIANT ClassMat,
 [in] VARIANT LoLimit,
 [in] VARIANT UpLimit,
 [in] VARIANT Width,
 [in] BSTR ResultFileName,
 [in] BSTR ResultMatrixFileName)

The classification matrix *ClassMat* and the previously loaded matrix must have the same dimensions (number of zones and zone numbers). The class matrix is created in the same dimensions as *ClassMat* and saved in format "\$V" to a file. Each element of the class matrix contains the number of the interval to which the corresponding element of the previously loaded matrix belongs. The numbering of intervals starts with 1. Intervals are numbered with integers; number of decimal places not indicated.

The matrix format of the class matrix cannot be changed.

The intervals are defined as follows:

```
MIN; LoLimit
LoLimit; LoLimit + Width
...
UpLimit - Width; UpLimit
UpLimit; MAX
```

All intervals are of the same length. The lower limit is always included in the interval, whereas the upper limit is excluded. The calculated upper limit will be ignored, if the specified upper limit differs from the calculated one.

The result file has the following structure:

```
$VISION
$VERSION: VersNr;FileType;Language
1.0;MUULI;D
$Interval:From;To;Number;Share;Accum.Number;Accum.Share
MIN;1;230.00;0.1620;230.00;0.1620
1;501;0.00;0.0000;230.00;0.1620
501;1001;460.00;0.3239;690.00;0.4859
1001;1501;250.00;0.1761;940.00;0.6620
1501;2001;90.00;0.0634;1030.00;0.7254
2001;2500;150.00;0.1056;1180.00;0.8310
2500;MAX;240.00;0.1690;1420.00;1.0000
```

MClassifyWithMatrixUsingCodeFile2 ([in] VARIANT ClassMat,
 [in] BSTR CodeFileName,
 [in] BSTR ResultFileName,
 [in] BSTR ResultMatrixFileName)

The classification matrix *ClassMat* and the previously loaded matrix must have the same dimensions (number of zones and zone numbers). The class matrix is created in the same dimensions as Matrix 1 and *ClassMat* and is saved in "\$V" format to a file. Each element of

the class matrix contains the number of the interval to which the corresponding element of the previously loaded matrix belongs. The numbering of intervals starts with 1.

Intervals are numbered using integers. No decimal places are written. The matrix format of the class matrix cannot be changed. For each element of the classification matrix (ClassMat) the interval is determined.

For each interval the belonging values of the matrix are summed up. The result is stored into the result file. The definition of the intervals is contained in a code file. The width of the intervals are set individually. The lower limit is always included in the interval, whereas the upper limit is excluded.

Example of an interval code file:

```
$Interval:From;To;
MIN;1;
1;501;
501;1001;
1001;1501;
1501;2001;
2001;2501;
2501;MAX;
```

The result file has the following structure:

```
$VISION
$VERSION: VersNr;FileType;Language
1.0;MUULI;D
$Interval:From;To;Number;Share;Accum.Number;Accum.Share
MIN;1;0.00;0.0000;0.00;0.0000
1;501;5.00;0.2000;5.00;0.2000
501;1001;10.00;0.4000;15.00;0.6000
1001;1501;0.00;0.0000;15.00;0.6000
1501;2001;5.00;0.2000;20.00;0.8000
2001;2501;0.00;0.0000;20.00;0.8000
2501;MAX;5.00;0.2000;25.00;1.0000
```

MCompareWithMatrixAndGetSum ([in] VARIANT Matrix, [in] BSTR Operator, [out, retval] VARIANT *Sum)

Compares each element of the matrix (matrix 1) loaded using MLoad to *Matrix*. All elements satisfying the condition that is defined by Operator are summed up. The specified Matrix and the previously loaded matrix must have the same dimensions (number of zones and zone numbers).

Feasible operators are:

- < less than
- > greater than
- = equal
- # not equal
- <> not equal
- != not equal
- >= greater than or equal
- <= less than or equal
- == equal
- .eq. equal
- .ne. not equal
- .ge. greater than or equal
- .gr. greater than
- .ls. "less" (same as less than)
- .le. less than or equal

**MCalibri ([in] BSTR ODFile,
[in] BSTR DistFile,
[in] long Type,
[in] long Weight,
[in] long Iterations,
[in] long GravType,
[in] long Multiliteration,
[in] float MultiPrecisionFactor,
[in] BSTR ResultFile,
[in] BSTR ProtocolFile,
[in] VARIANT ResultMat,
[in] BSTR FormatString)**

The MCalibri command calls the calibration functionality. The utility matrix loaded via MLoad is still available in the main storage after MCalibri has been finished.

Parameters

[in] BSTR ODFile	OD file (file with origin and destination sums)
[in] BSTR DistFile	File with trip distribution in \$LS format or as an interval code file
[in] long Type	Utility function types: 1 corresponds to function $f(w_{ij}) = a \cdot w_{ij}^b \cdot e^{-c \cdot w_{ij}}$ 2 corresponds to function $f(w_{ij}) = a \cdot e^{-c \cdot w_{ij}}$ 2 is the default value. If other values than 1 or 2 are specified, the default value is used.
[in] long Weight	Weighting: 1 means weighted 2 means not-weighted 1 is the default value. If other values than 1 or 2 are specified, the default value is used.
[in] long Iterations	Max. number of iterations for Calibri calculation. Maximum number of iterations is 1000, the default value is 50. If values less than or equal to 0, or greater than 1000 are specified, the default value is used.
[in] long GravType	Gravity model: 1 corresponds to a single-constrained gravity model 2 corresponds to a double-constrained gravity model using the multi procedure (Lohse) 1 is the default value. If other values than 1 or 2 are specified, the default value is used.
[in] long Multiliteration	Max. number of iteration steps for multi procedure: Maximum number of iterations is 100, the default value is 10. If values less than or equal to 0, or greater than 100 are specified, the default value is used. This parameter is only regarded if required by the currently selected gravity model.
[in] float MultiPrecisionFactor	Precision factor for termination condition of the Multi procedure. This parameter is only regarded if required by the currently selected gravity model.
[in] BSTR ResultFile	Name of the result file (code file, may be used as input for gravity model calculation)
[in] BSTR ProtocolFile	Name of protocol file (comparing empirical and theoretical trip distribution)
[in] VARIANT ResultMat	Name of OD demand matrix
[in] BSTR FormatString	Format of OD demand matrix

VB example: 8.9.4

**MAggregate2 ([in] BSTR CodeFileName,
 [in] short Function,
 [in] short ApplyTo,
 [in, defaultvalue(0)] short UseLimits,
 [in, defaultvalue(0)] double lowerLimit,
 [in, defaultvalue(0)] double upperLimit)**

This function is applied to the matrix that has been loaded into the computer memory using MLoad before.

It aggregates the matrix.

Zones to be aggregated are coded in a code file. Similar to “interactive aggregation“, also for the COM command several parameters can be set. Parameters contained in the code file are substituted by the ones taken from the COM procedure call.

Note: This script function does not support the "Weighted mean" option. For this purpose, specific commands are provided (AggregateWithWeightMatrix, MAggregateWithWeightMatrix).

Parameters

[in] BSTR CodeFileName	Name of file with Aggregation code From VISUM 9.3, the code file may contain the following parameters that before had to be specified separately:
[in] short Function	0 – Sum 1 – min 2 – max 3 – mean value
[in] short ApplyTo	0 – rows and columns 1 – rows only 2 – columns only
[in, defaultvalue(0)] short UseLimits	0 – do not use limits (default) 1 – use limits
[in, defaultvalue(0)] double lowerLimit	value of lower limit (default 0)
[in, defaultvalue(0)] double upperLimit	value of upper limit (default 0)

The optional parameters UseLimits, LowerLimit, UpperLimit have not necessarily to be specified with the command. If parameter UseLimits is set (1 or true), then the lower limit is used for values that are less than the lower limit, and the upper limit is used for values exceeding this upper limit.

**MAggregateWithWeightMatrix ([in] BSTR CodeFileName,
 [in] BSTR WeightMat,
 [in] short ApplyTo,
 [in, defaultvalue(0)] short UseLimits,
 [in, defaultvalue(0)] double lowerLimit,
 [in, defaultvalue(0)] double upperLimit)**

This function is applied to the matrix that has been loaded into the computer memory using MLoad before. Aggregates a matrix using a weight matrix, i.e. the matrix value of each aggregated zone is multiplied by the factor from the weighting matrix. (See VISUM Manual: "Visual aggregation" and "Weighted mean".) Parameters contained in the code file are substituted by the ones taken from the COM procedure call.

Parameters

[in] BSTR CodeFileName	Name of file with Aggregation code From VISUM 9.3, the code file may contain the
------------------------	---

	following parameters that before had to be specified separately:
[in] BSTR WeightMat	Name of weight matrix
[in] short ApplyTo	0 – rows and columns 1 – rows only 2 – columns only
[in, defaultvalue(0)] short UseLimits	0 – do not use limits (default) 1 – use limits
[in, defaultvalue(0)] double lowerLimit	value of lower limit (default 0)
[in, defaultvalue(0)] double upperLimit	value of upper limit (default 0)

The optional parameters UseLimits, LowerLimit, UpperLimit have not necessarily to be specified with the command. If parameter UseLimits is set (1 or true), then the lower limit is used for values that are less than the lower limit, and the upper limit is used for values exceeding this upper limit.

Input matrix and weighting matrix must have identical dimensions (number of zones and zone numbers).

VB example: 8.9.5, 8.9.6

**MTripDistribution ([in] BSTR ODFile,
[in] short functionType,
[in] double a,
[in] double b,
[in] double c,
[in] short constraintType,
[in] short normType,
[in] short multilter,
[in] double multiPrecisionFactor)**

This function calculates the trip distribution on the matrix that has been loaded using MLoad.

Note: When you use a gravity model code file as an OD file, the code file may also contain all parameters. Parameters contained in the code file are substituted by the ones taken from the COM procedure call.

Parameters

[in] BSTR ODFile	Code file, VISUM attribute file or OD file
[in] short functionType	Utility function types: 1 – combined (default), 2 – TModel 3 – BoxCox 4 – Logit 5 – Kirchhoff If an invalid value is entered, the default value is used.
[in] double a	Parameter "a" of the utility function
[in] double b	Parameter "b" of the utility function
[in] double c	Parameter "c" of the utility function
[in] short constraintType	Constraint: 1 – singly-constrained in terms of creation (origin-bound), default 2 – singly-constrained in terms of attraction (destination-bound) 3 – doubly-constrained (multi procedure) If an invalid value is entered, the default value is used.

[in] short normType	Standardization of matrix sum for multi procedure: 1 = Sum of creations (default) 2 = Sum of attractions 3 = Mean of sums 4 = Minimum of sums 5 = Maximum of sums If an invalid value is entered, the default value is used.
[in] short multilter	Number of iterations for multi procedure. value between 1 and 100 (incl.), default 10
[in]double multiPrecisionFactor	precision factor (multi procedure: termination condition), floating-point number, default = 3

MProjectionOnExpectedValue ([in] double value)

This command uses the matrix loaded via MLoad for projection: each OD relation of the matrix is projected such that the preset matrix sum is reached. (hitherto, \$HGS had to be used)

**MFratar ([in] BSTR ODFile,
[in] short constraintType,
[in] short expectedValue,
[in, defaultvalue(1)] short multiNormType,
[in, defaultvalue(10)] short multilter,
[in, defaultvalue(3.0)]double multiPrecisionFactor)**

This function calculates a single- or double-constrained projection using either a factor or a target value (hitherto, \$HQ, \$HQS, \$HZ, \$HZS, \$H, \$HS had to be applied, see the VISUM Manual).

Parameters

[in] BSTR ODFile	VISUM attribute file, OD file or code file
[in] short constraintType	Constraint: 1 – singly-constrained in terms of creation (origin-bound), default 2 – singly-constrained in terms of attraction (destination-bound) 3 – doubly-constrained (multi procedure) If an invalid value is entered, the default value is used.
[in] short expectedValue	0 – projection using a factor 1 – projection using target value
[in, defaultvalue(1)] short multiNormType	Standardization of matrix sum for multi procedure 1 = Sum of creations (default) 2 = Sum of attractions 3 = Mean of sums 4 = Minimum of sums 5 = Maximum of sums If an invalid value is entered, the default value is used.
[in, defaultvalue(10)] short multilter	Number of iterations for multi procedure. value between 1 and 100 (incl.), default 10
[in, defaultvalue(3.0)]double multiPrecisionFactor	precision factor (Multi procedure: Termination condition), floating-point number, default = 3

The optional parameters multiNormType, multilter, MultiQuality have not necessarily to be specified with the command. In case of missing parameter values, the default value will be

used. These parameters are only regarded if the multi procedure is activated (constraintType = 3).

Instead of the *OD file*, you can use a code file. Parameters contained in the code file are substituted by the ones taken from the COM procedure call.

8.7 Creating a Matrix

NewConstantMatrix ([in] long Dimension,
[in] VARIANT Value,
[in] VARIANT MatResult,
[in] BSTR FormatString)

Creates a new matrix with the dimensions (*Dimension*Dimension*). All elements of the matrix are set to the constant *Value*. Zone numbers are created starting with 1 (1..N, N = *Dimension*). The result is saved to the *Matresult* matrix in the *FormatString* format.

GenerateConstantMatrix ([in] VARIANT Mat,
[in] VARIANT Value,
[in] VARIANT MatResult,
[in] BSTR FormatString)

Creates a new matrix with the dimensions and zone numbers of *Mat*. All elements of the matrix are set to the constant *Value*. The zone numbers of the *Mat* matrix are used. The result is saved to the *Matresult* matrix in the *FormatString* format.

8.8 Matrix Functions

Add ([in] VARIANT Mat1,
[in] VARIANT Mat2,
[in] VARIANT MatResult,
[in] BSTR FormatString)

Adds elements of *Mat2* to *Mat1*. Different matrix dimensions permitted. The result is saved to the *Matresult* matrix in the *FormatString* format.

Subtract ([in] VARIANT Mat1,
[in] VARIANT Mat2,
[in] VARIANT MatResult,
[in] BSTR FormatString)

Subtracts elements of *Mat2* from *Mat1*. Different matrix dimensions permitted. The result is saved to the *Matresult* matrix in the *FormatString* format.

Multiply ([in] VARIANT Mat1,
[in] VARIANT Mat2,
[in] VARIANT MatResult,
[in] BSTR FormatString)

Multiplies elements of *Mat1* by elements of *Mat2*. Identical matrix dimensions are required (identical number of zones and identical zone numbers). The result is saved to the *Matresult* matrix in the *FormatString* format.

**Divide ([in] VARIANT Mat1,
[in] VARIANT Mat2,
[in] VARIANT MatResult,
[in] BSTR FormatString)**

Divides elements of *Mat1* by elements of *Mat2*. Identical matrix dimensions are required (identical number of zones and identical zone numbers). The result is saved to the *Matresult* matrix in the *FormatString* format.

**Max ([in] VARIANT Mat1,
[in] VARIANT Mat2,
[in] VARIANT MatResult,
[in] BSTR FormatString)**

Compares *Mat1* to *Mat2* and writes the maximum value to the result matrix, i.e. $\text{MAX}(\text{Mat1}[i,j], \text{Mat2}[i,j])$. Identical matrix dimensions are required (identical number of zones and identical zone numbers). The result is saved to the *Matresult* matrix in the *FormatString* format.

**Min ([in] VARIANT Mat1,
[in] VARIANT Mat2,
[in] VARIANT MatResult,
[in] BSTR FormatString)**

Compares *Mat1* to *Mat2* and writes the minimum value to the result matrix, i.e. $\text{MIN}(\text{Mat1}[i,j], \text{Mat2}[i,j])$. Identical matrix dimensions are required (identical number of zones and identical zone numbers). The result is saved to the *Matresult* matrix in the *FormatString* format.

**AddValue ([in] VARIANT Mat,
[in] VARIANT Value,
[in] VARIANT MatResult,
[in] BSTR FormatString)**

Adds *Value* to each *Mat* matrix element. The result is saved to the *Matresult* matrix in the *FormatString* format.

**SubtractValue ([in] VARIANT Mat,
[in] VARIANT Value,
[in] VARIANT MatResult,
[in] BSTR FormatString)**

Subtracts *Value* from each *Mat* matrix element. The result is saved to the *Matresult* matrix in the *FormatString* format.

**MultiplyValue ([in] VARIANT Mat,
[in] VARIANT Value,
[in] VARIANT MatResult,
[in] BSTR FormatString)**

Multiplies *Value* with each *Mat* matrix element. The result is saved to the *Matresult* matrix in the *FormatString* format.

**DivideValue ([in] VARIANT Mat,
[in] VARIANT Value,
[in] VARIANT MatResult,
[in] BSTR FormatString)**

Divides each *Mat* matrix element by *Value*. The result is saved to the *Matresult* matrix in the *FormatString* format.

**MaxValue ([in] VARIANT Mat,
[in] VARIANT Value,
[in] VARIANT MatResult,
[in] BSTR FormatString)**

Compares each *Mat* matrix element with the *Value* and writes the maximum value to the results matrix, i.e. $\text{MAX}(\text{Mat}[i,j], \text{value})$. The result is saved to the *Matresult* matrix in the *FormatString* format.

**MinValue ([in] VARIANT Mat,
[in] VARIANT Value,
[in] VARIANT MatResult,
[in] BSTR FormatString)**

Compares each element of the *Mat* matrix with *Value* and writes the minimum value to the result matrix, i.e. $\text{MIN}(\text{Mat}[i,j], \text{value})$. The result is saved to the *Matresult* matrix in the *FormatString* format.

Ln ([in] VARIANT Mat, [in] VARIANT MatResult, [in] BSTR FormatString)

Applies logarithm to each element of the *Mat* matrix (natural logarithm). The result is saved to the *Matresult* matrix in the *FormatString* format.

Exp ([in] VARIANT Mat, [in] VARIANT MatResult, [in] BSTR FormatString)

Applies exponential function to each element in the *Mat* matrix. The result is saved to the *Matresult* matrix in the *FormatString* format.

**ReciprocalValue ([in] VARIANT Mat,
[in] VARIANT MatResult,
[in] BSTR FormatString)**

Forms reciprocal for each element in the *Mat* matrix. The result is saved to the *Matresult* matrix in the *FormatString* format.

**Pow ([in] VARIANT Mat,
[in] VARIANT Value,
[in] VARIANT MatResult,
[in] BSTR FormatString)**

Raises each *Mat* matrix element to the power of *Value*. The result is saved to the *Matresult* matrix in the *FormatString* format.

**SetDiagonal ([in] VARIANT Mat,
[in] VARIANT Value,
[in] VARIANT MatResult,
[in] BSTR FormatString)**

The matrix elements on the (main) diagonal are set to *Value*. Other matrix elements remain unchanged. The result is saved to the *Matresult* matrix in the *FormatString* format.

**ExtractDiagonal ([in] VARIANT Mat,
[in] VARIANT MatResult,
[in] BSTR FormatString)**

Extracts the values from diagonal. All elements in the result matrix will be zero, except for the diagonal matrix cells. Their values are set to the diagonal values of the *Mat* matrix. The result is saved to the *Matresult* matrix in the *FormatString* format.

**MirrorUpperTriangle ([in] VARIANT Mat1,
[in] VARIANT MatResult,
[in] BSTR FormatString)**

The part of the matrix which is situated above the diagonal (the upper triangle) is mirrored to the lower triangle. The result is saved to the *Matresult* matrix in the *FormatString* format.

**MirrorLowerTriangle ([in] VARIANT Mat1,
[in] VARIANT MatResult,
[in] BSTR FormatString)**

The part of the matrix which is situated below the diagonal (the lower triangle) is mirrored to the upper triangle. The result is saved to the *Matresult* matrix in the *FormatString* format.

**Transpose ([in] VARIANT Mat1,
[in] VARIANT MatResult,
[in] BSTR FormatString)**

Transposes columns and rows of the matrix. The result is saved to the *Matresult* matrix in the *FormatString* format.

**Aggregate ([in] VARIANT Mat,
[in] VARIANT MatResult,
[in] BSTR CodeFileName,
[in] BSTR FormatString)**

Aggregates multiple zones of a matrix to one or more new zones. Zones can be aggregated to larger units. Zones can be renamed and / or renumbered or removed from the matrix. A code file (ASCII format) is required for aggregation. The result is saved to the *Matresult* matrix in the *FormatString* format.

**Split ([in] VARIANT Mat,
[in] VARIANT MatResult,
[in] BSTR CodeFileName,
[in] BSTR FormatString)**

Using the split function zones can be subdivided into smaller units. A code file (ASCII format) is required for splitting. The result is saved to the *Matresult* matrix in the *FormatString* format.

**Gravitate ([in] VARIANT Mat,
[in] VARIANT MatResult,
[in] BSTR CodeFileName,
[in] BSTR FormatString)**

Calculates the gravitation model for the matrix. A code file (ASCII format) is required for calculating the gravitation model. The result is saved to the *Matresult* matrix in the *FormatString* format.

VB example: 8.9.3

**Project ([in] VARIANT Mat,
[in] VARIANT MatResult,
[in] BSTR CodeFileName,
[in] BSTR FormatString)**

You need a code file (ASCII format) for extrapolation. The result is saved to the *Matresult* matrix in the *FormatString* format.

VB example: 8.9.3

**AggregateWithWeightMatrix ([in] VARIANT Mat,
[in] VARIANT MatResult,
[in] BSTR FormatString,
[in] BSTR CodeFileName,
[in] BSTR WeightMat,
[in] short ApplyTo,
[in, defaultvalue(0)] short UseLimits,
[in, defaultvalue(0)] double lowerLimit,
[in, defaultvalue(0)] double upperLimit)**

Aggregates a matrix using a weight matrix, i.e. the matrix value of each aggregated zone is multiplied by the factor from the weighting matrix.

Parameters

[in] VARIANT Mat	Name of input matrix
[in] VARIANT MatResult	Name of result matrix
[in] BSTR FormatString	Format of result matrix, e.g. \$V;D2
[in] BSTR CodeFileName	Name of code file with aggregation code. The code file may also contain the following parameters. These will be replaced by the ones specified in the function call if so.
[in] BSTR WeightMat	Name of weight matrix
[in] short ApplyTo	0 – rows and columns (default) 1 – Rows only 2 – Columns only
[in, defaultvalue(0)] short UseLimits	0 – do not use limits (default) 1 – use limits
[in, defaultvalue(0)] double lowerLimit	value of lower limit (default 0)
[in, defaultvalue(0)] double upperLimit	value of upper limit (default 0)

The optional parameters UseLimits, LowerLimit, UpperLimit have not necessarily to be specified with the command. If the UseLimits parameter is set to 1, then the lower limit is used for values that are less than the lower limit, and the upper limit is used for values that exceed the upper limit.

Input matrix and weighting matrix must have identical dimensions (number of zones and zone numbers).

**Aggregate2 ([in] VARIANT Mat,
[in] VARIANT MatResult,
[in] BSTR FormatString,
[in] BSTR CodeFileName,
[in] short Function,
[in] short ApplyTo,
[in, defaultvalue(0)] short UseLimits,
[in, defaultvalue(0)] double lowerLimit,
[in, defaultvalue(0)] double upperLimit)**

It aggregates the matrix. Zones to be aggregated are coded in a code file. Similar to “interactive aggregation“, also for the COM command several parameters can be set.

Note: This script function does not support the "Weighted mean" option. For this purpose, specific commands are provided (AggregateWithWeightMatrix, MAggregateWithWeightMatrix).

Parameters

[in] VARIANT Mat	Name of input matrix
------------------	----------------------

[in] VARIANT MatResult	Name of result matrix
[in] BSTR FormatString	Format of result matrix, e.g. \$V;D2
[in] BSTR CodeFileName	Name of code file with aggregation code. The code file may also contain the following parameters. These will be replaced by the ones specified in the function call if so.
[in] short Function	0 – Sum 1 – min 2 – max 3 – mean value
[in] short ApplyTo	0 – rows and columns (default) 1 – Rows only 2 – Columns only
[in, defaultvalue(0)] short UseLimits	0 – do not use limits (default) 1 – use limits
[in, defaultvalue(0)] double lowerLimit	value of lower limit (default 0)
[in, defaultvalue(0)] double upperLimit	value of upper limit (default 0)

The optional parameters UseLimits, LowerLimit, UpperLimit have not necessarily to be specified with the command. If parameter UseLimits is set (1 or true), then the lower limit is used for values that are less than the lower limit, and the upper limit is used for values exceeding this upper limit.

**TripDistribution ([in] VARIANT Mat,
[in] VARIANT MatResult,
[in] BSTR ODFile,
[in] short functionType,
[in] double a,
[in] double b,
[in] double c,
[in] short constraintType,
[in] short normType,
[in] short multilter,
[in] double multiPrecisionFactor,
[in] BSTR FormatString)**

This function calculates trip distribution for the *Mat* matrix.

Note: When you use a gravity model code file as an *OD file*, the code file may also contain all parameters. Parameters contained in the code file are substituted by the ones taken from the COM procedure call.

Parameters

[in] VARIANT Mat	Name of input matrix (impedance matrix)
[in] VARIANT MatResult	Name of result matrix
[in] BSTR ODFile	Code file, VISUM attribute file or OD file
[in] short functionType	Utility function types: 1 – combined (default), 2 – TModel 3 – BoxCox 4 – Logit 5 – Kirchhoff If an invalid value is entered, the default value is used.
[in] double a	Parameter "a" of the utility function
[in] double b	Parameter "b" of the utility function
[in] double c	Parameter "c" of the utility function

[in] short constraintType	<p>Constraint:</p> <p>1 – singly-constrained in terms of creation (origin-bound), default</p> <p>2 – singly-constrained in terms of attraction (destination-bound)</p> <p>3 – doubly-constrained (multi procedure)</p> <p>If an invalid value is entered, the default value is used.</p>
[in] short normType	<p>Standardization of matrix sum for multi procedure:</p> <p>1 = Sum of creations (default)</p> <p>2 = Sum of attractions</p> <p>3 = Mean of sums</p> <p>4 = Minimum of sums</p> <p>5 = Maximum of sums</p> <p>If an invalid value is entered, the default value is used.</p>
[in] short multilter	<p>Number of iterations for multi procedure. value between 1 and 100 (incl.), default 10</p>
[in]double multiPrecisionFactor	<p>precision factor (multi procedure: termination condition), floating-point number, default = 3</p>
[in] BSTR FormatString	<p>Format of result matrix, e.g. \$V;D2</p>

**ProjectionOnExpectedValue ([in] VARIANT Mat,
[in] VARIANT MatResult,
[in] BSTR FormatString,
[in] double value)**

This command calculates the projection for the matrix: each OD relation of the matrix is projected such that the preset matrix sum is reached. The result is saved to the *Matresult* matrix in the *FormatString* format.

Parameters

[in] VARIANT Mat	Name of input matrix (impedance matrix)
[in] VARIANT MatResult	Name of result matrix
[in] BSTR FormatString	Format of result matrix, e.g. \$V;D2
[in] double value	Forced matrix sum

**Fratar ([in] VARIANT Mat,
[in] VARIANT MatResult,
[in] BSTR FormatString,
[in] BSTR ODFile,
[in] short constraintType,
[in] short expectedValue,
[in, defaultvalue(1)] short multiNormType,
[in, defaultvalue(10)] short multilter,
[in, defaultvalue(3.0)] double multiPrecisionFactor)**

This function calculates a single- or double-constrained projection using either a factor or a target value (hitherto, \$HQ, \$HQS, \$HZ, \$HZS, \$H, \$HS had to be applied, see the VISUM Manual).

Parameters

[in] VARIANT Mat	Name of input matrix (impedance matrix)
[in] VARIANT MatResult	Name of result matrix
[in] BSTR FormatString	Format of result matrix, e.g. \$V;D2
[in] BSTR ODFile	VISUM attribute file, OD file or code file

[in] short constraintType	Constraint 1 – singly-constrained in terms of creation (default) 2 – singly-constrained in terms of attraction 3 – doubly-constrained (multi procedure) If an invalid value is entered, the default value is used.
[in] short expectedValue	0 – projection with factor 1 – projection to target value
[in, defaultvalue(1)] short multiNormType	Standardization of matrix sum for multi procedure 1 = Sum of creations (default) 2 = Sum of attractions 3 = Mean of sums 4 = Minimum of sums 5 = Maximum of sums If an invalid value is entered, the default value is used.
[in, defaultvalue(10)] short multilter	Number of iterations for multi procedure. value between 1 and 100 (incl.), default 10
[in, defaultvalue(3.0)]double multiPrecisionFactor	precision factor (Multi procedure: Termination condition), floating-point number, default = 3

The optional parameters multiNormType, multilter, MultiQuality have not necessarily to be specified with the command. In case of missing parameter values, the default value will be used. These parameters are only regarded if the multi procedure is activated (constraintType = 3).

Instead of the *OD file*, you can use a code file. Parameters contained in the code file are substituted by the ones taken from the COM procedure call.

8.9 Muuli Examples

8.9.1 Creating separate Muuli COM Object

The following example shows how to create a separate Muuli COM object.

```
Sub CreateMuuli()
  Dim muuli as Object
  Set muuli = CreateObject("VISUM130.ScriptMuuli")
  Dim versionNo As Variant
  versionNo = Muuli.GetVersionNumber()
  MsgBox versionNo
  Set muuli = Nothing
End Sub
```

8.9.2 Creating Muuli COM Object embedded in VISUM

An alternative way is to create a Muuli COM object embedded in VISUM.

References: Fehler! Verweisquelle konnte nicht gefunden werden. 8.2

```
Sub CreateMuuliFromVISUM()
  Dim visum as Object
  Set visum = CreateObject("Visum.Visum.130")
```

```

Dim versionNo As Variant
versionNo = visum.MatrixEditor.GetVersionNumber()
MsgBox versionNo
Set visum = Nothing
End Sub

```

In both examples the call of “Muuli” and “visum.MatrixEditor” returns the same object which offers the same methods, e.g. here “GetVersionNumber()”.

8.9.3 Destination-Coupled Gravitation Model

The following example shows the calculation of a destination coupled gravitation model. The matrices are specified explicitly.

References: 8.8

```

' Calculation of matrix using the destination-coupled gravitation model and
' extrapolation with margin alignment to fulfill both coupling conditions.
' *****
' Declaration of variables
Dim muuli          ' Variable that points to Muuli object
Dim sMat1          ' String with matrix file name
Dim sMat2          ' String with matrix file name
Dim sWidMat        ' String with file name of impedance matrix
Dim sOutPutFormat  ' String for formatting of results matrix
Dim sCodeFileGravit ' Name of code file for gravitation model
Dim sCodeFileHochr  ' Name of code file for extrapolation
Dim RetVal         ' dummy return value for Muuli operation(s)

' *****
' create the Muuli Object
Set muuli = CreateObject("VISUM130.ScriptMuuli")

' Set Outputformat
sOutPutFormat = "$V;d2"

' Set Matrices
sWidMat      = "widerst.mat"
sMat1       = "mat1.fma"
sMat2       = "mat2.fma"

' Set Code-File
sCodeFileGravit = "gravit.cod"
sCodeFileHochr  = "grahoch.cod"

' Calculation of matrix using the destination-coupled gravitation model
RetVal = muuli.Gravitare (sWidMat, sMat1, sCodeFileGravit, sOutPutFormat)

' Extrapolation with margin alignment to fulfill both coupling conditions.
RetVal = muuli.Project (sMat1, sMat2, sCodeFileHochr, sOutPutFormat)

Set muuli = Nothing

```

8.9.4 Calibri Method

This example uses the Calibri method. It deals with previously loaded matrices.

References: 8.6

```

' The following options are set:
' - utility function  $f(w_{ij}) = a * w_{ij}^b * e^{(-c * w_{ij})}$ 

```



```

' - weighted
' - max. 50 iteration steps for Calibri calculation
' - use of source-coupled gravitation model

'*****

' create the Muuli Object
Set muuli = CreateObject("VISUM120.ScriptMuuli")

'Set Matrices
sImpMat = "Test.dis"

sQZFile = "Test.QZ"
sDistFile = "Test.lst"

sResFile = "ScriptResFile.cod"
sProtFile = "ScriptProt.out"
sTripMat = "ScriptMatrix.fma"

sFormat = "$V;D2"

'load impedance matrix
muuli.MLoad sImpMat

muuli.MCalibri sQzFile, sDistFile, 1, 1, 50, 1, 0, 0, sResFile, sProtFile,
sTripMat, sFormat

Set muuli = Nothing

```

8.9.5 Aggregating a Matrix without Optional Parameters

This example aggregates a previously loaded matrix (weighted aggregation) without respecting upper and lower limits and without specifying the optional parameters.

References: 8.6

```

' create the Muuli Object
Set muuli = CreateObject("VISUM130.ScriptMuuli")

'Set Matrices
Mat = "Matrix1.mtx"
WeightMat = "Weight.mtx"
MatOut = "Out.mtx"

CodeFile = "AggCode.cod"
sFormat = "$V;D4"

muuli.MLoad Mat
muuli.MAggregateWithWeightMatrix CodeFile, WeightMat, 0
muuli.MSave MatOut, sFormat

Set muuli = Nothing

```

8.9.6 Aggregating a Matrix with Optional Parameters

This example aggregates a previously loaded matrix (weighted aggregation) with respecting upper and lower limits and with the specified optional parameters.

References: 8.6

```

' create the Muuli Object

```

```
Set muuli = CreateObject("VISUM130.ScriptMuuli")

'Set Matrices
Mat = "Matrix1.mtx"
WeightMat = "Weight.mtx"
MatOut = "Out.mtx"

CodeFile = "AggCode.cod"
sFormat = "$V;D4"

muuli.MLoad Mat
muuli.MAggregateWithWeightMatrix CodeFile, WeightMat, 0, 1, 10, 1000
muuli.MSave MatOut, sFormat
Set muuli = Nothing
```



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
