QlikView

Automation Interface Reference

(Updated for version 9.0)
2009-06-01

QlikView

Version 9.0 for Microsoft Windows®.

Lund, Sweden, June 2009
Authored by QlikTech International AB /JNN/ICO

Table of contents

Using QlikView with Automation and macros

Using QlikView with Automation and macros

1. Introduction

1.1. About this documentation

General

This documentation is provides a description of the QlikView Automation interface. The printed documentation is supplemented by a special corresponding QlikView qvw document APIguide.qvw, which can be found on the QlikView installation CD and on the QlikTech homepage. However, make sure that you have an updated version.

Visual Basic programming language

The Visual basic programming language is the typical tool for using the QlikView Automation interface. This document is however not a manual for Visual Basic. The reader is supposed to possess a basic knowledge of Visual Basic programming.

1.2. Disclaimers

- Some specifications may be changed during the continued development.
- As changes are made to the QlikView type library i future releases VB programs using typed object calls will have to be recompiled with the new QlikView.tlb in order to function with the new release.
- The QlikView automation interface lets you manipulate QlikView objects on a very deep internal level. Incorrect use of functionality may cause program errors.

1.3. Documentation outline

This document covers the general features of the QlikView Automation interface and some aspects of the Visual Basic programming language.

Chapter 2
gives you an overview of the user interface for using macros in QlikView and the Automation interface in terms of class hierarchies.

Chapter 3
discusses some aspects of the Visual Basic programming language.

Chapter 4
discusses how to get hold of a QlikView as an Automation object.

Chapter 5
discusses how to get hold of the local file system from VBScript.

Chapter 6
discusses how to use VBScript functions in the QlikView script.

Chapter 7
discusses how to use Automation for dynamic data update.

2. Using Automation and macros with QlikView

2.1. The QlikView Automation Interface

About Automation

QlikView is equipped with an Automation interface (Automation was previously known as OLE Automation). This interface allows an external program or internal macro to access and control the QlikView application.

Invokation of QlikView Automation

The Automation interface is an integral part of QlikView and you do not have to perform any special tasks to activate it.

In order for the Automation interface to work, you need a QlikView type library. The QlikView type library is integrated in the ***QV.exe*** file. No further steps are necessary to install the type library.

If more than one instance of QlikView is running on a computer, the type library of the instance first started will be in effect.

2.2. How Automation and macros can control QlikView

External control of QlikView

QlikView objects are accessible by means of Automation from external programs,
e.g. programs written in Visual Basic or C++ supporting Automation.

Such code can be used to control QlikView from other applications or from stand-alone programs.

Stand-alone executable files can be invoked from a QlikView document by means of launch buttons.

Internal VBScript interpreter

QlikView objects are also accessible via Automation from inside QlikView by means of the built-in VBScript interpreter.

Macros written in VBScript inside a QlikView application can currently be invoked as an action trigger in several ways:

Application events:
1. A macro can be run after opening a QlikView document.
3. A macro can be run after script re-execution.
4. A macro can be run after the Reduce Data command.
5. A macro can be run after a selection in any field in the application.

Sheet events
6. A macro can be run after a sheet is activated.
7. A macro can be run when a sheet is deactivated.

Sheet object events
8. A macro can be run after a sheet object is activated.
9. A macro can be run when a sheet object is deactivated.

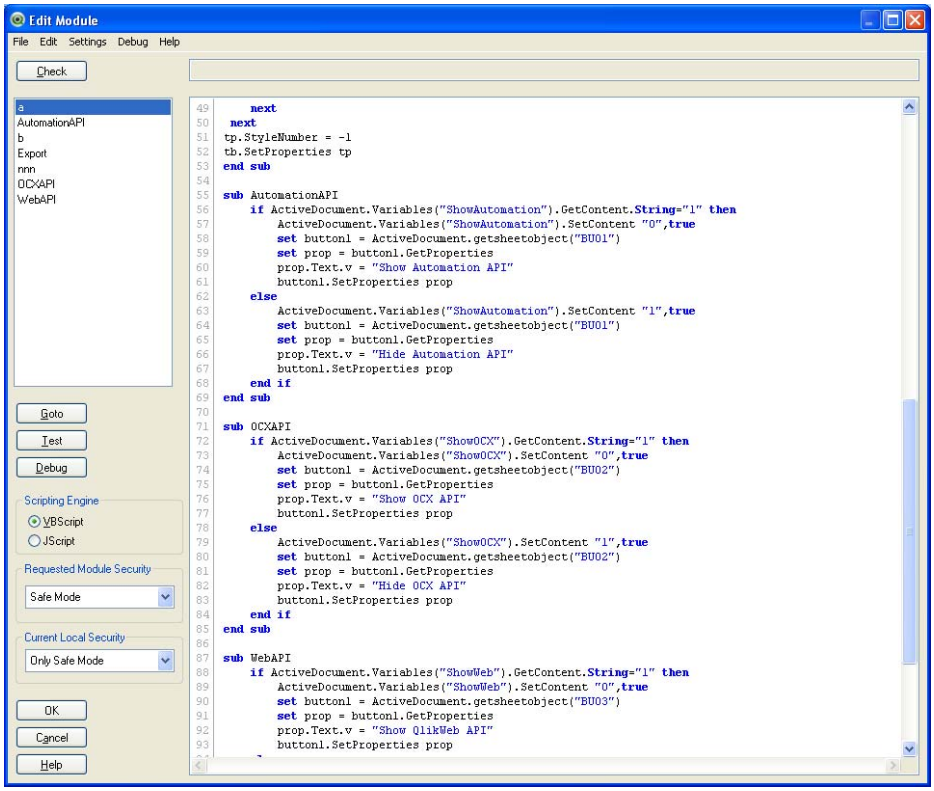
Button events
10. A button sheet object can be linked to a macro

Field events
11. A macro can be run after a selection has been made in a specified field.
12. A macro can be run when a selection is made in any field logically associated with a specified field.
13. A macro can be run after selections have been locked in a specified field.
14. A macro can be run after selections have been locked in a specified field.

Variable events
15. A macro can be run after a value has been entered in a specified variable.
16. A macro can be run when the value of a specified variable containing a formula has been changed due to a change in the formula value.

2.3. Edit Module dialog

Macros and custom defined functions can be written in VBScript using the Edit Module dialog. The module is saved with the application.



By choosing Edit Module from the File menu you will enter the Edit Module dialog. The following controls are available:

The Edit Module dialog

Macro edit box
: A large edit box where you type your macros. All macros should be written as subroutines or functions between a matching pair of **sub .. end sub** or **function .. end function** statements.

Check
: Once you have written a sub, you can have it syntax-checked and recognized as a valid entry point by the VBScript interpreter by clicking this button.

Message
: The current status and any error messages will be displayed in this box.

List of entry points
: As soon as an entry point has been recognized by the VBScript interpreter, it will appear in the list of entry points to the left in the dialog. Here you can also select an entry point.

Test
: After selecting an entry point in the list of entry points you can test an individual macro by clicking on this button. Error messages will be displayed in the Message box.

Scripting Engine
: Sets the scripting engine for the document. You may choose between VBScript and JScript. This documentation is written entirely for the VBScript option.

Intended Module Security Level
: The designer of the QlikView document can set the intended macro security level to **Safe Mode** or **System Access**. By indicating **Safe Mode** the document designer indicates that the macros in the module do not contain any code that can access the system or applications outside QlikView. Typically this would mean code containing `CreateObject`, `GetObject` or `Launch`. If such code is encountered during macro execution in a document declared to be in **Safe Mode**, the execution will fail. If however the document designer indicates **System Access** mode the end user will be prompted (see figure x) when opening the document to approve system access (**Allow System Access**), disable all macros in the document (**Disable Macros**) or allow only macros without system access (**Safe Mode**). As soon as the user has chosen to approve or ban the macros this will be remembered by the system and no more prompts will appear when opening the document.

User Allowed Security Level
: When opening a document declared to be in safe mode by the document's designer or a document created with QlikView version 5.01 or earlier, which contains potentially unsafe code in the macro module script, the user will be prompted to approve, disable or partially disable macros (see above). This choice will be remembered by the system but can be changed at any later time via this setting. The macro security level can be set to **Don't Run at All**, **Only Safe Mode** or **Allow System Access**.

OK
: This button saves changes and takes you back to the QlikView main menu.

Cancel
: This button discards changes and takes you back to the QlikView main menu.

Furthermore, the Edit Module dialog contains a number of commands in four menus:

FILE menu

Export to Module File...
: Saves the contents of the edit box in a text file to be specified in the Save Module dialog that appears. The file will have the extension .qvm.

Print...
: Opens the Print Options dialog (see page 327).

EDIT menu

Undo
: Undoes the latest change.

Redo
: Redoes the latest Undo.

Copy
: Copies the selected text to the Clipboard.

Cut
: Exports the selected text to the Clipboard.

Paste
: Pastes the contents of the Clipboard into the dialog at the position of the cursor.

Clear
: Clears all the text in the edit box.

Select All
: Selects all the text in the edit box.

Insert File...
: Lets you browse for a file containing macros and inserts its contents into the edit box at the position of the cursor.

Find/Replace
: Opens a dialog allowing you to find and replace numbers or characters in the macros.

SETTINGS menu

Configure
: Opens the Editor page in the User Preferences dialog (page 68), where you can set the font and color of the different text types appearing in the edit box.

HELP menu

Help
: Opens the QlikView help program.

2.4. Halting a macro that is running

A running macro can normally be interrupted by pressing CTRL+BREAK on the keyboard. You may however have to close open dialog boxes originating from the macro before this command can be executed.

The CTRL+BREAK option can be turned on and off via the Automation calls Application.EnableCtlBrk and Application.DisableCtlBrk.

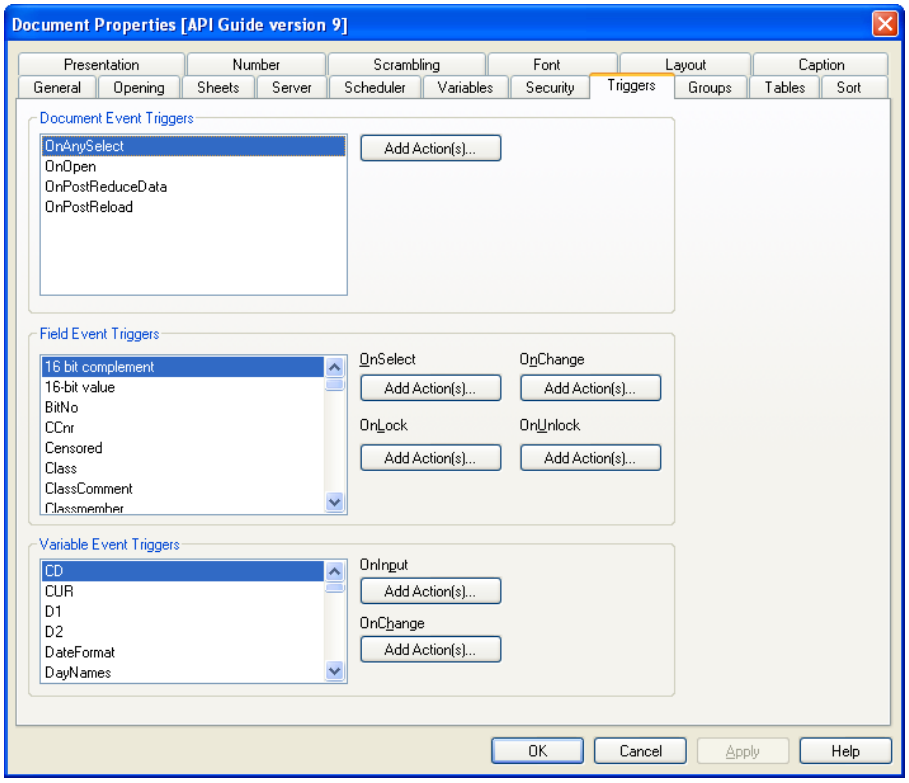
2.5. Invoking macros

Invoking macros on document, field and variable events

Edit Module — The Edit Module dialog is opened from the Tools menu.

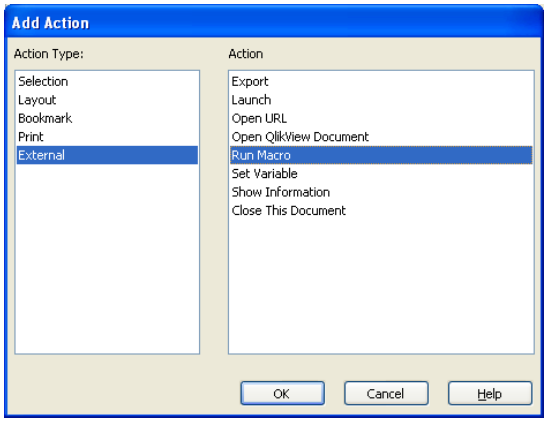
Password — This allows the application developer to protect the Edit Module dialog with a password. To set a password, click the Module Password button on the Security page of the Document Properties dialog.

Choose the Triggers page under Document Properties.



Add Action(s)…

In this dialog you will find an action that runs a macro among the external action types. Note: **Actions** add simplicity and are intended as an alternative to macros. Wherever **Actions** fulfill an intended purpose, they should be used instead of macros.



Document Event Triggers

In this group you can set macros to trigger on selected events in the document.

OnAnySelect — A run macro action, if added, will be executed each time a selection has been made in any field of the QlikView document.

OnOpen — A run macro action, if added, will be executed each time the QlikView document is opened.

OnPostReduceData — A run macro action, if added, will be executed after each time the Reduce Data command has been executed.

OnPostReload — A run macro action, if added, will be executed each time the script has been re-executed.

Field Event Triggers

In this group you can set macros to trigger on changes in the logical state of a specified field in the application. You must first select a field in the list before you may assign macros to events in it.

OnSelect — A run macro action, if added, will be executed each time a selection has been made in the selected field.

OnChange — A run macro action, if added, will be executed each time a selection has been made in any field which is logically associated with the selected field.

OnLock — A run macro action, if added, will be executed each time the field is locked.

OnUnlock — A run macro action, if added, will be executed each time the field is unlocked.

Variable Event Triggers

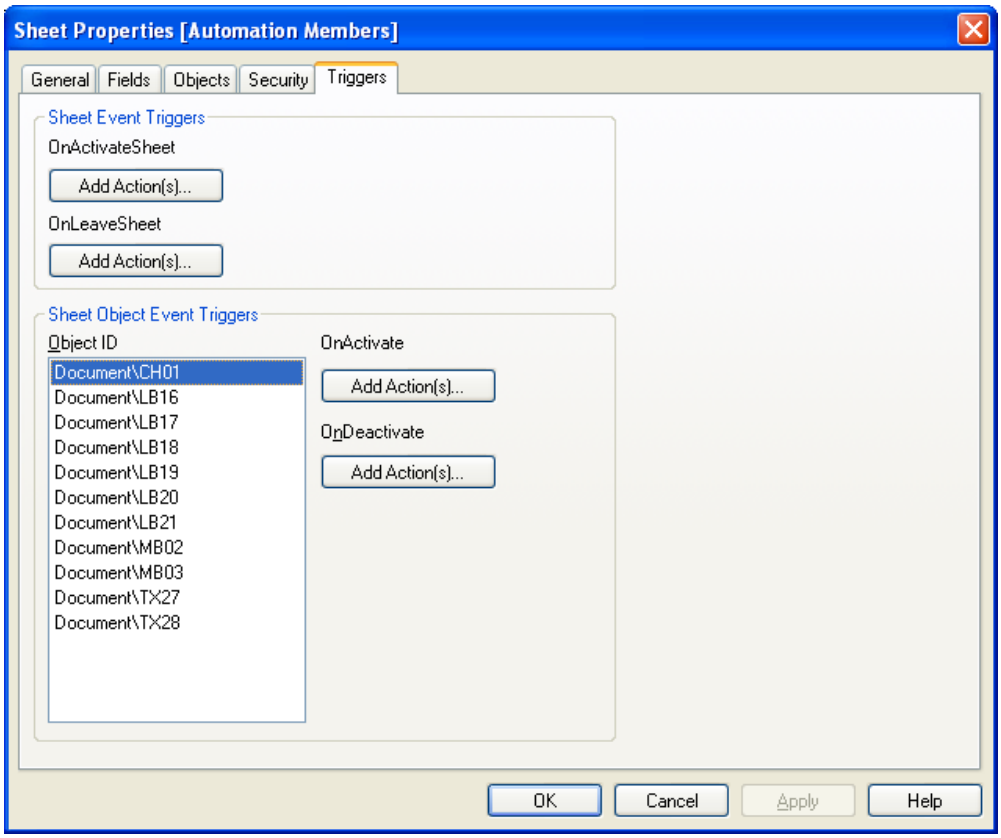
In this group you can set macros to trigger on changes in the contents of a specified variable in the document. You must first select a variable in the list before you may assign macros to events in it.

OnInput — A run macro action, if added, will be executed each time a new value is directly entered in the selected variable.

OnChange — A run macro action, if added, will be executed each time the value of the selected variable changes as a result of changes in other variables or the logical state of the application (typically applies when the variable contains a formula).

Invoking macros on sheet and sheet object events.

Choose the Triggers page under Sheet Properties.



Edit Module — By clicking on this button the Edit Module dialog is opened.

Sheet Event Triggers

In this group you can set macros to trigger on change of active sheet.

OnActivateSheet — In this dropdown list you can select from existing macro names or type any name which you later create a macro for in the Edit Module dialog. The macro, if it exists will be executed each time the sheet is activated.

OnLeaveSheet — In this dropdown list you can select from existing macro names or type any name which you later create a macro for in the Edit Module dialog. The macro, if it exists will be executed each time the sheet is deactivated.

Sheet Object Event Triggers

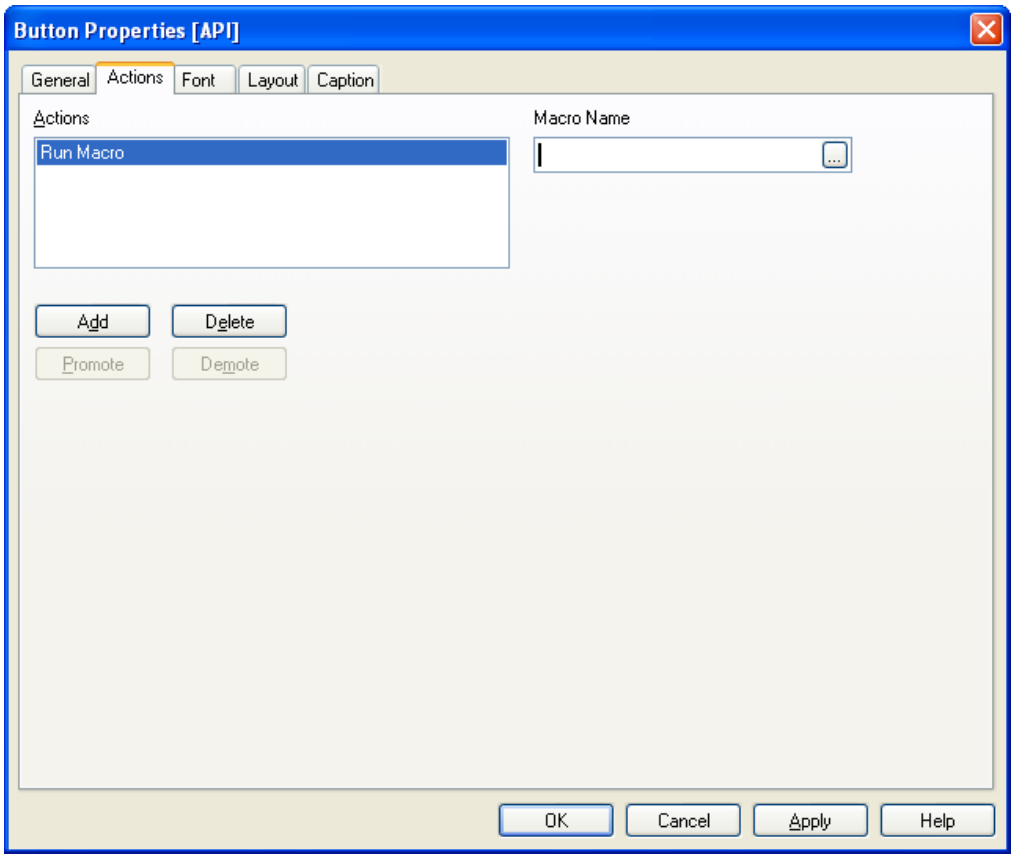
In this group you can set macros to trigger on activation and deactivation of a specified sheet object on the sheet. You must first select a sheet object in the Object ID list before you may assign macros to events in it.

OnActivate — In this dropdown list you can select from existing macro names or type any name which you later create a macro for in the Edit Module dialog. The macro, if it exists will be executed each time the sheet object is activated.

OnDeactivate — In this dropdown list you can select from existing macro names or type any name which you later create a macro for in the Edit Module dialog. The macro, if it exists will be executed each time the sheet object is deactivated.

Invoking macros with sheet object buttons

The Actions page of the Button Properties dialog becomes can be used to add a **Run** Macro Action



Macro Name — In this textbox you input any existing macro names or type any name which you later create a macro for in the Edit Module dialog. The macro, if it exists will be executed when the button is depressed.

2.6. Two types of classes

For the sake of clarity, the calsses in the QlikView Automation interface has been divided into two groups:

QlikView Object classes

QlikView Object classes are the fifteen classes which have a direct and obvious counterpart in a known QlikView entity visible through the User Interface. These are the program itself, the fields, the sheets and the various types of sheet objects.

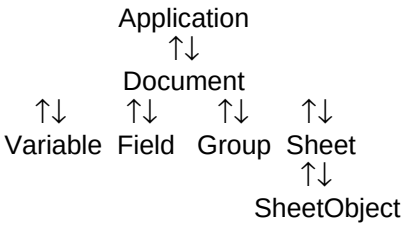
Interface Data Objects classes

Interface Data Objects are classes used to define various properties of the QlikView Object classes.

2.7. QlikView Object ownership hierarchy

The seven most basic classes in QlikView form a classical object hierarchy, where objects of a class are parents to children objects of another class. A child object cannot exist without its parent object, but does not inherit any of the parent's member methods or properties.

A schematic representation of the hierarchy could look as follows:

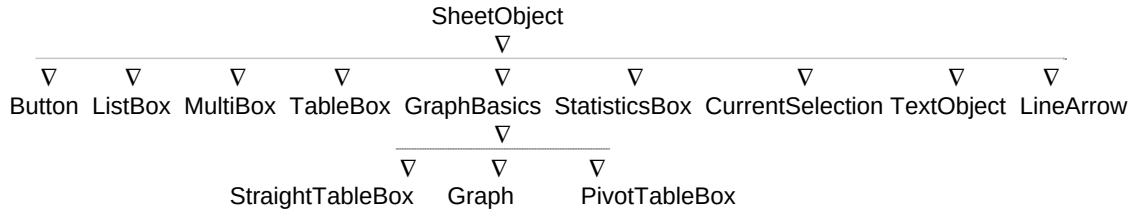


Legend : downward arrows point at children class
upward arrows point at parent class

2.8. QlikView Object inheritance hierarchy

The basic class for sheet objects contains methods common to all types of sheet objects. However, each of the sheet objects also need special methods, which only apply to them. Therefor all the classes related to sheet objects form a different type of hierarchy, where there are no parents an children, but rather a basic object and superset objects, which in addition to their own members inherit all the members of the basic object.

A schematic representation of this hierarchy could look as follows:



Legend : ∇ indicates that class below inherits all members from class above

2.9. Interface Data Object classes

The Interface Data Object classes all have a name beginning with a capital I. They typically appear as parameter objects or return objects in the methods belonging to the QlikView Object classes. They can also appear as parameter objects in the methods or as properties to other Interface Data Object classes.

3. The Visual Basic programming language

3.1. About Visual Basic

Visual Basic is a very suitable language for writing macros and other programs for controlling QlikView via it's Automation interface. Such code could of course also be written in other programming languages supporting Automation, e.g. Visual C++, but for most users Visual Basic in one of its forms is likely to be the first choice. For this reason a few words about the language could be appropriate in this documentation.

Visual Basic is a programming language with roots in the old BASIC language, known to almost everyone, who has ever programmed a computer. In comparison with its ancestor Visual Basic however has a number of additions which make it a quite modern object oriented programming language. Visual Basic has during the last few years turned into a macro and scripting language widely used by major software producers. It was therefor a natural choice to use the language (in its VBScript incarnation) as the macro language for QlikView.

As already mentioned above this document is not intended as a manual for Visual Basic. The reader is supposed to possess a basic knowledge of Visual Basic programming. However, so many examples are provided, that it should be possible for most readers to produce simple macros even without further study of Visual Basic.

3.2. Differences between VB, VBA and VBS

Visual Basic as a stand alone compiler (VB), Visual Basic for Applications (VBA) and Visual Basic Script (VBS) are all variations of the Visual Basic language. Most of the basic syntax and keywords are the same, but small variations exist. When it comes to advanced capabilities and the ability to communicate with the outside world from the code, there are large differences, especially between VB/VBA on one side and VBS on the other.

This documentation has a number of examples in the form of small pieces of Visual Basic code. They come both from the VB and the VBS environment. Sometimes, but not always there is a note indicating that the code works in one specific environment. The reader should be able to modify the code without a problem if a certain environment refuses to accept a certain syntax.

Some main differences:

	VB	VBA	VBScript
Compile stand-alone .exe	yes	no	no
Full VB syntax	yes	yes	no
Data types	full	full	limited

Different versions of VBScript

Apart from the incomplete listing of differences below, additional differences may exist as a result of which version of VBScript you are using. QlikView is curently shipped with VBScript version 3.1.

Some VB features not included in VBScript

Clipboard access
Collection
Conditional compilation
Constants
Data types (only variant supported)
Date/Time
DDE
Debug commands
Financial

Some features with limited implementation in VBScript

Array handling
Error handling (only On Error Resume Next)
File input/output

Some VB keywords and functions not supported by VBScript

Declare
Dim … as <datatype>
DoEvents
For Each … Next
GoSub … Return
GoTo
Line Numbers
Line Labels
Like
New
With … End With

3.3. Note about parameters to methods and properties

In VB, VBA and VBS the following rules apply to the use of parentheses around arguments:

- Always use parentheses around parameters to properties.
- Arguments to single parameter methods may optionally be enclosed in parentheses.
- Arguments to multiple parameter methods *must not* be enclosed in parentheses.

4. Getting hold of a QlikView document

4.1. Accessing QlikView documents from the internal VBScript interpreter

When using the internal VBScript interpreter, the only reference available is the ActiveDocument special property of the class Application. All references must be made starting from that point.

Note!
The ActiveDocument property from VBScript provides a reference to the QlikView document running the VBScript macro!

Examples:

```
rem ** VBScript Example 1 **
sub Clr
     set QVDoc = ActiveDocument
     QvDoc.ClearAll false
end sub


rem ** VBScript Example 2 **
sub a
     set up = ActiveDocument.GetApplication.GetUserPreferences
     Cp = up.ConfirmPurge
     if Cp=true then
         msgbox("Warning! No Reduce Data confirmation!")
     end if
end sub


rem ** VBScript Example 3 **
set doc1 = ActiveDocument
' ** ActiveDocument points at document running macro **
set App = doc1.GetApplication
set newdoc = App.OpenDoc ("C:\MyDocuments\QV4_Automation.qvw","","")
newdoc.Activate
' ** Note! newdoc is active but ActiveDocument points at doc1 **
```

4.2. Accessing QlikView documents from outside QlikView

QlikView documents can be accessed by means of the Automation interface from outside the QlikView program via e.g. Visual Basic (VB) or Visual Basic for Applications (VBA).

Before using QlikView Automation with typed calls you must include the QlikView type library (embedded in QV.EXE). In Visual Basic Studio this is done using the References command on the Project menu.

Note!
The ActiveDocument method used from VB provides a reference to the currently active QlikView document!

Examples (VB/VBA only! Do not work in VBScript !) :

```
rem ** VB/VBA Example 1 **
Private Sub OpenAndReload_Click()
     ' ** Open QV with named document **
     set QvDoc = GetObject("c:\MyDoc.qvw")
     QvDoc.Reload
End Sub

rem ** VB/VBA Example 2 **
Private Sub GetCurrentlyActiveDocument()
     dim ActiveDoc as new QlikView.ActiveDocument
     ' ** ActiveDoc points to active document in open QV. **
     ' ** The QlikView program must be started and have a **
     ' ** previously opened document.                      **
     …
End Sub

rem ** VB/VBA Example 3 **
Private Sub Ex2()
     Dim doc1 As New QlikView.ActiveDocument
     ' ** doc1 is set to active document in open QV **
     Set Appl = doc1.GetApplication
     ' ** Appl points at QV program                  **
     ' ** Appl.ActiveDocument points at doc1        **
     Set doc2 = Appl.OpenDoc("C:\MyDoc.qvw", "", "")
     doc2.Activate
     ' ** Appl.ActiveDocument now points at doc2    **
     ' ** Note difference from VBScript!            **
     …
End Sub
```

4.3. Untyped VB with QlikView Automation

Allthough typed references is quite convenient to work with and also creates faster code, it is sometimes desireable to use untyped code from VB (VBScript code is always untyped). The main reason is often to avoid the need for recompiling the VB code when the type library changes e.g. as the result of updated QlikView versions.

In order to achieve untyped code follow the steps below :
1) Do **not** include the QlikView type library in your project
2) Use the code shown below to get hold of QlikView
3) Do **not** use any statements such as **`dim … as QlikView.xxx`**

Example (Does not work in VBScript !) :

```
Private Sub UntypedCall()
     rem ** Create an untyped reference from VB **
     set Qv = CreateObject("QlikTech.QlikView")
     set ActiveDoc = Qv.ActiveDocument
     rem ** ActiveDoc can now be used largely as the **
     rem ** ActiveDocument reference in VBScript     **
     …
End Sub
```

5. Getting hold of the file system from VBScript

When using the internal VBScript interpreter, it is possible to get hold of a reference to the local file system by using a statement like the following :

```
set fso = CreateObject("Scripting.FileSystemObject")
```

This reference can then be used for operations towards the file system.

Example:

```
sub LogFunktion

  'This routine logs selection to a text file
  'Created by ICO

   set fso = CreateObject("Scripting.FileSystemObject")
   set mypath = ActiveDocument.GetProperties
   directory = mypath.MyWorkingDirectory

   On Error Resume Next

   ' See if file already exists.
   set filFile = fso.GetFile(directory & "log.txt")
   ' If not, then create it.
   if Err <> 0 then
      set filFile = fso.CreateTextFile(directory & "log.txt")
   end if

   set txsStream = filFile.OpenAsTextStream(8) 'For Appending

   set doc = ActiveDocument
   set mySelections = doc.fields("Field").GetSelectedValues

   for i = 0 to mySelections.Count - 1
      txsStream.WriteLine Now & " " & mySelections.Item(i).text
   next
   txsStream.WriteBlankLines 1
   txsStream.Close

end sub
```

6. VBScript function calls from script

Functions defined in the VBScript module of a QlikView application can be called from the script. If a function called is not recognized as a standard script function, a check will be made to see if a custom functions resides in the module. This gives you a large degree of freedom to define your own functions. The ActiveDocument Object cannot be used in functions called from the script. Using custom VBScript functions will of course be somewhat slower than executing the standard functions.

Examples:

```
rem ********************************************************
rem ********** THIS IS VBSCRIPT CODE FROM THE MODULE *********
rem ********************************************************

rem ******************** Global variables *******************
dim flag

rem *************** functions accessible from script *********

rem ********************** wrap for inputbox ******************
function VBin(prompt)
     VBin=inputbox(prompt)
end function

rem ********************** clear global flag ******************
function VBclearFlag()
     flag=0
end function

rem ***************** test if reference has passed ***********
function VBrelPos(Ref, Current)
     if Ref=Current then
           VBRelPos="Reference"
           flag=1
     elseif flag=0 then
           VBRelPos="Before "&Ref&" in table"
     else
           VBRelPos="After "&Ref&" in table"
     end if
end function


//  ********************************************************
//  ******************** THIS IS THE SCRIPT ******************
//  ********************************************************

let MaxPop=VBin('Max population in millions :');  // Ask limit
let RefCountry=VBin('Reference country :');       // Ask ref.
let dummy=VBclearFlag();            // Clears the global flag

Load
     Country,recno(),
     Capital,
     "Area(km.sq)",
     "Population(mio)",
     VBrelPos('$(RefCountry)',Country) as RelativePos
from country1.csv (ansi, txt, delimiter is ',', embedded labels)
where "Population(mio)" <= $(MaxPop);
```

Transfer of parameters

The following rules apply for parameter transfer of parameters between the load script and VBScript:

- Missing parameters are passed as NULL.
- If the actual expression evaluates to a valid number, the number is passed, else if the actual expression evaluates to a valid string, the string is passed, else NULL is passed.
- NULL is passed as VT_EMPTY.
- Return values are treated in a natural way.

7. Using Automation for Dynamic Data Update

Dynamic Data Update provides a mechanism for making transactions with the in-memory data of QlikView in real-time using syntax similar to SQL. The field data is updated in real-time without running the script.

Example:

```
Rem Dynamic Data Update

sub Update
  SET Result = ActiveDocument.DynamicUpdateCommand ("UPDATE * SET
Discount = if(Discount >= 35, 0, if (City='Stockholm', Discount + 5,
Discount + 2)) WHERE Country = 'SE'")
  if Result = false then
    MsgBox Result.ErrorMessage
  end if
end sub

sub Insert
  SET Result = ActiveDocument.DynamicUpdateCommand ("INSERT INTO *
(Country, City) VALUES (DK, Copenhagen), (NO, Oslo)")
  if Result = false then
    MsgBox Result.ErrorMessage
  end if
end sub

sub Delete
  SET Result = ActiveDocument.DynamicUpdateCommand ("DELETE FROM
CITY WHERE IsNull (Discount)")
  if Result = false then
    MsgBox Result.ErrorMessage
  end if
end sub
```