Qlikview Components (Qvc) Developer Guide

QVC is a collection of reusable script components for use by Qlikview Developers. The primary purpose of QVC is to speed development and provide a consistent model for building qlikview projects.

## Design Objectives

The design objectives for Qvc are:

1. Provide a consistent, stable and documented API.
2. Should be able to allow contribution by multiple distributed developers.
3. Must be able to grow, that is add new components, without losing clarity and quality.
4. Allow for continuous improvement and exploitation of new QV features.

Qvc is not meant to cover all scripting tasks and scenarios. Qvc is targeted at the “middle range” of scripting tasks.

Task Complexity

Qvc Target

At the bottom of the complexity stack are trivial simple tasks such as file loading. These tasks would not benefit enough from Qvc to justify using library routines. To support tasks at the high end would require increasing complexity in Qvc, possibly diminishing it’s quality and maintainability.

## Programming Conventions

### Structure:

The QVC source code is divided into “major” components to help make it easier to understand and maintain. The component files are named: Qvc\_*component*.qvs.

### Naming conventions:

* All names (except for local variables) should use capitalized full spellings with no spaces and no abbreviations. Examples:

BaseData  
 PrimaryKey

* Acronyms may be used but only the first letter should be capitalized. Examples:

Qvd  
 Html  
 QvdFilename

All externally visible API components; subroutines, functions, variables – will begin with the namespace prefix “Qvc.”

* Subroutines: Qvc.*function* Examples:

Qvc.Log  
Qvc.FileExists

Any member preceded by “\_” is private to Qvc. It should not be referenced by user script.

### Variable Names

#### Global Variables

Configuration and inter-subroutine communication is done using global variables. Variables should be named

Qvc.*component*.v.*name*

For example:

Qvc.Log.v.LogFilename

The *component* portion of the name generally matches the subroutine name. However, in some cases, a more general name is used when the variable is used by one or more subroutines.

Any variable preceded by “\_” is private to Qvc. It should not be referenced in user script.

Private global variables are used to communicate between routines or to maintain state between calls of the same routine. Private globals should use the Qvc.\* convention preceeded by “\_”. For example:

\_Qvc.Loader.v.QvdFile

Global variables must be deleted (nullified) in Qvc.Cleanup. Only variables definitely useful to the UI should persist.

#### Local Variables

Local variables may use any name but must be named with a leading “\_”For example:

\_maxval

Local variables ***must*** be deleted (nullified) at the end of the sub using a SET. For example:

SET \_maxval=;

(TODO is this good enough or should we introduce a reserved \_qvclocal.\* to not collide with user names?)

## Documentation

Generated documentation is created by using a “documentation block” within the Qvc subroutine or function. A documentation block begins with “/\*\*” (note two asterisks) and ends with “\*/”.

A documentation block immediately follows the script SUB statement and looks like this:

SUB Qvc.GetFieldValues( …)  
/\*\*  
@version $Id$  
Text description  
@tags  
\*/

The $Id$ will be expanded by SVN on first commit and should not be modified thereafter.

“Text Description” is the value that will be populated in the “Description” field of the generated documentation. Description may span multiple lines. Any line in the documentation block that is not a tag will be included in the description.

@tags are specific tags as follows. Tags should be completely contained on a single script line. There is currently no provision for a tag to span script lines.

**@syntax *example***A syntax example to invoke the member. Optional parameters should be enclosed in square brackets.

@syntax CALL Qvc.GetFieldValues('vStats', 'LastUpdate', ['Transactions.qvd']);

**@param *number description***Number is the ordinal parameter number starting with 1.   
Description explains the parameter. The type of the parameter should be given as the first word.

@param 1 String. Variable stem name in which to return values

**@var *variableName direction description***

variableName is the name of the variable.  
direction – is this an input (in) or an output (out) variable of this member?  
description explains the variable meaning and its default value. Allowable values should be enumerated.

@var Qvc.Calendar.v.CreateSetVariables in -1/0 (true/false) Should Calendar Set Analysis variables be created? Default is true.

Because Qvc ‘Functions” (QV variables with parameters) do not have a script statement delineating begin and end, the function tags are necessary to substitute for the SUB / END SUB script statements.

**@Function *functionName*@EndFunction**

The @Function tag should appear after “/\*\*” and the @EndFunction must be contained in its own “/\*” block.

For example:

/\*\*   
**@Function** Qvc.FileExists  
Returns true if a file exists. This function may only be used in script.  
@syntax LET vExists = $(Qvc.FileExists('dir\filename.ext'));  
@param 1 The relative or absolute file path as string.  
\*/  
The function code…  
/\*  
**@EndFunction**  
\*/

## Testing

In general, no subroutine or function should be created that cannot be tested with an automated test. This principle is critical to the quality and success of the project.

Test qvws must be created in the Tests directory. A test must be named as:

Test\_*membername*.qvw

The membername does not include the “Qvc.” prefix. For example:

Test\_GetFieldValues.qvw

The “Test\_” prefix is required by TestDriver.qvw. Matching the member name name ties the test qvw to the member name for organization and the test coverage report. The test qvw may use a non-matching name for the member name, but it must declare what the full name of the member it is testing via a call to Qvc.Test.RegisterTest.

Call Qvc.Test.RegisterTest(membername);

For example, Test\_Calendar also tests 'Qvc.CalendarFromField' and so contains:

Call Qvc.Test.RegisterTest('Qvc.CalendarFromField');

New test qvws should be created by copying $Test\_Template.qvw.. This qvw contains the sheet objects that report on the test results.

All tests should include both qvc.qvs and test runtime and qvc\_test.qvs. The template contains includes for both of these files.

At the end of the test script, all tests must call both:

CALL Qvc.Test.Cleanup;  
CALL Qvc.Cleanup;

The following routines are available for asserting expected test results.

CALL Qvc.Test.AssertEqual('msg',expected, actual);

CALL Qvc.Test.AssertTrue('msg',expr);

CALL Qvc.Test.AssertFalse('msg',expr);