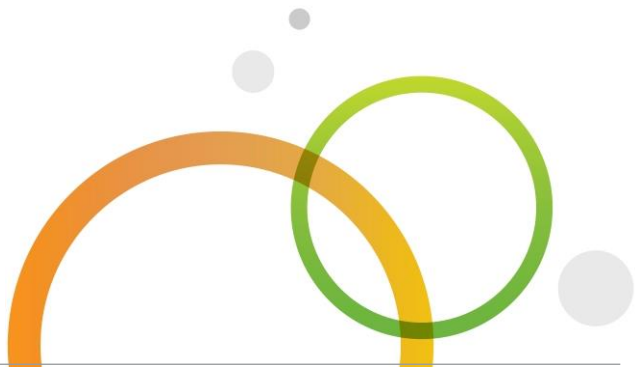
Qlik Deployment Framework

Function Reference Guide

February, 2017



qlik.com

Table of Contents

Function Reference Guide v1.7

Using a Sub function library during development is a part of the Build and Validation phase in the Qlik Application Development process, read more in ***Qlik Deployment Framework-Qlik Product Delivery process.pdf***

Container Initiation

Qlik Applications working in a container environment need to have an initiation string in the beginning of the Load Script to initiate QDF. The initiation lines are different between Qlik Sense and QlikView, but after initiation load scripts are identical between the two platforms. **Read more in the getting started guides**.

Qlik Sense Single Mount

For single mount add a *Sense folder connection (LIB)* with the name *Root* pointing to the framework starting point, add this line to the load script:

```
$(Include=lib://Root\InitLink.qvs);
```

QlikView

QlikView application must be stored under (preferably in a subfolder) under *1.Application* folder in a container. Paste in the initiation script code below into the first tab.

```
Let vL.InitLinkPath = Left(DocumentPath(), Index(DocumentPath(), '\1.Application'))&
'InitLink.qvs';
$(Must_Include=$(vL.InitLinkPath));
```

Clear Initiation Cache

To speed up initiation, a variable cache function has been introduced (v1.7 and later). The cache will validate the *Home* and *shared* container root path, if path is the same as last initiation then old global variables will be used. When the cache is used, these lines appear in log and output window.

```
'### QDF Info, Global Variables using cache'
'### QDF Info, Shared Global Variables using cache'
```

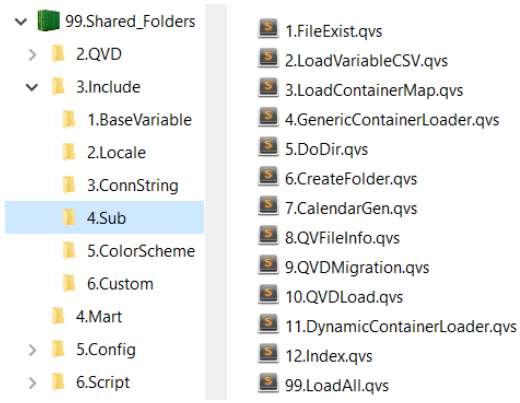
To override the cache function, add these lines before the initiation script.

```
set vG.BasePath=;
set vG.SharedBasePath=;
```

Sub Function Library

Qlik have the possibility of reusing scripts and functions by using the *Sub* and *Call* commands the Framework contains library of functions that is loaded in automatically during initiation.

Sub function library loaded during initiation is in Shared Folders container located under 99.Shared_Folders\3.Include\4.Sub.



Shared container is missing the local container sub function library will be used as a backup. Meaning as long as a Shared folder exists all sub function additions should be stored under the Shared container

Preinstalled Sub functions under *3.Include/4.Sub* folder should not be deleted or modified, this library is used by Qlik Deployment Framework initiation process.

99.LoadAll.qvs

The *99.LoadAll.qvs* include scrip calls in the sub function library last during QDF initiation (*1.InitLink.qvs)*. To automatic load in personal sub functions modify *99.LoadAll.qvs* in shared folders sub library as this is the default execution path. The global variable used in *99.LoadAll.qvs* when loading the sub function library is *vG.SharedSubPath* with *vG.SubPath*.as the backup (if shared folder missing).

Hint. Add additional sub functions, Qlik Community is a good place to look, instead of coding everything from scratch.

QlikView Components (QVC)

QlikView Components (QVC) is an open source library of reusable script that provides:

- Rapid script development
- Quality script development
- Implementation of best practices

QVC functionality consists of

- Calendars, AsOf
- Incremental (Delta) load
- Data modeling – Link Tables, Generic re-Join
- Utility functions – min/max values, loading variables
- Diagnostics & Logging

QVC is installed using the deploy tool and becomes an addition to the built-in QDF library, Qvc.qvs is initiated last in the *99.LoadAll.qvs* script. After QVC been installed it can be disabled in the script by adding this line before QDF initiation:
set vG.QVCDisable='true';

You can also remove QVC completely by unchecking the QVC option in the **Deploy Tool** or manually remove by deleting *Qvc.qvs* in both the Shared and active container.

For full QVC Documentation and Examples, download the qvc-nn.zip release package from
http://QlikViewComponents.org

Execute Functions

To execute a sub function use *Call function_name('Operator1','Operator2')*; A function statement must always end with a semicolon.

If for example operator one and three is used, operator two needs to be "blank" within quotas, example *Call function_name('Operator1','','Operator3')*;

Examples:

- vL.FileExist will return true of false depending on if the file exists
  Call vL.FileExist ('$(vG.QVDPath)\SOE.qvd');

- DoDir will list files on the file system into a table. Example below will list all files available in current container.
  Call DoDir('$(vG.BasePath)');

- *LoadVariableCSV loads in variables stored in csv files, in this case* all the HR tagged variables that is available in the Shared container. Note that the first operator need to be included but blank.
  Call LoadVariableCSV('','HR','Shared');

- *CALL Qvc.Routine(param1, param2, ...); To execute a* QVC function use *qvc.name* as seen here.

QDF Pre-Defined Functions

Below is description and syntax for all of the predefined Sub functions available in QDF, these functions are pre-loaded when running the initiation script.

FileExist

This sub function validates if a file or folder exists, can be used before load to avoid errors during script load.

Call vL.FileExist('Folder or file to validate');
Returns variable *vL.FileExist* with *true* or *false*.

Operator

- **Folder or file to validate** URL to folder of folders to validate wildcards is supported

Example

- *Call vL.FileExist ('$(vG.QVDPath)\1.NorthWind);*
  Will Check if*1.NorthWind* folder exists and return *vL.FileExist* = *true* or *false*

- *Call vL.FileExist ('$(vG.QVDPath)\SOE.qvd');*
  if vL.FileExist = 'false' then; trace '### Did not find file, exit script'; exit script; endif;

- Use * with caution as this could return a false true if a variable in the statement is missing for
  Call vL.FileExist ('$(NullVariable));*
  *Will return true as the function will search for * in the application location using relative path as $(NullVariable) returns null.*

LoadVariableCSV

SUB routine used for loading variables stored in csv files into the Qlik Script.
This file is used by *1.Init* to load Custom Global Variables.

Execute (Call) the Sub inside the script:

Call LoadVariableCSV('[My Variable File.csv]', ['Search Tag'], ['Container Prefix'], ['Comments as variables'] ['Container Map Mode']);

Operators:

- **My Variable File** Is the Variable File name to load, wild cards is possible. the function will by default try to find the variable file in *$(vG.BaseVariablePath)* (your container)
- **Search Tag** Is optional, will load variables based on tag's managed by the variable editor
- **Container Prefix** Is optional, will load variables from any container by using the prefix
- **Comments as variables** Is optional, will create a *_comment* variable for every real variable (if comments exist), this is nice way to add meta-data into expressions. *Comments as variables* can also be activated by setting the variable `SET vL.CommentsAsVariables=True;` before the 1.Init.qvs Initiation script.
- **Container Map Mode** is a special mode to create variables based on the Container Map, this is used internally by the Variable Editor.

Examples:
call LoadVariableCSV('')* Load all variables within my home container
call LoadVariableCSV('','HR','Shared') Load all the HR tagged variables stored in Shared container
call LoadVariableCSV('MyVariables','HR') Open *MyVariables* file and Load HR tagged variables
call LoadVariableCSV('MyVariables.csv','','','True') Load variables and Variable Comments
call LoadVariableCSV('','HR',AcmeHR') Load all HR tagged variables within AcmeHR container

LoadContainerGlobalVariables / LCGV

The LoadContainerGlobalVariables or LCGV function generates (mounts) Global Variable to other containers based on the Container Map. This function is intended to be used inside the Qlik scripts and is designed for easy use.

Call LCGV ('Container Prefix', [' Specific folder; Additional folders separated by ;],[Alias]);
Call LoadContainerGlobalVariables *('Container Prefix', [' Specific folder; Additional folders separated by ;] ,[Alias]);*

Operators:

- **Container Prefix** This is container prefix name (Tag) added when creating the container, in Qlik Sense this name is usually also the same as a valid folder connection
- **Specific folder** This is used to select folder/folders that should retrieve variables, these are separated by ";" without this setting all valid global variables will be fetched.
- **Alias** uses the added alias name instead of prefix in the generated global variables

Examples:

- Creates global variables to all valid resources in the AcmeTravel Container:
  Call LoadContainerGlobalVariables('AcmeTravel'); or
  Call LCGV('AcmeTravel');
  `Output:` *vG.AcmeTravelBasePath, vG.AcmeTravelQVDPath, vG.AcmeTravelIncludePath…*

- Load a single global variable, in this case Acme Travel QVD path *$(vG.AcmeTravelQVDPath).*
  Call LCGV('AcmeTravel','QVD');
  `Output:` *vG.AcmeTravelQVDPath*

- Generate several global path variables by use ';' as separator,
  in this case *vG.OracleQVDPath, vG.OracleIncludePath* variables will be created
  Call LCGV('Oracle', 'QVD;Include');
  `Output:` *vG.OracleQVDPath, vG.OracleIncludePath*

- Change name of generated global variables by using (3) Alias switch. This example will treat AcmeTravel as a *Shared* container, generating shared global variables
  Call LCGV('AcmeTravel','','Shared');
  `Output:` *vG.SharedBasePath, vG.SharedQVDPath, vG.SharedIncludePath…*

DoDir

DoDir is simple to use but powerful function that will index selected folder/file structure and return a table containing file name and path under selected file system. First include the script:

Call DoDir (Scan Path, [Table Name], [Folders Only], [Single Folder], [Qualified Felds], [Hide QDF Templates])

Operators:
Scan Path It the folder/file structure to scan
Table Name Is the Table name, optional default name is *DoDirFileList*
Folders Only Is an optional switch if set to *'true'* only folders will be returned
Single Folder Is an optional switch if set to *'true'* only one single folder will be indexed
Qualified Felds Is an optional switch if set to *'true'* all field named will be Qualified *based on the Table Name*
Hide QDF Templates Is an optional switch if set to *'true'* Template folders that starts with 0. Or incudes a # in QDF will be excluded. This switch is primarily used by the *DynamicContainerGlobalVariables* function.

Examples:

- *Call DoDir ('$(vG.IncludePath)'); Simple Example, returns all files under vG.IncludePath*
- *Call DoDir ('$(vG.IncludePath)*.qvs'); Will only return files with file type qvs under vG.IncludePath*
- *Call DoDir ('$(vG.IncludePath)', 'IncludeFileTable'); Change Table name to IncludeFileTable*
- *Call DoDir ('$(vG.IncludePath)', '', 'true'); Returns folder names only under vG.IncludePath*
- *Call DoDir ('$(vG.QVDPath)\HR.qvd'); Returns a line for this single file*
- *Call DoDir ('$(vG.QVDPath)', 'Tmp_Field', '' , '', 'true'); adds qualification on fields, example Tmp_Field.DoDirFileTime*

Output Table and fields:

DoDirFileName	FullyQualifiedName	DoDirFileSize	DoDirFileTime	DoDirContainerPath	DoDirFileExtension	DoDirFileN
desktop.ini	C:\QV-Docs\SourceDoc	46	3/2014 8:33:42 PM	desktop.ini	INI	
desktop.ini	C:\QV-Docs\SourceDoc	46	3/2014 8:33:41 PM	0.Template\1.Container	INI	
Folder.ico	C:\QV-Docs\SourceDoc	36870	3/2014 8:33:42 PM	Folder.ico	ICO	
Folder.ico	C:\QV-Docs\SourceDoc	99462	3/2014 8:33:41 PM	0.Template\1.Container	ICO	
Info.txt	C:\QV-Docs\SourceDoc	418	3/2014 8:33:42 PM	Info.txt	TXT	
Info.txt	C:\QV-Docs\SourceDoc	420	3/2014 8:33:41 PM	0.Template\Info.txt	TXT	
Info.txt	C:\QV-Docs\SourceDoc	418	3/2014 8:33:41 PM	0.Template\1.Container	TXT	
Info.txt	C:\QV-Docs\SourceDoc	438	3/2014 8:33:41 PM	0.Template\1.Container	TXT	
Info.txt	C:\QV-Docs\SourceDoc	455	3/2014 8:33:41 PM	0.Template\1.Container	TXT	
Info.txt	C:\QV-Docs\SourceDoc	432	3/2014 8:33:41 PM	0.Template\1.Container	TXT	

- ***DoDirFileName*** is the File Name without path
- ***FullyQualifiedName*** is the file name and complete path
- ***DoDirFileSize*** is the file size
- ***DoDirFileTime*** is file date and time
- ***DoDirContainerPath*** lists the files in relationship with the current container
- ***DoDirFileExtension*** Contains the File Extension in upper case, perfect to use when searching for types
- ***DoDirFileNameCount*** Counts file name (DoDirFileName) duplications.

CreateFolder

Create Folder function will -as the name says- create a folder (if non existing) or a folder structure.

sub *CreateFolder (vL.FolderName)*

Operators:
vL.FolderName Is the folder name or folder structure to create

Examples:

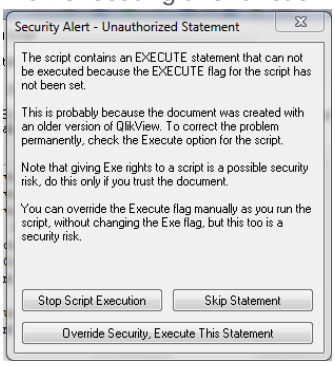
- *Call CreateFolder(' $(vG.QVDPath)1.Northwind'); Will create 1.Northwind folder under vG.QVDPath*
- *Cal CreateFolder (' $(vG.QVDPath)1.Extract\1.Northwind'); Will create 1.Northwind folder under vG.QVDPath\1.Extract*

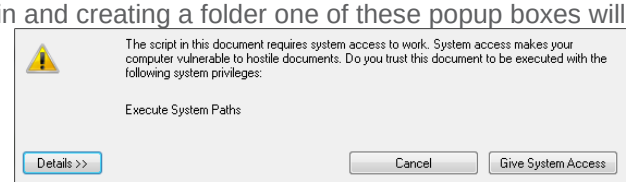
Qlik Sense

CreateFolder works with Qlik Sense 3.2 and later.

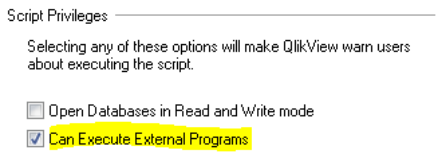
QlikView Developer

When executing this function in and creating a folder one of these popup boxes will appear:





Press Override Security to execute the folder creation, next run the folders are already created and the box will not return. Recommendation is to activate *Can Execute External Programs* this will also allow Publisher to run the function.



CalendarGen

Master calendar function created by *Jonas Valleskog*, enhancements added by Qlik. Generic calendar enables scalable handling of creating and navigating multiple date fields. Calendar Gen also have native Qlik Sense calendar support, meaning that calendar will be dynamic and selectable within Qlik Sense.

CALL CalendarGen('Date Field',['Calendar Table'] [,'Months Left Fiscal Date'] [,'Min Date', 'Max Date'][,'Link Table'][,'DateFormat'];

Operators:

- **Date Field** is the date field to link calendar. Generated Calendars are based on the added fields as multiple fields are supported (separated by ,) a master calendar will be created for each entry.
- **Calendar Table** (Optional) is the master calendar table name default is the same name as Date Field
- **Months Left Fiscal Date** (Optional) to activate Fiscal Dates, set no of months left of the Calendar year the month the Fiscal year begins. E g '3' if the first month of the Fiscal year is October, -3 negative fiscal year
- **Min Date (Optional)** Set hard Minimum calendar date ex. '11/7/1996' (depending on locale settings)
- **Max Date** (Optional) Set hard Maximum calendar date ex.'8/13/1999'(depending on locale settings)
- **Link Table** (Optional) By default link table is identified based on Date Field use this setting if need to override
- **Date Format** (Optional) Sets date format, default is to use environmental *$(DateFormat)* variable, multiple date formats are supported (separated by ,).

Examples:

- *Creates a OrderDate master calendar and auto determin min and max date*
  Call CalendarGen('OrderDate');
- *Create master calendars for both OrderDate and ShippingDate*
  Call CalendarGen('OrderDate,ShippingDate');
- *Fiscal Dates and rename to OrderDateCalendar*
  Call CalendarGen('OrderDate','OrderDateCalendar' ,'3');
- *Min and Max date*
  Call CalendarGen('OrderDate','' ,'','11/7/1996','8/13/1999');

The sub function will return table with the standard fields below:

- *Table Name* **– The *Date Field*** table name used as key field to data model
- *Table Name* **Week –** Week number field Ex. 32,33,34
- *Table Name* **Year** – Year field Ex. 2001, 2002
- *Table Name* **Month** – Month field Ex. Jul, Aug
- *Table Name* **Day** – Day number field Ex. 1,2,3,4
- *Table Name* **WeekDay –** Weekday short name field Ex. Mon, Tue, Wen
- *Table Name* **Quarter –** Quarter field Ex Q1, Q2, Q3, Q4
- *Table Name* **MonthYear –** Concatenated month and year field Ex. 08-2002, 09-2002
- *Table Name* **QuarterYear–** Concatenated quarter year field Ex. Q3-2002, Q4-2002
- *Table Name* **WeekYear–** Concatenated week year field Ex. 32-2002, 33-2002
- *Table Name* **YTD Flag –** Year to Date Flag field shows 1 if current year
- *Table Name* **PYTD Flag -** Past Year to Date flag field shows 1 if last year
- *Table Name* **CurrentMonth Flag–** Current Month flag shows 1 if historical month is same as current month
- *Table Name* **LastMonth Flag-** Last Month flag shows 1 if historical month is same as last month
- **num** *Table Name*- Autonumber field based on rows ex. 1,2,3,4,5,6…700,701,702
- *Table Name* **numMonthYear** – Autonumber field based on MonthYear field ex. 2, 28, 59, 89
- *Table Name* **numQuarterYear** – Autonumber field based on QuaterYear field ex. 2, 89, 181
- *Table Name* **numWeekYear** – Autonumber field based on WeekYear field ex. 2, 4, 11, 18, 25

Calendar Tips and tricks:

- Check out *6.Calendar-Example* to get inspiration. Copy or re-create the calendar objects (time related list boxes) laid out in the front-end of the example application.
- Use *DateFormat* variable when formatting date, this creates flexibility when changing locale.
  ex. Date(OrderDate,'$(DateFormat)') AS OrderDate
- To avoid potentially slow queries against large in-memory tables, contemplate storing out the date field to QVD first and use the QVD store as the input source to the MinMax: table creation.
- If gaps in calendars for missing dates are not an issue, consider replacing AUTOGENERATE() logic for generating the calendar table with a distinct list of each date seen in the source table instead.

QVFileInfo

QvFileInfo sub function returns information (in table format) regarding Qlik files that stores metadata at the moment QVW and QVD file formats (qvf files are not supported).

Call QVFileInfo('Fully Qualified file Name',['Table Name'])

Operators:

- **Fully Qualified file Name** is the path and name of qvd or qvw file.
- **Table Name (Optional)** is name of the table returning the result default table name is *QVFileInfo* linked with *QVFileInfo_field* (field details table)

Examples:

- Will extract meta-data for Customer.qvd
  Call QVFileInfo('$(vG.QVDPath)\Customer.qvd')
- Will extract meta-data for Customer.qvd to table QVFileTable
  Call QVFileInfo('$(vG.QVDPath)\Customer.qvd','QVFileTable')
- Below example combines QVFileInfo and DoDir to index the Qlik files and use *FullyQualifiedName* field as link to the QVFileInfoTable

```
Call DoDir('$(vG.BasePath)');
for vL.LoopDoDirRows = 1 to NoOfRows('DoDirFileList')
    LET vL.FullyQualifiedName = peek('FullyQualifiedName',$(vL.LoopDoDirRows),'DoDirFileList');
Call QVFileInfo ('$(vL.FullyQualifiedName)');
next
```

Returned table *QVFileInfo* contains below meta-data:

- ***FullyQualifiedName*** is the file name and complete path, use as link to DoDir Table
- ***QVTablesKey*** Table link key to *QVFileInfo_Fields* table
- ***QVTableName*** Name of tables in an QVW file or name of Table in a QVD file
- ***QVFileTime*** Data reload date
- ***QVTableNbrRows*** Total number of rows in *QVTableName*
- ***QVTableNbrFields*** Total number of fields in *QVTableName*
- ***QVTableNbrKeyFields*** Total number of Key fields in *QVTableName* only used by QVW files
- ***QVTableComment*** Table Comments, only used by QVW files
- **QVFileName** Name of the qvd file
- **QVTableCreator** Name of qvf or qvw application that created the qvd

QVFileInfo_Fields is a help table, containing Field information regarding QVD and QVW files:

- ***QVTablesKey*** Table link key to *QVFileInfo* table
- ***QVFieldName*** Name of Fields in a Table
- ***QVFieldComment*** Field Comments, only used by QVW files

QVDMigration

QVDMigration sub function migrates and consolidates qvd data between containers, using fixed file names or wildcard (*) migrating a qvd folder in one single statement. QVDMigration can optionally migrate selected fields and scramble fields if needed. The sub function is primarily designed for data migration into a self-service (sandbox) environment. Needed subfolders in destination path will automatically be created by use of *CreateFolders* function.

Call QVDMigration (QVD Source File, QVD Destination File, [Select specific fields (, separator) leave blank for all fields], [Scrambled fields (, separator)], [Table Name Suffix], [Include Subfolders], [Format-Spec], [No of Records]);

Operators:

- **QVD Source** File is the QVD source file or folder
- **QVD Destination** File is QVD destination path. Optionaly, to rename file add filename
- **Fields to select** (Optional) used when selecting specific fields from the Source QVD. Multiple fields are separated with (,).
- **Scrambled fields** (Optional) used when scrambling fields from the Source QVD. Multiple fields are separated with (,). Scramble overrides Fields to select parameter if dual entries found. Scrambling will have performance impact so carefully select fields to scramble.
- **Table Name Suffix** (Optional) primarily used as meta-data separator between source and destination this by adding a suffix on the destination qvd table names. The difference will be exposed in Governance Dashboard as shown below.

TableName	QVD\QVX
Customer	C:\QV-Docs\SourceDocs\1.Production\0.Administration\2.QVD\Customer.qvd
	C:\QV-Docs\SourceDocs\1.Production\1.AcmeSales\2.QVD\Customer.qvd
Customer-Shared	C:\QV-Docs\SourceDocs\1.Production\99.Shared_folders\2.QVD\Customer.qvd

Separating Table Name (Meta Data) by using Table Name Suffix, shown in Governance Dashboard

- **Include Subfolders** (Optional) If set to true subfolders under Source Files will also be migrated, needed subfolders in destination path will automatically be created by use of CreateFolders function
- **Format-Spec** (Optional) export to other formats than qvd, options are txt or qvx
- **No of Records** (Optional) limit number of records migrated

Examples:

- Migrate Customer.qvd to shared QVD folder without any manipulation
  Call QVDMigration ('$(vG.QVDPath)\Customer.qvd','$(vG.SharedQVDPath)');
- Migrate Customer.qvd to shared QVD folder and changing name to Customer_new.qvd
  Call QVDMigration ('$(vG.QVDPath)\Customer.qvd','$(vG.SharedQVDPath)\Customer_new.qvd');
- Migrate fields CustomeID and CompanyName in all Customer*.qvd files to shared QVD folder
  Call QVDMigration
  ('$(vG.QVDPath)\Customer*.qvd','$(vG.SharedQVDPath)','CustomeID,CompanyName');
- Migrate fields CustomeID and CompanyName in Customer.qvd to shared QVD folder scramble CustomerID field
  Call QVDMigration ('$(vG.QVDPath)\Customer.qvd','$(vG.SharedQVDPath)',
  'CustomerID,CompanyName','CustomerID');
- Migrate all Customer qvd files to shared QVD folder, scrambling CustomerID field in all the qvd's
  Call QVDMigration
  ('$(vG.QVDPath)\Customer*.qvd','$(vG.SharedQVDPath)\Customer.qvd','','CustomerID');

QVDLoad

QVDLoad will load up qvd files into a data model based on the meta-data headers in the qvd files. Also qvd files stored in subfolders can optional be loaded. QVDLoad is based on *QVDMigration* and have the same code and switches except for destination path.

Call QVDLoad(QVD Repository, [Select specific fields (, separator) leave blank for all fields], [Scrambled fields (, separator)], [Table Name Suffix], [Include Subfolders], [No of Records]);

Operators:

- **QVD Repository** is the QVD source file or folder storage
- **Fields to select** (Optional) used when selecting specific fields from Repository. Multiple fields are separated with (,).
- **Scrambled fields** (Optional) used when scrambling fields from Repository into the application. Multiple fields are separated with (,). Scramble overrides Fields to select parameter if dual entries found. Scrambling will have performance impact so carefully select fields to scramble.
- **Table Name Suffix** (Optional) will add a suffix on all tables in the data model
- **Include Subfolders** (Optional) If set to true qvd files in subfolders will also be loaded
- **No of Records** (Optional) limit number of records loaded

Examples:

- Load in all qvd files in *vG.QVDPath* folder and create a data-model based on table headers
  Call QVDLoad ('$(vG.QVDPath)');

- Load in all qvd files stored in every subfolder under *vG.QVDPath*
  Call QVDLoad ('$(vG.QVDPath)','','','','true');

- Load in fields *CustomeID* and *CompanyName* in all qvd files.
  Call QVDLoad ('$(vG.QVDPath)','CustomeID,CompanyName');

- Loads fields *CustomeID* and *CompanyName* and scramble *CustomerID* field from *Customer.qvd*
  Call QVDLoad ('$(vG.QVDPath)\Customer.qvd', 'CustomerID,CompanyName','CustomerID');

DynamicContainerGlobalVariables / DCGV

DynamicContainerGlobalVariables or DCGV "mounts" folders and creates GlobaLVariable links in similar way to LCGV. The difference is that DCGV's global variables are created from folders names within other containers instead of using the container folder mapping table (stored in 1.Init.qvs). DCGV can also be used to create "Global Variable mounts" to optional folder structures and not only within a container. Use-case is when converting to a QDF environment and "mask" the old environment as a container.

Call DCGV*('Container Name or URL', [' Specific Folder [; Additional folders separated by ;']] , ['Override Prefix']);*

Operators:

- ***Container Name or URL (Optional)*** Container prefix name (identifier) or a URL to folder structure to retrieve global path variables under. If empty current home container will be used.
- ***Specific Folder (Optional)*** This is used to select specific folder/folders that should retrieve *Global Variables*. Add additional folders separated by ;
- ***Override Prefix (Optional)*** Uses fixed *Global Variable* prefix in variable names instead of the container name. This is neat to use when pointing *Container Path Name* to a file structure or containers not included in the map.

There are some naming conventions needed to be obliged to identify and create global variables within a container, this to keep the amount of potential global variables down to a minimum. The below conventions are not used when ***Container Path Name*** and ***Specific Folder*** are used in combination.

- Only folders containing an **initial number** will be identified as a Container folder and get a variable.
  Example: Folder 1.Extract will get the global variable vG.xxxExtractPath where xxx is the container prefix name if plausible.

- Folders with a starting *0.* are discarded as this marks template folders.
  Example: Folder 0.Templates will not get a correlating global path variable

- The global variable name is the name (without space) between first and second dot (.) or between first dot and file name end if there's only one dot. Text after the second dot is treated as descriptive information.
  Example:
  Folder 1.Extract.QVDFiles will have the global variable vG.xxxExtractPath
  Folder 1.Extract will have the global variable vG.xxxExtractPath
  Folder 1.Extract QVDFiles will have the global variable vG.xxxExtractQVDFilesPath (space is removed)

Examples:

- Load all global path variables to the *AcmeTravel* Container:
  call DCGV('AcmeTravel');

- Load a single global path variable, in this case linking to a folder named 7.Data in the AcmeTravel container. The Global Variable becomes *(vG.AcmeTravelDataPath)*
  call DCGV('AcmeTravel','Data');

- Load several global path variables by use ';' as separator,
  in this case *vG.OracleDataPath,* and *vG.OracleApplicationPath* variables will be created
  call DCGV('Oracle','Data;Application');

- Retrieve global path variables from folders under c:\temp that have *Data* in the name will create the Global Variable *vG.DataPath* pointing to *c:\temp\Data* folder
  QlikView: *call DCGV('c:\temp', 'Data');*
  Qlik Sense: *call DCGV('lib://MyStuff','Data');*

- Add the optional Prefix name *Oracle* on the above example will create the Global Variable *vG.OracleDataPath*
  call DCGV('c:\temp', 'Data', 'Oracle');

Index Functions

Index is functions that creates and maintains a set of indexes for Qlik Data files (QVD). These indexes are used when searching for data types across multiple qvd files this means that developers and power users select needed data using a simple command. Finding the data is done autonomously by the system in the background. The index is stored in one single location, under *vG.SharedConfigPath* while the qvd's can be spread out across the environment depending on security or organizational considerations.
There are three index functions implemented:

- **IndexAdd** Will create the QVD indexes, should be done during qvd creation.
- **IndexLoad** Loads Qlik data based on combination of index criteria's like file name, tags, table, fiels…
- **IndexDel** Delete index and optionally referring qvd file (need legacy mode activated).

Use the *Index Monitor* application under *0.Administration/3.IndexMonitor* to monitor Indexes and QVD files.

IndexAdd

IndexAdd creates a QVD index. The index is based on meta-data and tags collected from the QVD header. The index can thereby be recreated if need be. The index default location is $(*vG.SharedConfigPath)/Index* location can be modified if needed.

Call IndexAdd(['QVD path and name'],['Index folder name'] ,['Container name'] ,['Tags'] ,['Alternative Index path'])

Operators:

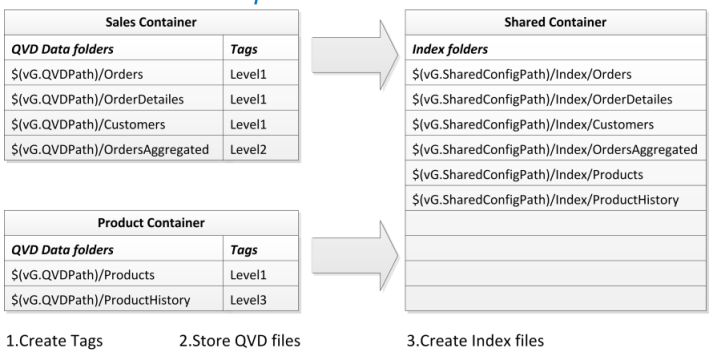
- ***QVD path and name*** (Optional) Path and name of QVD file to Index, wild card (*) is supported
- ***Index folder name*** (Optional) Place the Index in a specific folder, default is to use qvd folder name
- ***Container name*** (Optional) Specify the QVD files container name, this is usually identified automatically
- ***Tags*** (Optional) Add index tag, recommendation is to use the *comment table* function instead as this will be more persistent.
- ***Alternative Index path*** (Optional) will change the default Index path *($(vG.SharedConfigPath)/Index)* This is not recommended as all functions would need the alternative path specified

Examples:

Call IndexAdd('vG.QVDPath\Customers.qvd'); Will add an index for *Customers*.qvd file
Call IndexAdd('vG.QVDPath.qvd');* Will add an index for all qvd files in *vG.QVDPath* path

Using Comment field to tag QVD data

It is strongly recommended to add tags into your Qlik data files, these tags will identify what data to load. For example, if we create an aggregated QVD data layer it could be tagged -for example- as *Level2* (where level 1 is un-aggregated). This is done by adding tag (or tags) into the qvd meta-data header using the *Comment Table* function. This should be done before storing the QVD, creating the index using *IndexAdd* should be done after *Store into qvd* command.



Example:

```
Comment Table [$(vL.TableName)] with 'Level2';
Store [$(vL.TableName)] into '$(vG.QVDPath)\Folder\$(vL.TableName).qvd';
Call IndexAdd('$(vG.QVDPath)\Folder\$(vL.TableName).qvd');
```

Several tags can be added using comma (,) as separator as shown below:

```
Comment Table [$(vL.TableName)] with 'Level2,SalesAgg';
```

QVD naming conventions

When using the Index functionality, it's important to have a god qvd naming convention as *IndexLoad* function will use the qvd name as primary search criteria, example:

$(vG.QVDPath)\Orders\01-02-2015.qvd [Table Name]\[Day]-[Month]-[Year]

In this case orders are stored according to date, and the QVD folder is the same as table name. The date and folder name can be used during search.

Security requirements
Orders folder will be automatically created in the Index by using the *CreateFolder* function, for this the qvw file should have ***Can Execute External Programs*** switch checked.

Qlik Sense limitations
Qlik Sense Native mode does not allow the execution command, thereby creation of index folders will not work (Indexing of Qlik qvd files works in Sense). Qlik Sense qvd files is in a different format and have no meta-tags just yet, so *IndexAdd* function will not work against this qvd format.

Index fields
IndexAdd will create a tiny index file for every qvd file. The index file contains descriptive meta-data of the Qlik Data File (QVD). Almost all the index fields are searchable when using the *IndexLoad* function. Below is a list of the Index fields:

- ***QVDFileName*** Name of the added qvd file
- ***QVTableName*** Table name
- ***QVDSourcePath*** URL to the qvd file (only used as backup for Container + relative path)
- ***QVDSourceContainerName*** Name of qvd container (used when retrieving the qvd)
- ***RelativePath*** Relatively where in the container is the qvd stored
- ***QVDTag*** Tags that is referred to this qvd (created by *Comment Table*)
- ***QVDIndexStorageName*** Index storage sub folder name (usually the same as qvd folder name)
- ***QVDTimestamp*** Time of qvd creation taken from meta-data (not the same as file time).
  Convert timestamp to date: *Let vL.Date = timestamp(**QVDTimestamp**,'$(DateFormat)');*
- **QVDFields** Field that the qvd contains
- **QVDTableCreator** qvd creator file name (qvw)
- **QVDNbrRecords** Number of records the qvd contains
- **QVDNbrFields** Number of fields the qvd contains

IndexLoad

IndexLoad loads qvd data based on index search criteria's like tags and field names. The qvd fieldname is the primary search criteria so it's strongly recommended to have a qvd naming convention like *day-month-year-TableName.qvd*.

Call IndexLoad(['QVD file name'],['Table Name'] ,['Index Folder Name'] ,['Tags'] ,['Fields'] ,['Index Only'] ,['Load Max Rows'] ,[['Alternative Index path'],['Load Expressions'])

Operators:

- ***QVD file name*** (Optional) Name of QVD to load, wild cards (*01-2015*) is supported
- ***Table Name*** (Optional) Load in a table, can be combined with ***QVD file name***
- ***Index Folder Name*** (Optional) use this specific index only, can be combined with ***QVD file name***
- ***Tags*** (Optional) load data containing a specific tag, can be combined with ***QVD file name***
- ***Fields*** (Optional) load selected fields separated by comma (,) can be combined with ***QVD file name***
- ***Index Only*** (Optional) will only load in the Index, true will use default table name *(vL._QVDIndexTable).* Type table name from default *vL._QVDIndexTable*. This is used when developing apps where the Index is needed.
- **LoadMaxRows** (Optional) will limit how many rows that can be loaded. This will only stop sequential QVD file to load a big QVD will probably load above this limit.
- ***Alternative Index path*** (Optional) will change the default Index path *($(vG.SharedConfigPath)/Index)* This is not recommended as all functions would need the alternative path specified
- **Load Expressions** (Optional) If set to true it will load related expressions based on tags and container collected from Index, this operator calls the *LoadVariableCSV* function.

Examples:

Call IndexLoad('2014'); Load all qvd files that contains the name 2014
Call IndexLoad('','Customers'); *Load all qvd files that contains the customer table*
Call IndexLoad('', '','Orders'); *Load all qvd's in the Orders Index*
Call IndexLoad('','','','Level2'); *Load all Level2 tagged qvd files*
Call IndexLoad('','','','','CustomerID,ContactName,Fax'); *Load all QVD's that contains these three fields*
Call IndexLoad('','','','','','IndexTable'); *Will not load any data, just the Index table as IndexTable*
Call IndexLoad('', '', '', '', '', '', '30000'); *Load all qvd's until reaching about 30000 records*

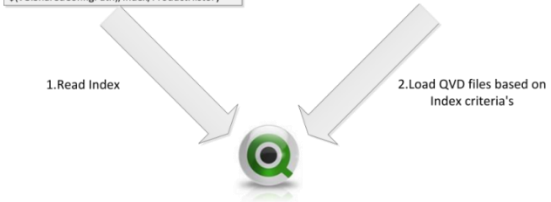
These different criteria's can be combined, Examples:

Call IndexLoad('2014','Customers','','','CustomerID,ContactName,Fax','30000'); Loads year 2014 containing the table
Customers and fields *[CustomerID,ContactName,Fax]* also we should not load more than 30000 records.

Call IndexLoad('2014','','','Level2'); *Will l*oad level 2 data from year 2014

Shared Container	
Index folders	
$(vG.SharedConfigPath)/Index/Orders	
$(vG.SharedConfigPath)/Index/OrderDetailes	
$(vG.SharedConfigPath)/Index/Customers	
$(vG.SharedConfigPath)/Index/OrdersAggregated	
$(vG.SharedConfigPath)/Index/Products	
$(vG.SharedConfigPath)/Index/ProductHistory	

Sales Container	
QVD Data folders	***Tags***
$(vG.QVDPath)/Orders	Level1
$(vG.QVDPath)/OrderDetailes	Level1
$(vG.QVDPath)/Customers	Level1
$(vG.QVDPath)/OrdersAggregated	Level2



IndexDel

IndexDel delete indexes and optionally associated qvd files. This could be needed keeping a consistent qvd strategy. Example, Storing 24 month of history indexes and qvd files older than 24 month should be removed else history will keep on growing infinite. *IndexDel* will search for the qvd fieldname (from the index) so it's strongly recommended to have a qvd file naming convention, like *TableName\day-month-year-TableName.qvd.*

Call IndexDel('Index file name',['Delete associated QVD files'] ,['Index Folder Name'] ,['Alternative Index path'])

Operators:

- ***Index file name*** Name of index to delete, wild cards (*) is supported
- ***Delete associated QVD files*** (Optional) if *true* qvd files associated to the indexes will also be deleted
- ***Index Folder Name*** (Optional) use this specific index only, can be combined with ***Index file name***
- ***Alternative Index path*** (Optional) will change the default Index path *($(vG.SharedConfigPath)/Index)* This is not recommended as all functions would need the alternative path specified

Call IndexDel('2011-12');* Deletes all Index files that starts with the name *2011-12*
Call IndexDel('2011-12','true');* deletes all index and associated qvd files that starts with the name *2011-12*

1.Delete Index

Shared Container
Index folders
$(vG.SharedConfigPath)/Index/Orders
$(vG.SharedConfigPath)/Index/OrderDetailes
$(vG.SharedConfigPath)/Index/Customers
$(vG.SharedConfigPath)/Index/OrdersAggregated
$(vG.SharedConfigPath)/Index/Products
$(vG.SharedConfigPath)/Index/ProductHistory

2. Optionally, delete associated qvd files

Sales Container	
QVD Data folders	***Tags***
$(vG.QVDPath)/Orders	Level1
$(vG.QVDPath)/OrderDetailes	Level1
$(vG.QVDPath)/Customers	Level1
$(vG.QVDPath)/OrdersAggregated	Level2

Qlik Sense limitation

Qlik Sense Native mode does not allow the execution command thereby *IndexDel* will not work in Sense unless legacy mode is enabled.

QlikView security requirements

IndexDel uses the execution command in Qlik to delete files, for this the qvw file need to have ***Can Execute External Programs*** switch checked in.

Internal Functions

Below are functions used internally by the framework, these could be accessed in the script but usually not needed.

LoadContainerMap

LoadContainerMap function is used for loading in the Container Map csv file and return information for a specific container. This function is mainly used internally by other functions like LCGV to create Global Variable links (mount) to other containers

sub LoadContainerMap('Container Map file', 'Container name', [' Optional $(vG.BasePath)']);

Operators:

- **Container Map file** Is the name and path of the container map that should be validated
- **Container name** Is the prefix name of the container that should be validated
- **$(vG.BasePath) Optional** specially designed to identify *Root Path (vG.RootPath)*. This is done by opening the container map, validate where I am and the Root Path in relation to my container. The operator must be global variable base path (*vG.BasePath)*. If this process fails, the Root Path will be set to one folder above your container. When using this switch the *Container name* parameter is not used.

These variables are returned by LoadContainerMap:

- **vL.ContainerFolderName** This is the Container folder name
  vL.ContainerPathName This is the Container prefix name
- **vL.Comment** Comments regarding the container
  vL.LoadContainerMapCount Returns a result (Variable prefix name) only if Variable prefix duplication found. This so that Variable Editor can alert operator to remove duplication.
- **vL.RootPath** (Optional) returns the Global Variable *vG.RootPath*