



Qlik Deployment Framework

QlikView Development Guide

April, 2017





Table of Contents

| | |
|--|-----------|
| Why the need for a Development Guide? | 4 |
| Company Specific Development Guide | 4 |
| Platform strategy from a development perspective | 4 |
| Standards | 5 |
| Qlik Deployment Framework (QDF) Containers | 5 |
| QDF Deploy Tool | 5 |
| Container Type Selector | 5 |
| Folder Naming Convention | 6 |
| Container Variables | 7 |
| Global Variables | 8 |
| Custom Global Variables and Universal variables | 8 |
| Using Deployment Framework Containers when scripting | 8 |
| Get started with QDF in QlikView Developer | 8 |
| Getting Started Guide | 9 |
| Additional Naming Convention | 9 |
| Back End Development | 10 |
| Scripting basics | 10 |
| Linking (mount) Containers | 10 |
| Data strategy | 11 |
| Staging data | 11 |
| Database Connection String in QlikView | 11 |
| Qlik Security Table (Section Access) | 12 |
| Reuse of script code | 14 |
| Include files | 14 |
| Sub Functions | 14 |
| Pre-Defined Functions | 15 |
| QlikView Components (QVC) | 15 |
| Qlik Deployment Framework- Function Reference Guide | 15 |
| Data Modeling | 16 |
| Data models | 17 |
| Optimization strategy | 19 |
| Optimization Tips and Tricks | 20 |

| | |
|--|-----------|
| Other scripting best practices include: | 21 |
| Application logging | 23 |
| Deployment Framework log tracing and debugging | 23 |
| Using binary load with Deployment Framework | 23 |
| QlikView Front End Design | 24 |
| UI Design | 24 |
| QlikView Developer Toolkit | 24 |
| Color Scheme Variables | 27 |
| Variable expressions | 28 |
| QlikView Macros | 28 |
| QlikView Actions | 29 |
| Variable Editor | 30 |
| Overview | 30 |
| Variable Editor with Qlik Sense | 30 |
| Container Map Editor | 30 |
| Troubleshooting & Support | 34 |
| Support Types | 34 |

Development Guide v1.7.0

Using development best practices and guidelines is part of the Build and Validation phase in the Qlik Application Development process, read more in ***Qlik Deployment Framework-Qlik Product Delivery process.pdf***

Build

- Application development
- ETL
- Resource reusability

Validation

- Quality Assurance
- DevOps

Why the need for a Development Guide?

The Developer Guide is a reference manual for Qlik developers. Qlik developers are individuals who design and implement applications and their areas of expertise range from data modeling to scripting to UI design. This document is designed to facilitate much clearer understanding of the methodologies and practices that are optimal for producing highly usable, highly optimized and highly configurable Qlik applications, whether used by small departments or by large enterprises.

Company Specific Development Guide

This document is a high level guide on how to develop using Qlik Deployment Framework and predefinitions. Best practice is to create an adapted company specific development guide containing your development guidelines like:

- ETL and QVD strategy
- Container strategy
- Security requirements
- DTAP process

Platform strategy from a development perspective

When it comes to development, QlikView offer a wide range of flexibility. For many reasons there's a good idea to set up a corporate developing best practice. A corporate developing standard doesn't only include standards for how to optimize each and every single application but does also embrace methodologies and practices like reusability and overview.

Standards

It's important to have and use standards during Qlik development. There are many ways of getting the same result, but not all of them are efficient and understandable. Utilizing *Qlik Deployment Framework* in combination with development guide lines we create consistent multi development environment. Standards are needed for:

- Reuse of data
- Reuse of code
- Reuse of expressions and variables
- Multiple development
- Governance
- Creating and collecting understandable metadata

Using standards will result in lower cost of ownership by making governance easier and TCO lower.

Qlik Deployment Framework (QDF) Containers

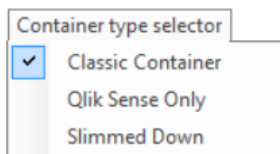
Qlik Deployment framework is based on the concept of **resource containers**. Containers are security boundaries, isolated file structures placed side by side. A container can be moved and/or renamed without changing any Qlik script logic. Each container has identical file structures and contains predefined script functions.

QDF Deploy Tool

Containers are created using the QDF Deploy tool available for download on [Qlik community](#). The container script code is open source available on GitHub approved updated will be incorporated into the deploy tool that also updates existing frameworks to latest standard. **Read more regarding Deploy Tool in the attached Read-me notes.**

Container Type Selector

From QDF version 1.6.5 you have the possibility to select between three container layouts. This mean that QDF Deploy tool extracts different container layout (folders) depending on the selection done in the *Container Type Selector*. The different settings and correlating layout are presented below.



- **Classic Container (default)** has same container layout as traditional QDF (from version 1.0)
- **Qlik Sense Only** Here we have removed everything related to QlikView, for example Application and mart folders. Extract, Transform, Load folders have also been added under the QVD structure.
- **Slimmed Down** is intended for smaller deployments and several folders have been removed. Extract, Transform, Load folders have also been added under the QVD structure.

| Classic Container | Qlik Sense Only | Slimmed Down |
|--|--|---|
| <ul style="list-style-type: none"> ▼ 1.Example <ul style="list-style-type: none"> > 1.Application 2.QVD > 3.Include > 4.Mart > 5.Config > 6.Script 7.Export 8.Import > 9.Misc | <ul style="list-style-type: none"> ▼ 1.Example <ul style="list-style-type: none"> ▼ 1.QVD <ul style="list-style-type: none"> 1.Extract 2.Transform 3.Load > 2.Config > 3.Include 4.Export 5.Import > 6.Misc | <ul style="list-style-type: none"> ▼ 1.Example <ul style="list-style-type: none"> > 1.Application ▼ 2.QVD <ul style="list-style-type: none"> 1.Extract 2.Transform 3.Load > 3.Include > 4.Config 5.Import |

Default Framework containers.

- **Administration** container it's from here additional containers are created and managed.
- **Shared folders.** A fresh QDF installation always contains a shared container, this is a repository to store scripts, configuration and data files that are shared and reusable by all applications.
- **Example** This is a container containing some examples and QVD files, this is used during the exercise documentation.

Folder Naming Convention

Folder names inside the container are standardized and simplified to fit as many languages and companies as possible. Before each container and subfolder there are a sequential number that makes it easier to document the structures. Example *1.2.1* symbolizes the first Container (1), The QVD folder (2) and a subfolder (1). This can also be used as name convention for reload tasks. Follow the number sequence and never use space when creating new containers or subfolders inside the container.

Classic Container (default) layout

| | |
|-----------------------|--|
| <i>0.Template</i> | Folder used for examples and templates. <i>Only exists in the 0.Administration Container.</i> |
| <i>1.Application</i> | QlikView Applications are resided in subfolders under 1.Applications |
| <i>2.QVD</i> | QlikView Data files are stored in subfolders under 2.QVD |
| <i>3.Include</i> | Folder where QlikView Include files are stored. These are script parts that are called from the main QlikView script. |
| <i>1.BaseVariable</i> | Stores all the variables needed to use the framework, like paths inside the container |
| <i>2.Locale</i> | Locale for different regions, used for easy migration between regions |
| <i>3.ConnString</i> | Stores connection strings to data sources |
| <i>4.Sub</i> | Store for sub routines, this is a way to reuse code between applications |
| <i>5.ColorScheme</i> | Company standard Color Scheme would be placed here |
| <i>6.Custom</i> | Store for custom include scripts |
| <i>4.Mart</i> | Resides QlikView QVW marts (in subfolders) for data discovery usage, these folders could be shared. |
| <i>5.Config</i> | Configuration and language files like Excel and txt. This folders could be shared to make configuration changes easier |
| <i>6.Script</i> | Store for special scripts run by the publisher or scheduled tasks |
| <i>7.Export</i> | Folder used to store from QlikView exported data, probably txt or qvx |
| <i>8.Import</i> | Folder used to store import data from external systems |
| <i>Info.txt</i> | Information files describing the folder purpose and Path variable. There are Info files in every folder. |
| <i>Version.xx.txt</i> | Version Revision list |

Container Variables

Each preinstalled folder in a container has corresponding environmental variable (*Global Variable*). Scripting using these variables makes it possible to move code and applications between containers without any script modifications, it also makes it possible to seamlessly share scripts between QlikView and Qlik Sense. To initiate QDF and create the global variables an initiation script (*InitLink.qvs*) need to be added in the beginning of the Qlik Sense load script.

Global Variables

Global variables are auto-generated by QDF during the initiation script, these variables are named **Global** because they are reused between applications within and across containers. The name standard is **vG.xxx (Variable Global)**.

| | | |
|-------------------|--|---|
| 0.Administration | | |
| 1.AcmeStore | | |
| 1.Application | <u>vG.ApplicationPath</u> | Application Folder |
| 2.QVD | <u>vG.QVDPath</u> | QlikView Data files (QVD) repository |
| 3.Include | <u>vG.IncludePath</u> | QlikView Include files repository |
| 1.BaseVariable | <u>vG.BaseVariablePath</u> | Initiation script and Global Variables storage |
| 2.Locale | <u>vG.LocalePath</u> | Regional / language Locale |
| 3.ConnString | <u>vG.ConnStringPath</u> | Connection string repository |
| 4.Sub | <u>vG.SubPath</u> | Sub Function Library |
| 5.ColorScheme | <u>vG.ColorSchemePath</u> | Color and Picture templates |
| 6.Custom | <u>vG.CustomPath</u> | Repository to keep custom include scripts |
| 4.Mart | <u>vG.MartPath</u> | Qlik mart repository |
| 5.Config | <u>vG.ConfigPath</u> | Stores QlikView configuration files |
| 6.Script | <u>vG.ScriptPath</u> | Folder to store special scripts used by publisher |
| 7.Export | <u>vG.ExportPath</u> | Data Export repository, probably txt or qvx |
| 8.Import | <u>vG.ImportPath</u> | Data Import from external systems |
| 2.AcmeHR | | |
| 99.Shared_Folders | | |

Global Variables within the classic container

Custom Global Variables and Universal variables

- **Custom Global Variables** are framework variables created manually by the developers. These variables are also auto-generated during initiation using the *CustomVariables.csv*, that file exists within each container under *3.Include\1.BaseVariable*. Adding name and value in this csv file makes variables available for the Qlik Sense applications that is using the home folder where the csv is resided.
 - **Global variables** should only be used when variables are shared by several applications in a Container.
- **Universal Variables** is similar to *Custom* but used across multiple containers. Universal are stored in the Shared Folders Container creating a “single point of truth” across all containers. Universal Variables are stored in $\$(SharedBaseVariablePath)\CustomVariables.csv$ file and loaded during the framework initiation process.
 - **Universal variables** should be used when variables are shared by several applications across all Containers.

Using Deployment Framework Containers when scripting

When using the Deployment Framework applications need to have an initiation include script in the beginning of the load script which identifies the environment (within your home container) and generates global variables that is used during development. This initiation script is called *InitLink.qvs* and resides in the base of every container.

Get started with QDF in QlikView Developer

This initiation creates environmental *Global Variables* like container folder path and *Custom Global Variables* that is additional Global Variables created by you.

1. Create or save QlikView application (preferably in a subfolder) under *1.Application* folder in the container. Create a first Tab called **Qlik Deployment framework** and Paste in the initiation script code below.


```

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

```

2. Reload and check in QlikView Variable Overview for Global Variables (vG.xxx) pointing to container folders, as shown below:

| Variable Overview | | |
|----------------------|--|---------|
| Variables | | |
| Variable Name | Value | Comment |
| vG.BasePath | C:\Users\mbg\Documents\QlikViewDeployment\ | |
| vG.TemplatePath | C:\Users\mbg\Documents\QlikViewDeployment\ | |
| vG.BaseVariablePath | C:\Users\mbg\Documents\QlikViewDeployment\ | |
| vG.SubPath | C:\Users\mbg\Documents\QlikViewDeployment\ | |
| vG.RootPath | C:\Users\mbg\Documents\QlikViewDeployment\ | |
| vG.ApplicationPath | C:\Users\mbg\Documents\QlikViewDeployment\ | |
| vG.DocumentationPath | C:\Users\mbg\Documents\QlikViewDeployment\ | |
| vG.QVDPPath | C:\Users\mbg\Documents\QlikViewDeployment\ | |

3. These variables are used when developing instead of hardcoding or using relative paths. The benefit is that applications can be moved in all directions without breaking the script.

Getting Started Guide

More tips and trix on how to setup QDF is available under [Qlik Deployment Framework-Getting Started Guide.pdf](#)

Additional Naming Convention

It's important to follow a variable naming convention so that existing application variables don't collide with the framework variables. A name standard also makes it easier to understand and search among variables.

- Store often used expressions as Local variables
- Store reusable expressions as Global or Universal variables
- Extended name standard for Variables are possible, example:
 - Local expressions variables starts with *vL.Calc_*
 - Global expressions variables starts with *vG.Calc_*
- Variables defining a path should always end with a '\'
- Reset local variables that are only used inside the script and not in the UI.
Enter the variable name and =; example: SET *vL.test* =;

Back End Development

Back end development involves the process that starts with extracting data from one or many data sources and ends up in creating a Qlik's associative data model managed by the load script editor.

Scripting basics

Scripting is the environment in which a Qlik Developer will automate the extract, transform and loading process of bringing data in the Qlik environment. Each Qlik application contains a script through which this process is enabled.

Best practices dictate that using multiple tabs within a script will split out the various parts, enabling a simple view of the information for future development and support. Depending on the complexity of the application, you may have a variety of different script sections. The common parts of a script are below:

- QDF initiation tab
- Security (usually hidden script)
- Dates and Calendar information
- Tab per data source
- Tab per key measure/core table
- Tab per lookup table

Linking (mount) Containers

After QDF initiation you can start using the sub function library (more documentation later in this document) below function is important as it creates Global Variable Links other resource containers (and available folders). The benefit of using Global Variable links in the script is to create generic, reusable and movable code as QDF validates and regenerates the links every time the script runs.

Load Container Global Variables (LCGV)

By using **LCGV** function it's possible to create Global Variable links (mounts) between containers (security access needed).

Example: `call LCGV('AcmeTravel')` Will create all Global Variables linking to 2.AcmeTravel container. Variables created will have similar name as home container but with the additional *AcmeTravel* prefix, like *vG.AcmeTravelQVDPath* for QVD path to AcmeTravel container

`call LCGV('Oracle','QVD;Include');` Will create two Global Variable links to different resources in Oracle container, by using an additional switch and ';' separator creates Global Variables *vG.OracleQVDPath* and *vG.OracleIncludePath* (instead of linking all folders as in the first example).

Additional Sub Functions

Qlik have the possibility of reusing scripts and functions by using the **Sub** and **Call** commands. As presented above with the **LCGV** function. The Framework contains library of nice to have functions.

Qlik Deployment Framework- Function Reference Guide

The function reference guide is available as a separate document ***Qlik Deployment Framework-Function Reference Guide.pdf***. As the sub functions are identical to both Qlik Sense and QlikView the same guide applies to both platforms.

Data strategy

To handle infinite amounts of data using Qlik we need to store data into Qlik Data (QVD) files, Qlik then reads QVD data in millions of records per second. This means that instead of loading all available data into Qlik, we only load needed data. This seems simple at first, but for this we need to understand what data we have and data needed? There could also be a need to prepare and “stage data” this to cleans and aggregate. A data strategy is to structure (using containers) and document the process. Read more in the *Qlik Deployment Framework-Deployment Guide.pdf*

Staging data

Staging data is a process of intermediately storing data between the sources of information, most often in QVD files. Staging usually consists on these steps:

- Main
 - Include statement for connection string ODBC/OLE DB stored in *vG.ConnStringPath*
 - Meta information about the application. I.E owner, purpose.
- Extract
 - Extracting the sources needed. Using an incremental approach when applicable. If there is no need for transformation the source could be stored directly to the presentation layer in the QVD-folder using Global variable (*vG.QVDPath*).
- Transform
 - When transformation is needed. For example creating new fields, cleansing information, aggregate and so on. This will be executed here.
 - In this stage recommendation is to use the QDF Index function to tag and index the QVD for easy search and retrieval.

Database Connection String in QlikView

Connection strings are a security credential and should always reside in the architecture Back end, in the data tier. Well protected from unauthorized access. Remember that architecture front-end (QlikView Server and QlikView Web Server) does not have any open ports the back-end, these servers could be in different network zones and security boundaries. When distributing QlikView applications via the Publisher in Back end to the application tier, scripts and connection strings will automatically be removed.

Security considerations

By separating the connection string from the script reusability and higher security will be achieved. There are two ways to separate and reuse the connection strings:

Include File

Best practice is to keep the connection strings in a separate Include file. This behavior is supported by Deployment Framework. Use the Global Variable *vG.ConnStringPath* to connect inside your container, example:

```
// Connection string to Northwind Access data source
$(Must_Include=$(vG.ConnStringPath)\0.example_access_northwind.qvs);
```

If the connection string is in another container, for example the Shared folders use the Global Variable *vG.SharedConnStringPath* to connect, example:

```
// Connection string to Northwind Access data source
$(Must_Include Include=$(vG.SharedConnStringPath)\0.example_access_northwind.qvs);
```

Recommendation is to use **Must_Include** so that the Qlik script fails if include script is missing.

Connection string as a Global Variable

In Global Variable Editor, there is the possibility to add the connection string into a Custom Global Variable. This method is not as secure as using an include file. Include files can be secured by different security groups this is not possible when using Global Variables that will be reused across all applications within a Container. But when a container is secured and dedicated for a source system (example Oracle container) connection strings as global variables could be used.

Qlik Security Table (Section Access)

In Qlik the data security access in a qvw or qvf file is set in the script. Access rights and User Levels are defined in the Section Access part of the script. Section Access can be used to set access restrictions to data, sheets and sheet objects. All access control is managed via files, SQL databases or inline clauses in the same way as Qlik normally handles data. If an access section is defined in the script it must be followed by the statement Section Application to load normal data.

Section Access system fields

Access levels are assigned to users in one or several tables loaded within the Section Access. These tables can contain several different user-specific system fields, typically NTNAME and the field defining the access level, ACCESS. The full set of section access system fields are described below. Other fields like GROUP or ORGANISATION may be added to facilitate the administration, but QlikView does not treat these fields in any special way. None, all, or any combination of the security fields may be loaded in the access section. However, if the ACCESS field is not loaded, all the users will have ADMIN access to the document and the section access will not be meaningful.

Section access system fields for QlikView are:

- **ACCESS** A field that defines what access the user should have.
- **NTNAME** A field that contains a string corresponding to a SSO user name or group name.
- **USERID** A field that contains a user ID that has the privilege specified in the field ACCESS.
- **PASSWORD** A field that contains an accepted password.
- **SERIAL** A field that contains a number corresponding to the QlikView serial number.
- **NTDOMAINSID** A field that contains a string corresponding to a Windows NT Domain SID
- **NTSID** A field that contains a Windows NT SID.

Section Access in Combination with QlikView Publisher

While QlikView Publisher can use its “loop and reduce” functionality to reduce a QVW by rows by user or group as it is being reloaded, you can also accomplish this in Section Access dynamically as the document is opened, either method will work and both have benefits. The Loop and reduce from Publisher will help you to reduce the memory footprint of the QVWs on your server(s), while the Section Access method is portable with the document. Another reason to use Section Access is the application of authentication in the QVW, through SSO or user ID and password. This is especially important if the QVW is going to be enabled for download from the AccessPoint or otherwise distributed to users.

Best Practices when using Section Access:

- In Section Access, always use the Upper() function when utilizing a load statement, use it on every column no matter what. (even when reading from .qvd)
- AD Groups for security if possible
- Security in include files for reuse
- Add the Publisher’s service account to the Section Access table
- Utilizing a ‘Star Schema’ design for the data model with NO LINK Tables.
Link tables hurt performance greatly!
- Best case is to have 1 fact table with the dimensions all directly connected to the fact.
In rare instances should additional ‘snowflake’ dimensions to be used.
- In the fact tables, have no more than 30 – 40 columns defined (there can be a few more/less, but do not have 150 columns unless you fact is less than 10 Million records (with a decent server)
- Many times, having too many columns are a situation brought on by utilizing ‘Role Playing Metrics’.
While this may be helpful, too many of these metrics create a performance degradation on the server.

Reuse of script code

For easier manageability and faster development, it's recommended to reuse script code as much as possible. By using Deployment Frameworks predefined structures and variables it's easy to reuse script code. There are two ways of reusing code in Qlik Script:

- Include files
- Use of functions

Include files

An include file is just a Qlik script (text file) that is included into the main script during execution. Qlik include scripts use the prefix *qvs*. The entire or parts of the script can thus be put in a file for reuse.

All Include files are stored in *6.Custom* folder, the global variable for *6.Custom* folder is *vG.CustomPath* and should always be used when accessing a custom script, meaning that it's not a part of the Deployment Framework initiation process. Example: `$(Include=$(vG.CustomPath)\1.xyz_Calculations.qvs);`

Sub Functions

Qlik have the possibility of reusing scripts and functions by using the **Sub** and **Call** commands. As presented with the *LCGV* function. The Framework contains library of nice to have functions. All sub functions are stored under the *3.Include\4.Sub* folder and are initiated during QDF initiation.

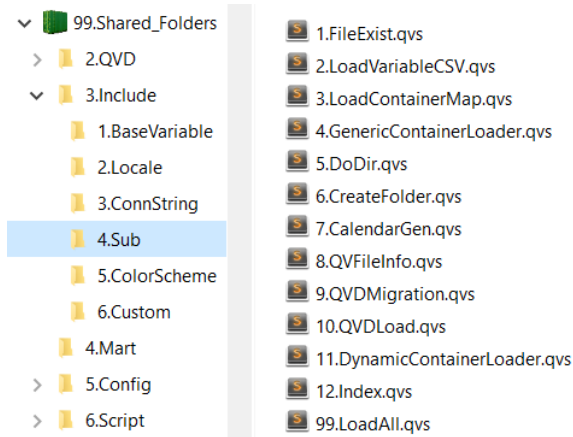
Use `Call function_name('Input parameters or variables')` command to execute the preloaded function.

Another function example, *vL.FileExist* will return true or false depending on if the file exists

`Call vL.FileExist ('$(vG.QVDPATH)\SOE.qvd')`

Pre-Defined Functions

QDF contains library of functions that is loaded automatically during initiation. A copy of the function library is stored under every container, but the primary sub function library is located under *3.Include\4.Sub* in Shared_Folders container.



If no Shared container exists, as a backup the local container sub function library will be used. Meaning as long as a Shared folder exists all sub function additions should be stored under the Shared container

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

QlikView Components (QVC)

Optionally, the open source library *QlikView Components (QVC)* can be installed using the deploy tool in addition to the built-in QDF library, read more on [QVC here](#). If QVC been installed it can be disabled in the script by adding this line before QDF initiation:

```
set vG.QVCDisable='true';
```

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

Qlik Deployment Framework- Function Reference Guide

Function reference guide containing functions and its operators is available as a separate document *Qlik Deployment Framework-Function Reference Guide.pdf*. As the sub functions are identical to both Qlik Sense and QlikView the same guide applies to both platforms.

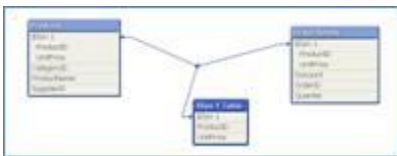
Data Modeling

Understanding

The cornerstone of Qlik is the associative in-memory search technology.

There are some very specific characteristics with this technology that you have to keep in mind.

- Two fields in different tables with exactly the same name, case sensitive, will automatically be connected to each other and fields with exactly the same field value, case sensitive, will be associated with each other.
- If two tables have more than one field in common, Qlik will automatically create a synthetic key a kind of link table. The easiest way to detect a synthetic key is by opening the table viewer (Ctrl-T):



Synthetic key

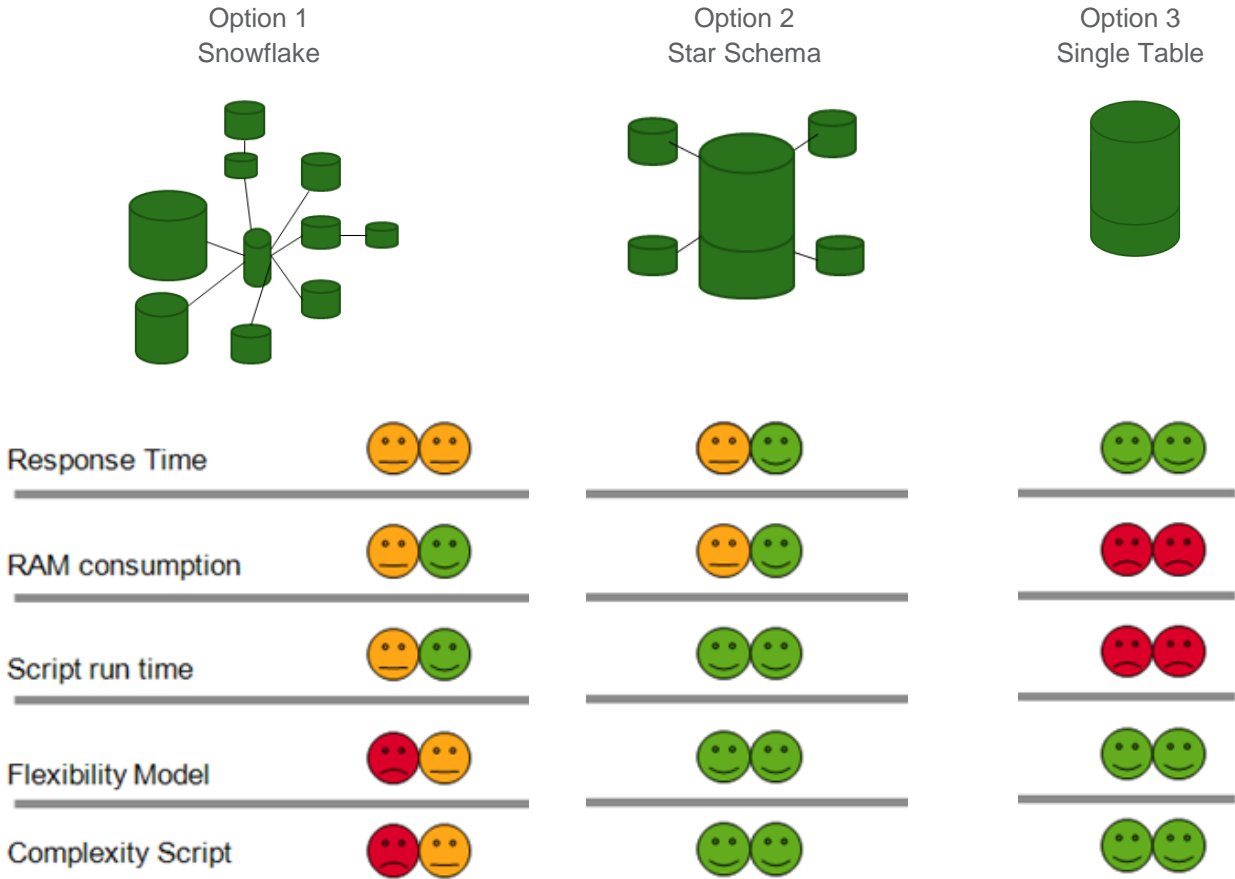
Another characteristic with the associative database is that the number of distinct (unique) values in a table is more important than the number records. By delimit the number of distinct values in a table the performance of an application can be significantly improved.

Example: Let's say you have a fact table with 1 billion recs, one of the fields is a timestamp field containing date and time (measured down to fraction of seconds) with almost 800 million distinct values. Two alternative actions will both improve the performance:

- If you don't need to analyze on time level, simply transfer the field to a date field (use makedate function) and there will not be more than 365 distinct values for one year.
- If you need to analyze on time level, determine on what time level you need to analyze (hour, minute) and create a new field, Time. Depending on what level you decide to analyze, hour will give you 24 distinct values and minute will give maximum 1440 distinct values)

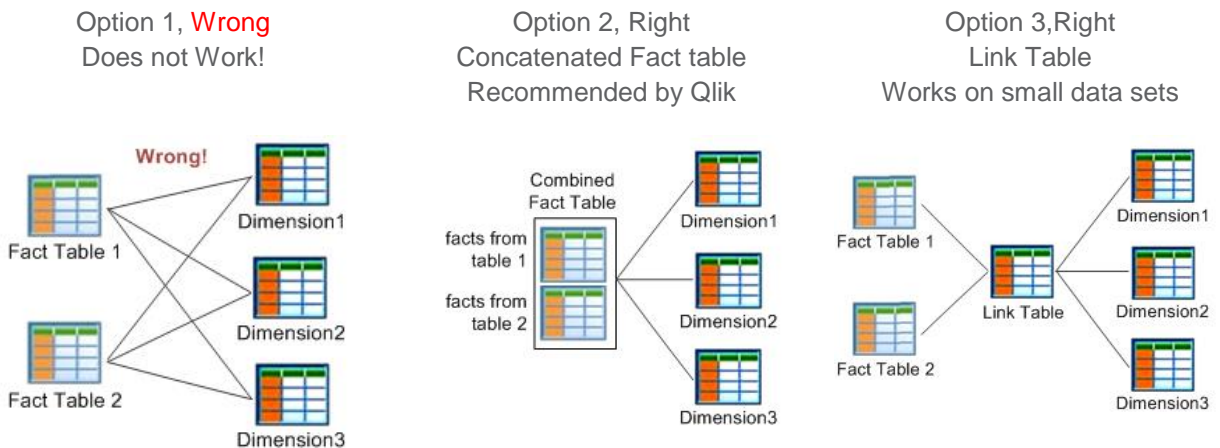
Data models

Represented below are diagrams of 3 basic data models that can be built in Qlik (along with many other combinations). Using these 3 examples we can demonstrate some of the differences in performance, complexity and flexibility between them.



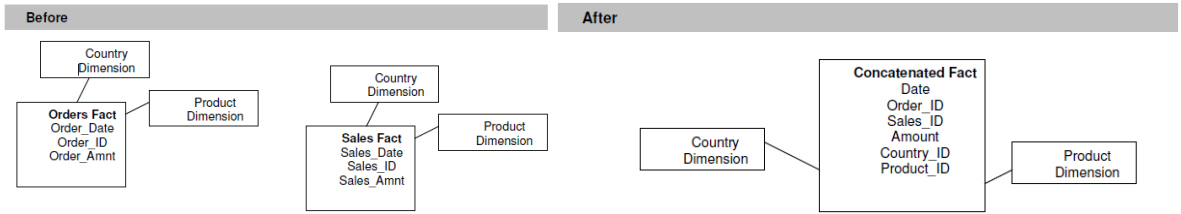
Multiple fact tables

While star schemas are generally the best solution for fast, flexible QlikView applications, there are times when multiple fact tables are needed. Here are the wrong and right ways to join them:



Further examples of how to build and use link tables are contained in QlikCommunity on line (<http://community.qlik.com/>)

To show how this could be accomplished, the section below takes us through a scenario of two facts tables to be combined into one fact table.



Script Example:

```

Load OrdersFact
  Order_Date as Date
  Order_ID
  Order_Amount as Amount
  Country_ID
  Product_ID
  'Order' as TransactionType
CONCATENATE
Load SalesFact
  Sales_Date as Date
  Sales_ID
  Sales_Amount as Amount
  Country_ID
  Product_ID
  'Sale' as TransactionType
  
```

Placing the 'Sale' and 'Order' text types in the script will provide you with a column to determine the transaction type.

Sales

| Region | Product | Date | Sales |
|---------|---------|------------|-------|
| RegionA | P1 | 2009-01-31 | 100 |
| RegionA | P1 | 2009-02-28 | 120 |
| RegionA | P1 | 2009-03-31 | 140 |
| RegionA | P2 | 2009-01-31 | 500 |
| RegionA | P2 | 2009-02-28 | 550 |
| RegionA | P2 | 2009-03-31 | 600 |
| RegionB | P1 | 2009-01-31 | 50 |
| RegionB | P1 | 2009-02-28 | 55 |
| RegionB | P1 | 2009-03-31 | 60 |
| RegionB | P2 | 2009-01-31 | 200 |
| RegionB | P2 | 2009-02-28 | 180 |
| RegionB | P2 | 2009-03-31 | 160 |

Plan Yearly

| Region | Date | Plan |
|---------|-----------|-------|
| RegionA | 2009-01-1 | 8000 |
| RegionB | 2009-01-1 | 10000 |

Procurement Cost

| Product | Date | Cost |
|---------|------------|------|
| P1 | 2009-01-31 | 130 |
| P1 | 2009-02-28 | 1400 |
| P1 | 2009-03-31 | 1600 |
| P2 | 2009-01-31 | 500 |
| P2 | 2009-02-28 | 650 |
| P2 | 2009-03-31 | 600 |

Concatenated Facts

| Region | Product | Date | Sales | Plan | Cost |
|---------|---------|------------|-------|-------|------|
| RegionA | P1 | 2009-01-31 | 100 | | |
| RegionA | P1 | 2009-02-28 | 120 | | |
| RegionA | P1 | 2009-03-31 | 140 | | |
| RegionA | P2 | 2009-01-31 | 500 | | |
| RegionA | P2 | 2009-02-28 | 550 | | |
| RegionA | P2 | 2009-03-31 | 600 | | |
| RegionB | P1 | 2009-01-31 | 50 | | |
| RegionB | P1 | 2009-02-28 | 55 | | |
| RegionB | P1 | 2009-03-31 | 60 | | |
| RegionB | P2 | 2009-01-31 | 200 | | |
| RegionB | P2 | 2009-02-28 | 180 | | |
| RegionB | P2 | 2009-03-31 | 160 | | |
| RegionA | | 2009-01-1 | | 8000 | |
| RegionB | | 2009-01-1 | | 10000 | |
| | P1 | 2009-01-31 | | | 130 |
| | P1 | 2009-02-28 | | | 1400 |
| | P1 | 2009-03-31 | | | 1600 |
| | P2 | 2009-01-31 | | | 500 |
| | P2 | 2009-02-28 | | | 650 |
| | P2 | 2009-03-31 | | | 600 |

A concatenation of fact tables example.

Preceding Loads

The use of preceding load statements can simplify your script and make it easier to understand. See the code below for an example of this.

Table1:

```
LOAD CustNbr as [Customer Number],
      ProdID as [Product ID],
      floor(EventTime) as [Event Date],
      month(EventTime) as [Event Month],
      year(EventTime) as [Event Year],
      hour(EventTime) as [Event Hour];
SQL SELECT
      CustNbr,
      ProdID,
      EventTime
FROM MyDB;
```

This will simplify the SQL SELECT statement so that the developer can continue to test/augment the statement using other tools, without the complexity of the Qlik transformations embedded in the same SQL statement.

For more information on the Preceding LOAD feature, see the help.qlik.com.

Optimization strategy

Qlik is known for its wide user adoption. One of the main reasons for this is its capability to manage large data sets with short response time. Although a Qlik application most often is easy and fast to develop it's a very good idea to establish an optimization strategy as part of your development platform. As with most Qlik development, optimization is divided into a back-end and a front-end part. While back-end optimization focus on effective script and data modeling, the front end focus on user interface design with its charts, dimension and expressions. For long term success it is strongly recommended that you have an optimization focus in your application development, especially when you know that the application should hold a large data set and be distributed to a large number of users. A good idea is to have an optimization step connected to the validation/approval phase in your development process, this of course both for new applications as well as for changed/ improved applications.

You can read more detailed information, tips and tricks, about optimization in the back-end and front-end section of this document.

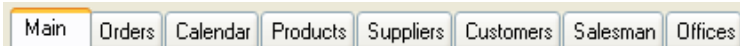
Optimization Tips and Tricks

- Please keep in mind that what really counts when it comes to optimization of a data model is the number of records.
- Don't normalize data too much. Plan for 6 – 10 total tables in a typical application. This is just a guideline, but there is a balance to be struck with data models. See the Data Model section of this document for more details.
- Eliminate small "leaf" tables by using Mapping Load to roll code values into other dimensions or fact tables.
- Store any possible field as a number instead of a string
- De-normalize tables with small numbers of field
- Use integers to join tables together
- Only allow 1 level of snow flaked dimensions from the fact record.(fact, dimension, snowflake, none)
- Use Autonumber when appropriate, will reduce application size
- Split timestamp into date and time fields when date and time is needed
- Remove time from date by floor() or by date(date#(..)) when time is not needed
- Reduce wide concatenated key fields via Autonumber, when all related tables are processed in one script (There is no advantage when transforming alphanumeric fields, when string and the resulting numeric field have the same length)
- Use numeric fields in logical functions (string comparisons are slower)
- Is the granularity of the source data needed for analysis? If not aggregate by using aggregating function like "sum() group by"
- Create numeric flags (e.g. with 1 or 0)
- Reduce the amount of open chart objects
- Calculate measures within the script (model size <> online performance)
- Limit the amount of expressions within chart/pivot objects, distribute them in multiple objects (use auto minimize)

Additional scripting best practice

Other scripting best practices include:

- Use Autonumber only after development debugging is done. It's easier to debug values with a number in it instead of only being able to use surrogates. See Reference Manual if you are not sure how/when to use Autonumber.
- Put subject areas on different tabs so you don't confuse the developers with too much complexity



- Name the concatenate/join statements
- Use *HidePrefix=%*; to allow the enterprise developer to hide key fields and other fields which are seldom used by the designer (this is only relevant when co-development is being done).
- When using the *Applymap()* function, fill in the default value with something standard like 'Unknown' & Value which is unknown so users know which value is unknown and can go fill it in on the source system without the administrators having to get involved. See Reference Manual if you are not sure how/when to use Applymap().

```
StateMapping:
mapping load * inline [
St,State
Tx,TX
Te,TX
Tex,TX];
LOAD
ApplyMap( 'StateMapping' , St, 'Other')
```

- Never use Underscores or slashes (or anything 'techie') in the field names. Instead code user friendly names, with spaces.
- Instead of:"mnth_end_tx_ct" use:"Month End Transaction Count"
- Only use Qualify * when absolutely necessary. Some developers use Qualify * at the beginning of the script, and only unqualify the keys. This causes a lot of trouble scripting with left join statements, etc. It's more work than it's worth in the long run. See Reference Manual if you are not sure how/when to use Qualify and Unqualify.
- Use "Include" files or hidden script for all ODBC/OLEDB database connections.
- Use variables for path name instead of hard-coding them throughout your script. This reduces maintenance and also provides a simple way to find paths (assuming you put them in the first tab to make it easy to find).
- All file references should use Container naming convention.
- Always have the Log file option turned on if you need to capture load-time information for debugging purpose
- Comment script headings for each tab. See example below:

```
=====
// App Name:   Wireframe
// Author:     Matt Stephens, QlikTech
// Created:    June, 2010
// Purpose:    This app is a template app demonstrating the use of
//             wireframe backgrounds to organize QlikView screens into
//             logical and effective presentation themes. There is also
//             a zip file called Wireframe Images.zip that accompanies
//             this QVW. It holds dozens of pre-built wireframe images
//             in various color schemes.
// Modified:   July 18, 2010 BPN - added Intro tab comments
=====
```

- Comment script sections within a tab with short descriptions. See example below:

```
// -----
// Load the Sessions table first
// -----
Sessions:
LOAD
MakeDate(LEFT(Timestamp,4), MID(
Date(Timestamp, 'YYYYMMDD') & '_'
Time(Timestamp) as SessionsTi
Timestamp as Timestamp,
```

- Add change date comments where appropriate. See example below:

```

Looptable:
LOAD FileName as QVDName
//FROM $(MetaPath)FileList.qvd(qvd)
resident FileList //changed 2010-09-06
WHERE UPPER(Extention) = 'QVD';

```

- Use indentation to make script more readable by developers. See example below:

```

// -----
// Main loop though all QVDs found above
// -----
for X = 1 to fieldvaluecount('QVDName');
  let QVDName = fieldvalue('QVDName',$(X));

  Load *,
  upper('$(QVDName)' & '_' & Date(Today(), 'YYYY-MM-DD') as LoadDateKey,
  lower('$(QVDName)') as FieldQVDFileName,
  Upper('$(QVDName)' & '-' & date(Today(), 'YYYY-MM-DD')) as FieldHeaderKey;

  QvdFieldHeader:
  LOAD
  //lower('$(QVDName)') as QVDFileName,
  date(Today(), 'YYYY-MM-DD') as FieldHeaderDate,
  FileName as QVDFieldName,

```

- Never use LOAD * in a load statement. Instead list the columns to load explicitly so that you know what fields will be loaded and this won't change as new columns are added or deleted from source tables. This also helps developers to identify the loaded fields in the script. See example below:

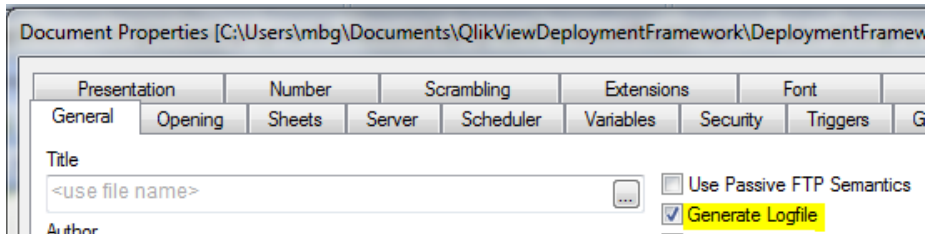
```

// =====
// Locations Data from a QVD
// =====
Locations:
LOAD LocationNbr as [Location Nbr],
AddressLine1 as [Address Line 1],
AddressLine2 as [Address Line 2],
City as City,
Country as Country,
CountryRegionCode as [Region Code],
PostalCode as [Postal Code],
[State / Province] as [State Code]
FROM [$(QvdPath)Locations.qvd] (qvd);

```

Application logging

In QlikView it's best practice to turn Document logging on under Document Properties and General Tab. These logs can be used to monitor the system by use of the Governance Dashboard. These logs are also very handy when debugging.



Note that Qlik Sense always have the script log activated and stored under the `%AppData%\Qlik\SenseVog` folder

Deployment Framework log tracing and debugging

When the log is activated it's easy to find where in the QDF initiation scripts the problem has accrued.

Search for the log trace that starts with `### DF` alt `### DF Error` and after the section/include file name.

If error in the script is not generated in Deployment Framework section a good idea is to comment the initiation scripts and thereby using old Global Variables. The advantages of this is that the application log and debug sequence is shorter thereby easier to debug. Remember to activate QDF initiation after the debugging.

If having problems with Section Access, Input Fields or other faults making application access impossible, use the initiation script (`1.Init.qvs`) as your escape. The command `Exit script;` in the beginning of `1.Init.qvs` will exit before the faulty script part executes.

Using binary load with Deployment Framework

To load from a Qlik mart the binary load statement need to be used in the scripts. Binary load can only be put as the `first` statement of a script. Best practice is thereby to use relative search path to the qvw or qvf in the binary section, instead of the framework global variables. Example:

```
Binary [..\..\4.mart\0.example_northwind_mart\example_northwind_mart.qvw];
```

The Deployment framework `InitLink.qvs` include sections will follow the Binary load section.

QlikView Front End Design

When creating a new user application, it should always take the starting point from a Template Application. The document template should include the standard structure in the script and the companies, visual guidelines implemented. Qlik Sense is not valid in this section as to uses responsive design and a limited color scheme.

UI Design

Design matters. It impacts user adoption rates, utilization rates, speed of analysis and usage patterns. All of these things impact how effective your QlikView document can be. The principles of good interface design promoted by Stephen Few and Edward Tufte are the basis for the best practices Qlik recommends when designing and building a QlikView application. The outline below shows (at a high level) some of those tenants of good design.

QlikView Developer Toolkit

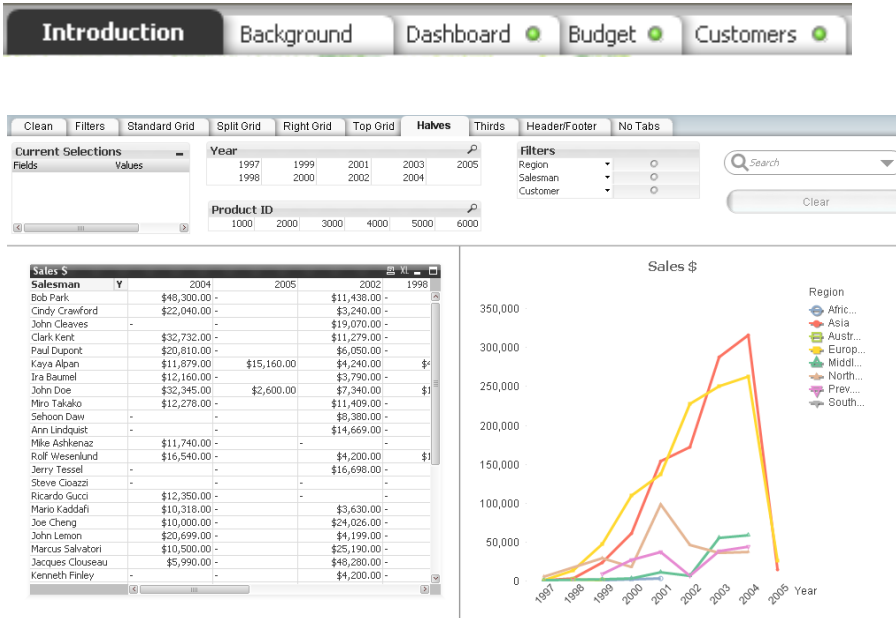
The number one tool for a QlikView Designer is the *QlikView Developer Toolkit* which is available with the installation of QlikView 11. The purpose of the Developer Toolkit is to help QlikView developers make more attractive & useable applications. There are a variety of backgrounds, guides, and panels that can be incorporated into your design to get you started

Developer Toolkit is divided up into several folders of assets

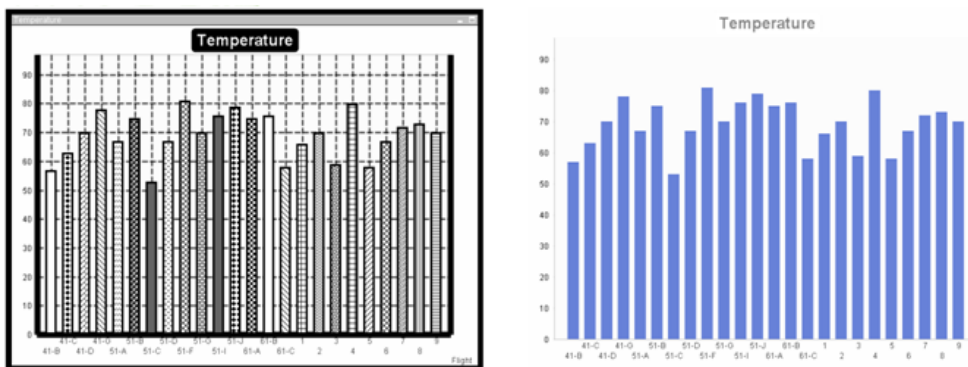
- Backgrounds: help define space to place objects on
- Buttons: images to use as buttons
- Guides & Rulers: help you align objects within QlikView
- Icons: useful images for common tasks
- Panels: can be used to define spaces when using a background you have found
- Qlik: Qlik branded images
- Rules: are simple line styles to divide up regions of space
- Shadows: are more graduated ways of dividing space

UI Best Practice

Use of supplied or developed templates and tabs for consistency and simplicity:



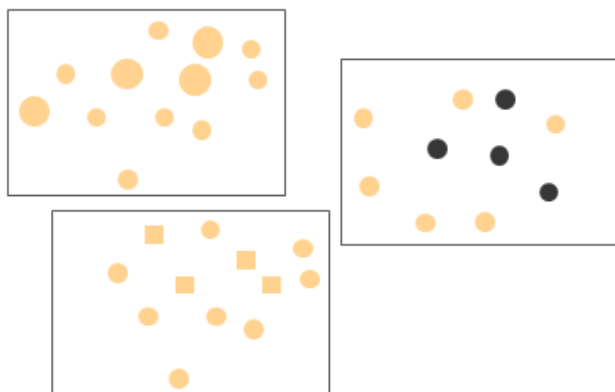
Use of implied closure to limit non-data ink space:



Use of neutral and muted colors and use of contrast: Muted and neutral colors are much less strenuous on the eyes and increase user adoption. Use of contrast helps the eyes quickly identify interest points or exceptions. These concepts go together, since the use of contrast with primary colors is difficult to do. Consider a combination of muted colors and the use of contrast in all charts, especially where exceptions or outliers are meant to be highlighted.



Use of size, shapes and intensity to call attention to data points: Shapes are another rapid identification point for the eyes. They can be used to segment data points into groups. Color intensities work well for ranges of values or outliers.



Additional UI Best Practice

- Put a current selections box on every sheet in the same location
- Make list boxes appear in the same locations on every sheet
- Organize list boxes and multi-boxes first in the frequency of use (most used on the top, least used on the bottom). Then, sub-sort the list boxes into groups in hierarchical order (largest group on the top, smallest group on the bottom).
- Put dropdown select properties on every straight/pivot table
- Use Variables as expressions instead of defining the expressions directly in the expression editor
- When Creating a Drill group, add an expression for the label of the field in the drill group. The expression should be equal to Only (All Higher fields) & '>' & 'current field name', so that it equates to *SalesRepA>Product.SalesRepA* is the item which was drilled into, Product is the values which are represented in the chart
- Instead of defining exceptions in straight/pivot tables, instead use charts which show the exceptions quickly
- Always include a Help / How-To tab and/or a link to a help site on our website. Examples of Help/How-To tabs are included in the Getting Started section in QlikView. Consider copying one of the interactive How-To pages into a template that you can use across applications.
- Name each sheet and object with descriptive headers
- Black & White charts are best when considering color blindness and simplicity
- Red & Green - Many people are red/green color-blind - consider this e.g. when using visual cues
- Red and green are also associated with good and bad indicators / performance. Only use red and green when you mean to indicate good and bad.
- Design for a fixed resolution that applies to your organizations desktops (e.g.1024 x 768)
- Always consider sort order and whether to present frequency (# or %) in list boxes (sometimes very useful but definitely not always)
- Repeated objects (clear buttons) at the same position in every sheet
- Multi boxes can be good for people that are used to working with QV but they are not very intuitive. List boxes take more space but are better (you can e.g. see the gray areas better).
- Clean layout in charts – line up axis titles, chart title, text, etc...
- Hierarchy dimensions placed in order
- Time and Dates are crucial elements of most apps and they must be highly intuitive to search and use
- Table columns should always be searchable (display totals in tables whenever it makes sense)

Qlik strongly recommends the incorporation of design best practices for all developers and designers when starting a Qlik deployment. Good interface design leads to high adoption rates and effective interfaces.

QlikView's rich UI layer allows for world class visualization and design in all applications.

For new QlikView deployments and new designers it is strongly recommended that QlikView Designer training be attended by all developers and designers. The Designer courses are structured to reinforce good design and to learn the QlikView techniques that help deliver that design in a simple, elegant way. They are also a great opportunity to practice good design and apply that design to your QlikView applications in a lab setting.

Many of the design best practices are displayed in the demo applications that are publicly available at <http://www.demo.qlik.com>. Also visit QlikCommunity for more tips.

UI Design References

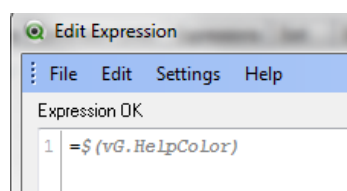
- QlikView Developer Toolkit
- QlikView Demo <http://www.demo.qlikview.com>
- [Information Dashboard Design](#), by Stephen Few
- [Show Me the Numbers](#), by Stephen Few
- [The Visual Display of Quantitative Information](#), Edward R. Tufte
- [Visual Explanations](#), by Edward R. Tufte

Color Scheme Variables

Use Global Variables to reuse company color schemes. It is easier more consistent to develop the GUI when using pre-defined color variables. Use the Color Scheme global variable `vG.ColorSchemePath` and store schemas in include files. Example, include Color scheme script in Deployment Framework tab after init section:
`$(Include=$(vG.ColorScheme)\0.Example_ColorScheme.qvs);`

In the scheme include file add global color variables by using the company RGB codes
`SET vG.HelpColor = RGB(234,94,13);`

Apply the global color variables on the objects



Variable expressions

Global Variables is a good way of reusing expressions, edit the expressions in Variable Editor.

Reasons for holding expressions in variables:

- To achieve reuse: the formula for a measure such as Sales usually remains the same across an application, so it doesn't make sense to write it on every chart.
- To enforce consistency in the formulas: by avoiding the risk of having different formulas that calculate the same measure.
- To provide a single point to apply changes: if and when a formula needs to be changed, you only need to change one variable and all the charts and other objects that refer to that variable will follow.
- To allow the end user to make changes through an input box, when needed. This could be the case of targets for KPIs or general parameters.

Expression Optimization Tips

- Eliminate Count(Distinct x)'s They are very slow
- Eliminate Count Numbers, or Count Texts, they are almost as slow as Count(Distinct)
- `date(max(SDATE,'DD.MM.YYYY'))` is factor xxx faster than `max(date(SDATE,'DD.MM.YYYY'))`
- Use numeric flags (e.g. with 1 or 0) which are pre-calculated in the script
- `sum(Flag * Amount)` and `sum(if(Flag, Amount))` use instead `sum({Flag=1} Flag * Amount)`
- Reduce the amount of open chart objects
- Limit the amount of expressions within chart/pivot objects, distribute them in multiple objects (use auto minimize)

QlikView Macros

The following are some reflections you should be aware of when you start including macro statements in your application. There are also a number of reasons why to avoid macros

Running a macro could result in deletion of the QlikView Server cache. undo-layout buffers and undo logical operation buffers and this in general has a very large negative impact on performance as experienced by the clients. The reason for deleting the caches etc. is that it is possible to modify properties, selections from the macros, thus opening up for conflicts between the cached state and the state that was modified from a macro and these conflicts will practically always crash or hang the clients (and in worst case; hang or crash the server as well).

The macros themselves are executed at VBS level while QlikView in general is executed at assembler level which is thousands of times faster by de-fault. Furthermore, the macros are single threaded synchronous as opposed to QlikView that is asynchronous and heavily threaded and this causes the macros to effectively interrupt all calculations in QlikView until finished and thereafter QlikView has to resume all interrupted calculations which is a delicate process and very much a source (at least historically) for deadlocks (i.e. QlikView freezes while the macro is still running, without any possibility that the macro will be finished).

While QlikView is increasingly optimized in terms of performance and stability, the macros will always maintain their poor performance and the gap between genuine QlikView functionality and the macros will continue to increase, making macros less and less desirable from a performance point of view. This fact combined with the above fact that the macros tend to under-mine all optimizations made in QlikView calls for severe negative tradeoffs as soon as macros become an integral part of any larger application.

The macros are of secondary nature when it comes to QlikView functionality - first all internal basic QlikView functions are run and tested and thereafter the macros are run and tested which effectively means that macros will never have the same status or priority as basic QlikView functionality - always consider macros as a last

resort but nothing much else. Since the automation API reflects the basic QlikView in terms of object properties etc., the macro content may actually change between versions making this a very common area for migration issues. Once a macro is incorporated in an application, this application has to be revisited with each new version in order to make sure that the macros were not affected by any structural changes in QlikView and this makes macros extremely heavy in terms of maintenance.

Only a subset of macros will work in a server environment with thin clients (Java, Ajax) since local operations (copy to clipboard, export, print etc.) are not supported, though some of these have a server-side equivalent (e.g. ServerSideExport etc.) that is very expensive in terms of performance with each client effectively affecting the server performance in a negative way.

In conclusion: what we are striving for is a heightened awareness when it comes to macros and what may work with a few thousand records does not necessarily scale very well when macros are involved and the problems tends to manifest themselves and become more serious when larger datasets are involved. It is also important to note that certain events can only be captured through the use of macros and for this reason it may be difficult to avoid macros altogether. The R&D department always strives to incorporate as much of this functionality as possible as basic QlikView functionality, thus limiting the use of macros in the long run – however as previously stated: certain events are difficult to catch except from an outside macro...

Given all of the above, macros cannot be part of any recommended QlikView design pattern!

QlikView Actions

Action has been around since QlikView 9. They are derived from the old button shortcuts, which they also replace. Apart from offering a much wider range of operations than the old shortcuts (including most common operations on sheets, sheet objects, fields and variables), you may also define a series of operations within a single action. The introduction of actions should greatly reduce the need for macros, which is good since macros are never efficient from a performance point-of-view.

Actions can not only be used on buttons. Also text objects, line/arrow objects and gauge charts can be given actions, which are executed when clicking on the sheet object in question.


The trigger macros of previous versions of QlikView have been replaced by trigger actions. This gives you the possibility to build quite elaborate triggers without the use of macros. Trigger macros from previous versions will be automatically translated to a Run Macro action when loaded into QlikView.


Read more about Triggers in the QlikView Reference Manual.

Variable Editor

Overview

Variable Editor is a QlikView application that graphically controls Deployment Framework. System and Custom Global Variables can be added and edit within Variable Editor and all containers are plotted in a Container Map (master is stored in Administration container) this map is edited and containers created using Variable Editor, start by clicking on the Variable Editor Shortcut. If using User Access Control (UAC) right click on the *VariableEditor.cmd* and run as Administrator.

 [VariableEditor Shortcut](#)

 [VariableEditor.cmd](#)

Variable Editor with Qlik Sense

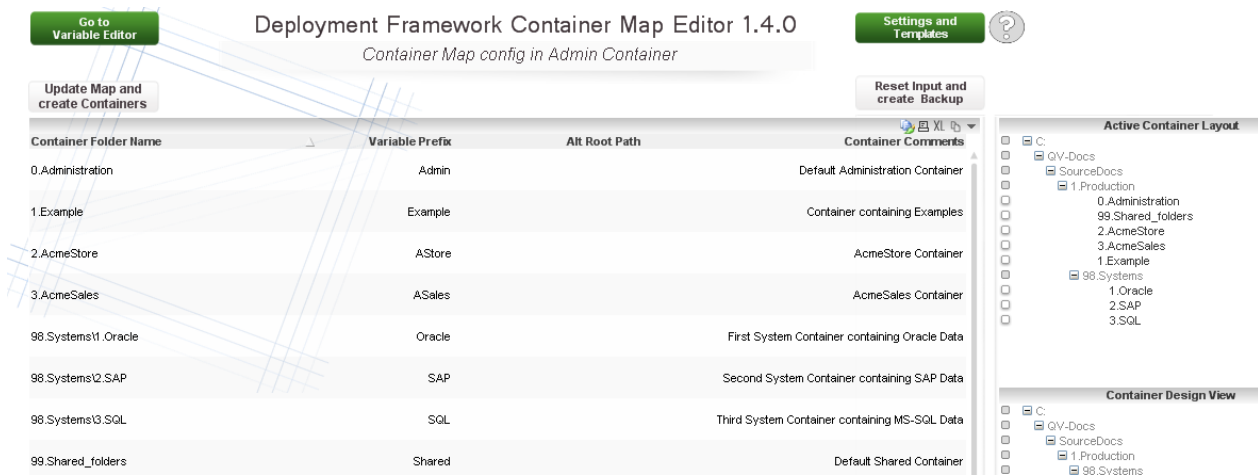
There is for the moment no variable editor for Qlik Sense. As Variable Editor uses an imbedded license it works with QlikView personal edition without the need of a QlikView license, personal edition can be downloaded for free. This means that Qlik Sense administrators can maintain QDF without additional costs.

Help

There is a help button in the VariableEditor available when needed.

Container Map Editor

Is used to administrate and populate containers within the framework. Press Go to Container Map to change to container view. Container Map is used by Deployment framework to identify containers.



| Container Folder Name | Variable Prefix | Alt Root Path | Container Comments |
|-----------------------|-----------------|---------------|---|
| 0.Administration | Admin | | Default Administration Container |
| 1.Example | Example | | Container containing Examples |
| 2.AcmeStore | AStore | | AcmeStore Container |
| 3.AcmeSales | ASales | | AcmeSales Container |
| 98.Systems\1.Oracle | Oracle | | First System Container containing Oracle Data |
| 98.Systems\2.SAP | SAP | | Second System Container containing SAP Data |
| 98.Systems\3.SQL | SQL | | Third System Container containing MS-SQL Data |
| 99.Shared_folders | Shared | | Default Shared Container |

Edit or modify container map in the table, remember that it's only the container Map that is changing not the physical container structure.

Container Input Fields

- *ContainerFolderName* contains the Container folder Name. To create or add in a sub container structure type *folder name\container name*. Example 1: *1.Oracle* to create a container under file root. Example 2: *98.System\1.Oracle* to create a container under 98.System folder.
- *ContainerPathName* , enter prefix share variable names in *ContainerPathName* field, example *Oracle*.
- *Alt root path*, add a container in another file system, add a UNC path in *alt root path* field.
- *Container Comments* Is descriptive Meta Data regarding the containers, very good to use for documenting the solution.

Reset Input and Create Backup

Will reset (revert) all inputs and also create a backup of the Container Map.

Retrieve Backup

Use Retrieve Container Map Backup to get back to the backup stage.

Update Container Map

Use this button to apply the new Container map after adding and/or modifying the container layout.



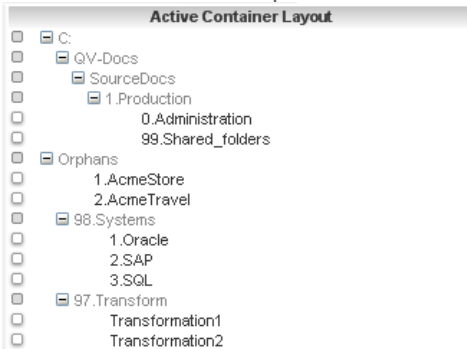
Create New Containers option

Create New Containers will create containers based on the current container Map. This button is only shown after Update Container Map is applied and accepted.



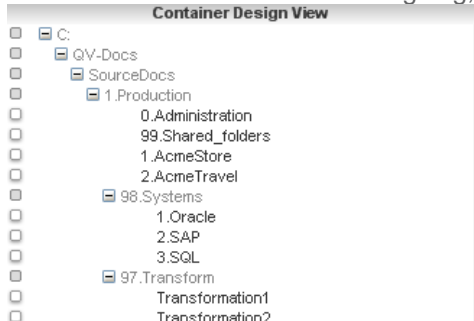
Active Container Layout

Shows physical containers that exist within the Container Map Container that exists in the Map and not in real life will be shown as Orphans as shown in the example below:



Container Design View

Shows the architecture while designing, in this view no Orphans is shown and no reload/refresh is needed.



System Variables

System Variables are actually also Global Variables that start with (*vG.*), the difference is that System Variables are predefined variables used to store system settings like QlikView Server log path. System Variables are also not preloaded, *3.SystemVariables.qvs* include script needs to be run to load in the System Variables into QlikView. System Variables are modified by the Variable Editor and are stored in *\$(BaseVariablePath)\SystemVariables.csv*. There is usually only one System Variable version, the main is stored in 0.Administration container and is by default replicated out to the other containers.

Variable Input Fields

- *VariableName* Type *SET* or *LET* in front of your variable name. Use *vG.* or *vU.* as Global or Universal Variable prefix. Example1 *SET vG.statistics*. Example2 *SET vU.statistics*.
- *VariableValue* Type value or text, when entering text do not use brackets (") this is done automatically. Do not combine numbers and letters when using LET function, use the SET function instead for this.
- *Comments* Used for comments like author and creation date
- *Search Tag* Used only for easy search

Variable Files, Custom Global Variables

Custom Global Variables will automatically be loaded into QlikView applications when using Deployment Framework. Each Container has its own Custom Global Variable file that the applications use. For Global Variables that need to be used across containers modify Shared Custom Variable file with Variable editor.

Refresh Create a Backup

Will refresh the view without updating Variable files and at the same time create a backup.

Retrieve Backup

Use Retrieve Backup to get back to the backup stage created by Change Variable File and Create a Backup button.

Update Variables

Use this button to apply the new variables after adding and/or modifying.

Add and Remove Variable Files

Variable Editor has the possibility to add variable files into the selected container in addition to the default *Custom Global Variables*. Type the variable filename into the *Add Variable File* input box and press enter like example below:

Add Variable File = HR_KPI

The *Refresh and Create a Backup* box will now change to a Create Variable File box

Create Variable File

When pressing apply the new csv file (empty) will be created as *HR_KPIVariables.csv* and stored under selected container *3.Include\1.BaseVariable*.

To remove a Variable File add the command **del** before the filename and run the script like example below:

Add Variable File = del HR_KPI

The box will change to *Delete Variable File*.

Delete Variable File

Variable files other than Custom Variables will not be loaded by *1.Init.qvs* into the applications by default.

Add Sub Function below into the application script instead:

```
$(Include=$(vG.SubPath)\2.LoadVariableCSV.qvs);
```

```
call LoadVariableCSV('$(vG.BaseVariablePath)\HR_KPIVariables.csv', '[Tag]')
```

More detailed documentation on Variable Editor can be found in **Operations Guide**.

Troubleshooting & Support

Support Types

Supporting Qlik applications and environments can be done in several ways.

As a best practice, Qlik recommends that support levels and services be identified for the following areas:

- Qlik Applications (QVW, QVF)
- QlikView Interface (end user support)
- Qlik Sense platform
- QlikView platform (Server/Publisher)
- Qlik Data Architecture

Many clients utilize a certification process for applications of high importance. This can help especially when business teams are creating their own apps and your support team is only responsible for supporting the certified applications that it had a chance to code/interface/data review. Read more in Qlik Deployment Framework-Deployment Guide.pdf.