Qlik Deployment Framework

Qlik Sense Development Guide

February, 2017

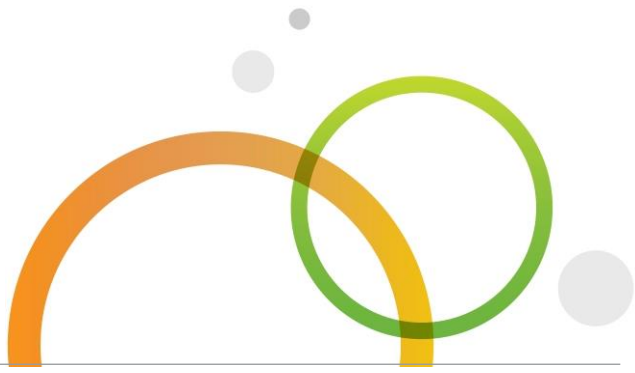


Table of Contents

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***

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
- Security requirements
- Application Lifecycle Management

Platform strategy from a development perspective

When it comes to development, Qlik Sense 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.

Developing using QDF Containers

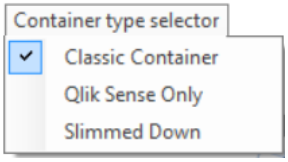
Qlik Deployment framework is based on the concept of **resource containers.** Containers are security boundaries, isolated file structures placed side by side. A container can be moved and/or renamed without changing any Qlik script logic. Each container has identical file structures and contains predefined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

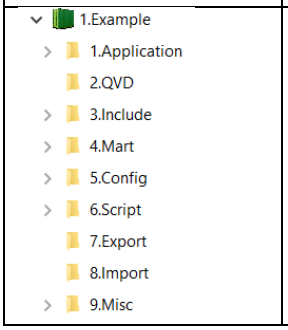
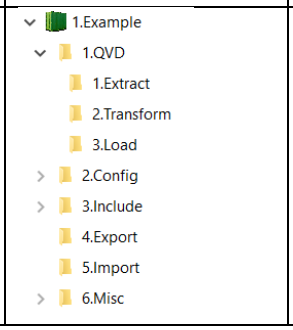
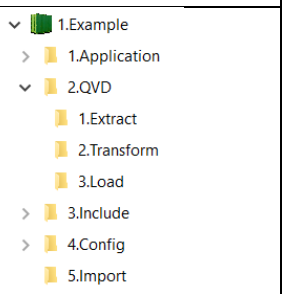
From QDF version 1.6.5 you have the possibility to select between three container layouts. This mean that QDF Deploy tool extracts different container layout (folders) depending on the selection done in the *Container Type Selector*. The different settings and correlating layout are presented below.



- **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
		

Deployment Framework always includes.

- ***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.

Sense only container

Container folders		Global Variables	Description
0.Template			Folder used for examples and templates. *Only exists in the 0.Administration Container*.
1.QVD		*vG.QVDPath*	QlikView Data files are stored in subfolders under 2.QVD
	1.Extract	*vG.ExtractPath*	Extract path for QVD files
	2.Transform	*vG.TransformPath*	Transform path for QVD files
	3.Load	*vG.LoadPath*	Load path for QVD files
2.Config		*vG.ConfigPath*	Configuration and language files like Excel and txt. This folders could be shared to make configuration changes easier
3.Include		*vG.IncludePath*	Folder where QlikView Include files are stored. These are script parts that are called from the main QlikView script.
	1.BaseVariable	*vG.BaseVariablePath*	Stores all the variables needed to use the framework, like paths inside the container
	2.Locale	*vG.LocalePath*	Locale for different regions, used for easy migration between regions
	3.Custom	*vG.CustomPath*	Store for custom include scripts
	4.Sub	*vG.SubPath*	Store for sub routines, this is a way to reuse code between applications
4.Export		*vG.ExportPath*	Folder used to store from QlikView exported data, probably txt or qvx
5.Import		*vG.ImportPath*	Folder used to store import data from external systems
6.Misc			To store documentation, extension and other things related to this container
Info.txt			Information files describing the folder purpose and Path variable. There are Info files in every folder.
Version.xx.txt			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 and reused between applications across a container. The name standard is *vG*.xxx (***Variable Global)***.

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.

Additional Variable 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.Cal**c**_*
- 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* =;

Get started with Qlik Deployment Framework Containers

Qlik applications need to have an initiation include script in the beginning of the load script which identifies the current environment (within your home container) and generates global variables. These variables are used during Qlik Script development.

Initiate QDF in Qlik Sense

To learn how to initiate QDF with Qlik Sense please read ***Qlik Deployment Framework-Qlik Sense Getting Started Guide.pdf*.**

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).

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 above 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 of false depending on if the file exists

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

QlikView Components (QVC)

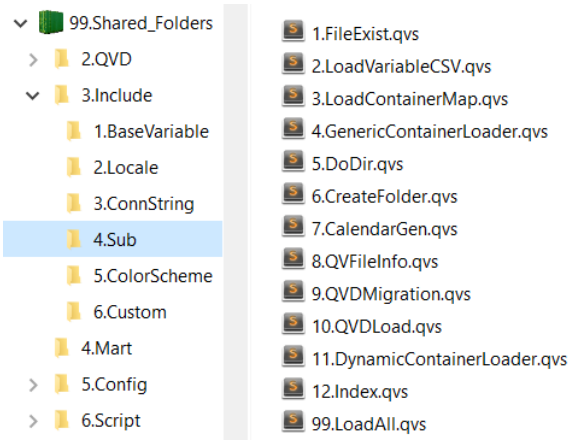
Optionally, the open source library *QlikView Components (QVC)* can be installed using the deploy tool in addition to the built-in QDF library, read more on QVC here. If QVC been installed it can be disabled in the script by adding this line before QDF initiation:

set vG.QVCDisable='true';

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

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.

Qlik Deployment Framework- Function Reference Guide

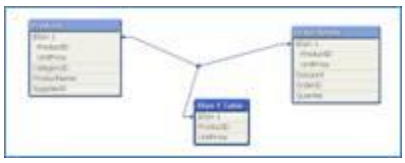
The function reference guide is available as a separate document ***Qlik Deployment Framework-Function Reference Guide.pdf***. As the sub functions are identical to both Qlik Sense and QlikView the same guide applies to both platforms.

Data 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

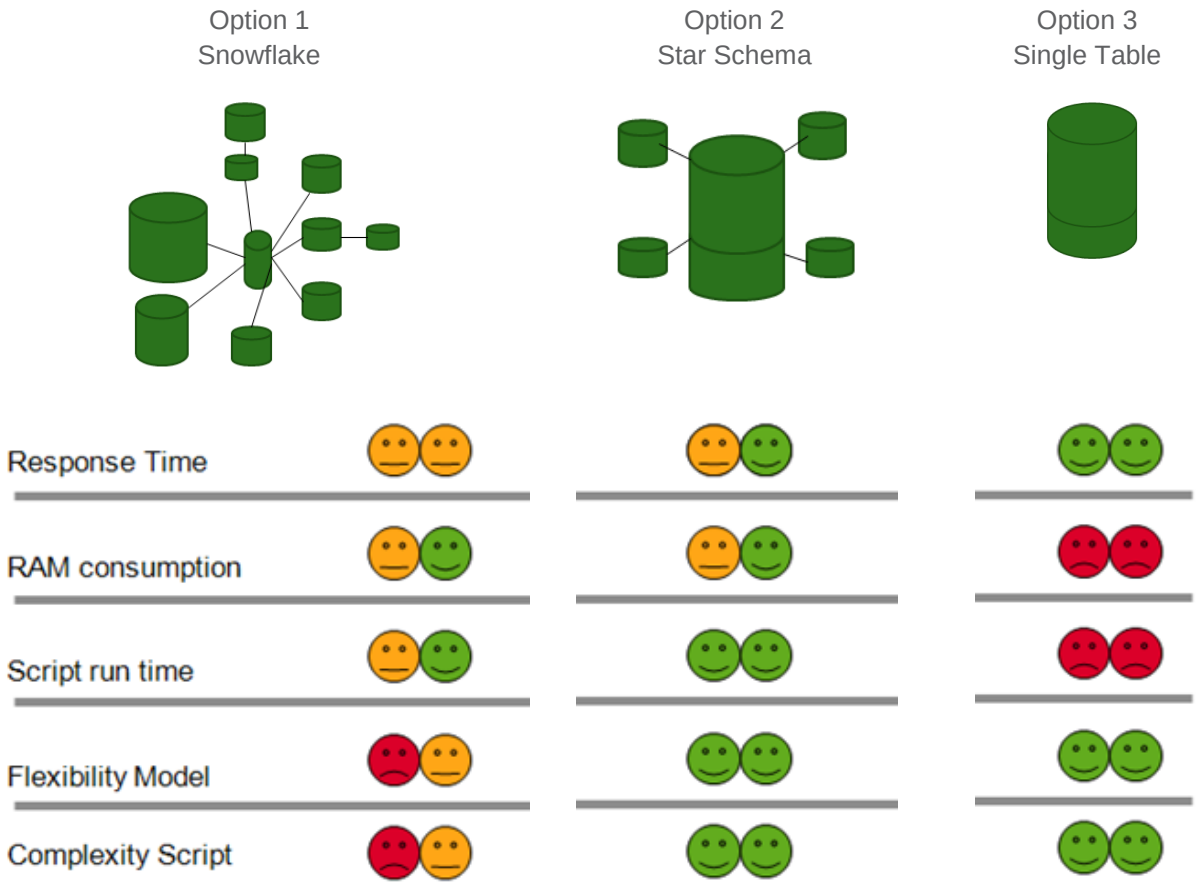
Another characteristic with the associative database is that the number of distinct (unique) values in a table is more important than the number records. By delimit the number of distinct values in a table the performance of an application can be significantly improved.

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 CalendarGen 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)

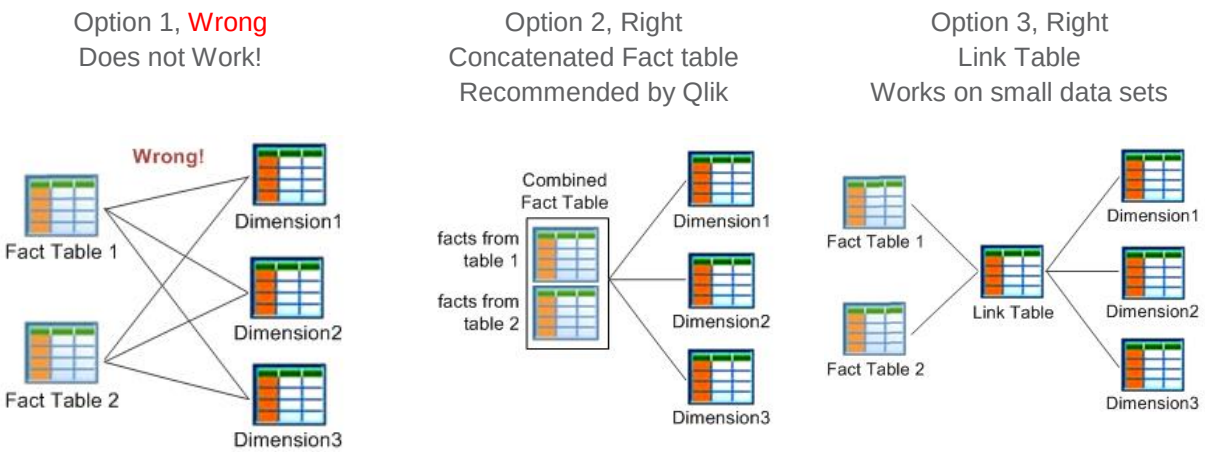
Data models

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.



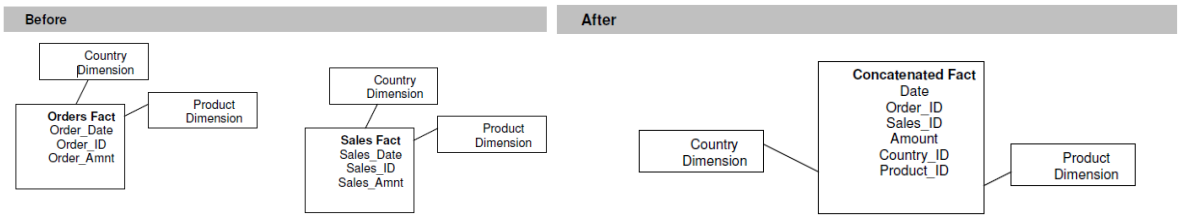
Multiple fact tables

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 Qlik Community 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
Region A	P1	2009-01-31	100
Region A	P1	2009-02-28	120
Region A	P1	2009-03-31	140
Region A	P2	2009-01-31	500
Region A	P2	2009-02-28	550
Region A	P2	2009-03-31	600
Region B	P1	2009-01-31	50
Region B	P1	2009-02-28	55
Region B	P1	2009-03-31	60
Region B	P2	2009-01-31	200
Region B	P2	2009-02-28	180
Region B	P2	2009-03-31	160

Plan Yearly

Region	Date	Plan
Region A	2009-01-1	8000
Region B	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
Region A	P1	2009-01-31	100		
Region A	P1	2009-02-28	120		
Region A	P1	2009-03-31	140		
Region A	P2	2009-01-31	500		
Region A	P2	2009-02-28	550		
Region A	P2	2009-03-31	600		
Region B	P1	2009-01-31	50		
Region B	P1	2009-02-28	55		
Region B	P1	2009-03-31	60		
Region B	P2	2009-01-31	200		
Region B	P2	2009-02-28	180		
Region B	P2	2009-03-31	160		
Region A		2009-01-1		8000	
Region B		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. 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.

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)

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 user 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 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,
```