QlikView Best Practice Guidelines: Development

Version: 0.5 - draft
Date: Jan, 2011
Author: BPN, JCA

Best Practices Guidelines: Development

QlikView

Best Practices Guidelines: Development

Table of Contents

QlikView

Best Practices Guidelines: Development

Introduction

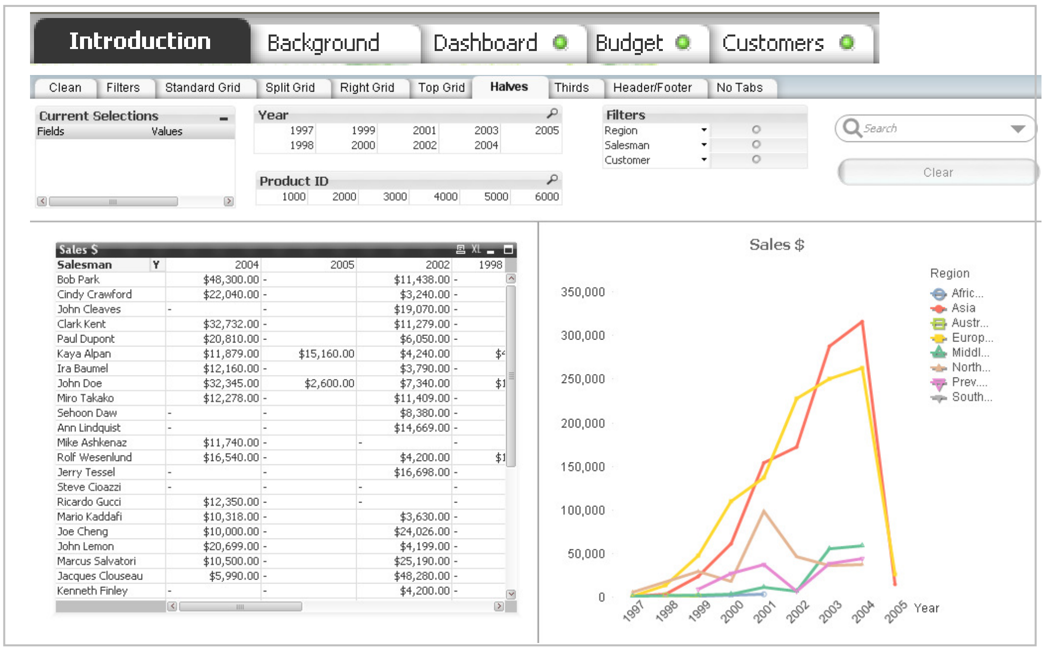
This Best Practices guide is a reference manual for QlikView developers. QlikView developers are individuals who design and implement QlikView applications and their areas of expertise range from data modeling to scripting to UI design. This document is designed to facilitate much clearer understanding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

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 QlikTech recommends when designing and building a QlikView document. The outline below shows (at a high level) some of those tenants of good design.   QlikTech makes many QlikView examples, documents, slide decks and other materials available to help demonstrate these principles.

Examples

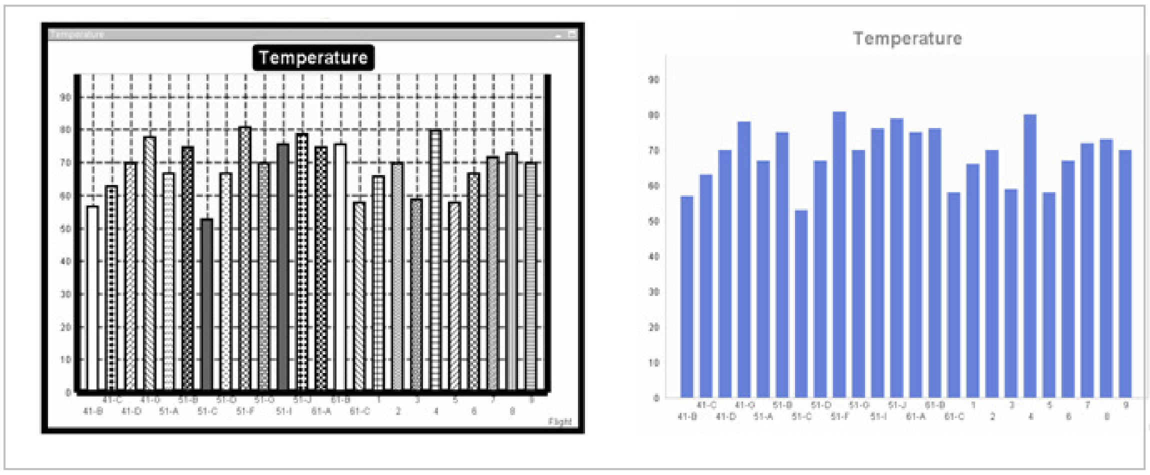
Use of supplied or developed templates for consistency and simplicity:



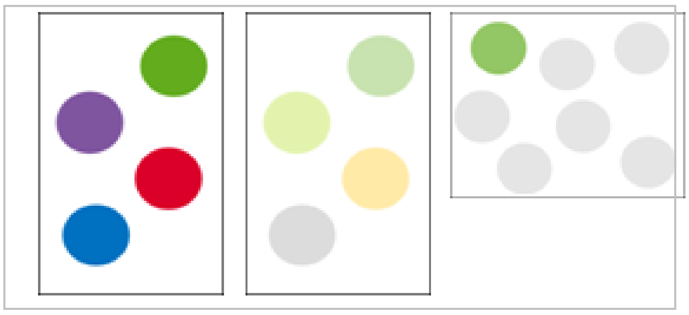
Sales $

Salesman	2004	2005	2002	1998
Bob Park	$48,300.00	-	$11,438.00	-
Cindy Crawford	$22,040.00	-	$3,240.00	-
John Cleaves	-	-	$19,070.00	-
Clark Kent	$32,732.00	-	$11,279.00	-
Paul Dupont	$20,810.00	-	$6,050.00	-
Kaya Alpan	$11,879.00	$15,160.00	$4,240.00	$4
Ira Baumel	$12,160.00	-	$3,790.00	-
John Doe	$32,345.00	$2,600.00	$7,340.00	$1
Miro Takako	$12,278.00	-	$11,409.00	-
Sehoon Daw	-	-	$8,380.00	-
Ann Lindquist	-	-	$14,669.00	-
Mike Ashkenaz	$11,740.00	-	-	-
Rolf Wesenlund	$16,540.00	-	$4,200.00	$1
Jerry Tessel	-	-	$16,698.00	-
Steve Cioazzi	-	-	-	-
Ricardo Gucci	$12,350.00	-	-	-
Mario Kaddafi	$10,318.00	-	$3,630.00	-
Joe Cheng	$10,000.00	-	$24,026.00	-
John Lemon	$20,699.00	-	$4,199.00	-
Marcus Salvatori	$10,500.00	-	$25,190.00	-
Jacques Clouseau	$5,990.00	-	$48,280.00	-
Kenneth Finley	-	-	$4,200.00	-

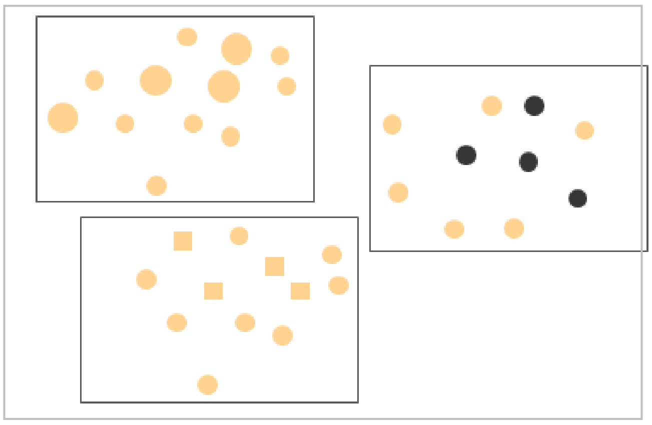
Use of implied closure to limit non-data ink space:



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.



Many of the design best practices are displayed in the demo applications that are publicly available at http://www.demo.qlikview.com. In addition, there is a slide deck presentation covering design techniques for QlikView which is very comprehensive. Please visit QlikCommunity and search for ***DataVisualization.ppt***.

Other best practices for UID Design include:

- 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 hieratical 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 organisations 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)

QlikTech strongly recommends the incorporation of design best practices for all QlikView developers and designers when starting a QlikView 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 QlikView 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.

UI Design References:

- QlikTech Demo Site http://www.demo.qlikview.com
- QlikTech Visual Design Presentation on QlikCommunity ***DataVisualization.ppt***
- 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

Scripting

Overview

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

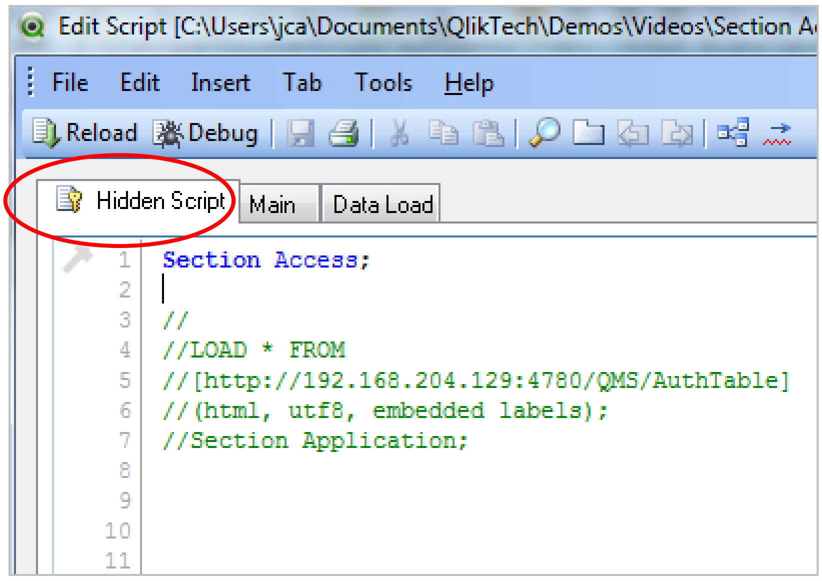
Best practices dictate that using multiple tabs within a script will split out the various parts, enabling a simple view of the information for future development and support. Depending on the complexity of the application, you may have a variety of different script sections. The common parts of a script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

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

Security Tab (Hidden Script)

In QlikView it is possible to restrict the privileges of a document user from the ***Document Properties: Security*** and the ***Sheet Properties: Security pages***. Any settings can be altered if the document user is logged in as ADMIN.

The user identity and password needed for opening a user restricted document are specified in the load script and will show up in the log file if you allow QlikView to generate one. However, by having the user access in the hidden script instead, the log file will not give away any login information. The Hidden Script button opening the hidden script is found in the Edit Script menu.


Edit Script [C:\Users\jca\Documents\QlikTech\Demos\Videos\Section A
File
Edit
Insert
Tab
Tools
Help
Reload
Debug
Hidden Script
Main
Data Load

```
Section Access;

//
//LOAD * FROM
//[http://192.168.204.129:4780/QMS/AuthTable]
//(html, utf8, embedded labels);
//Section Application;
```

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;
```

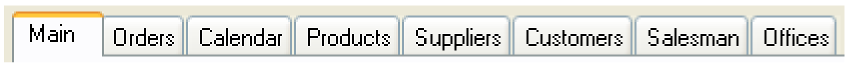
This will simplify the SQL SELECT statement so that the developer can continue to test/augment the statement using other tools, without the complexity of the QlikView transformations embedded in the same SQL statement.

For more information on the Preceding LOAD feature, see the QlikView Reference Manual.

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 the QlikView 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

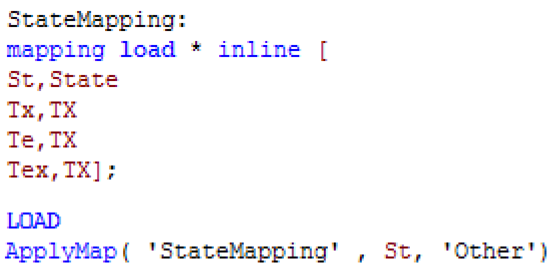


- Name the concatenate/join statements

- When adding script to a QVW, it is best to do a binary load on large data sets then extend the script. Later merge the script after development is near complete. This doesn't functionally change anything, but it saves time during development.

- 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 the QlikView 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 the QlikView Reference Manual if you are not sure how/when to use Qualify and Unqualify.

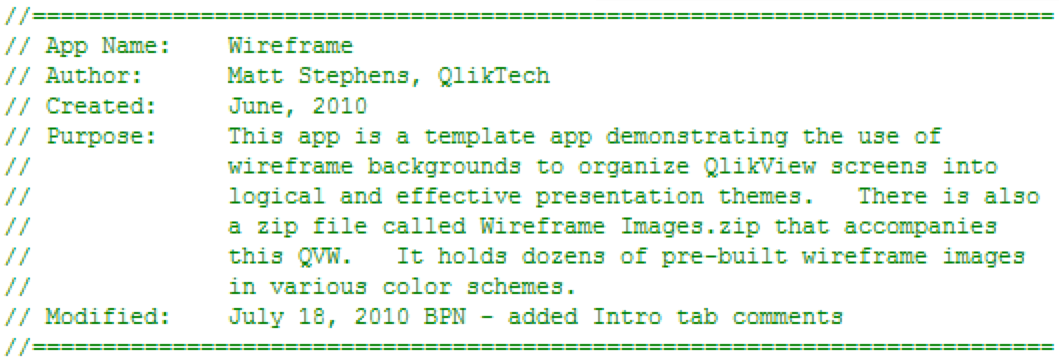
- Use “Include” files or hidden script for all ODBC/OLEDB database connections.

- 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 UNC naming convention. Do not use C:\MyDocs\...

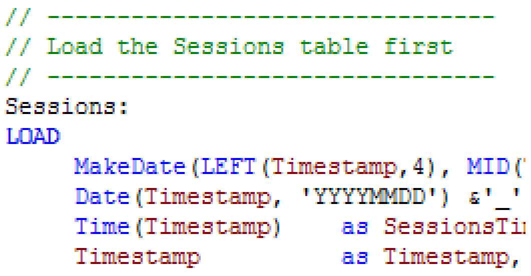
- Always have the Logfile option turned on if you need to capture load-time information for degbugging purpose

- Comment script headings for each tab. See example below:



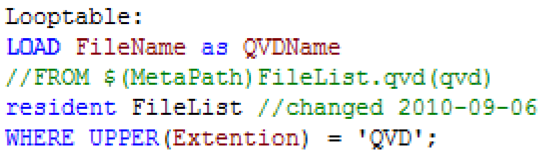
```
//==============================================================================
// App Name:    Wireframe
// Author:      Matt Stephens, QlikTech
// Created:     June, 2010
// Purpose:     This app is a template app demonstrating the use of
//              wireframe backgrounds to organize QlikView 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:



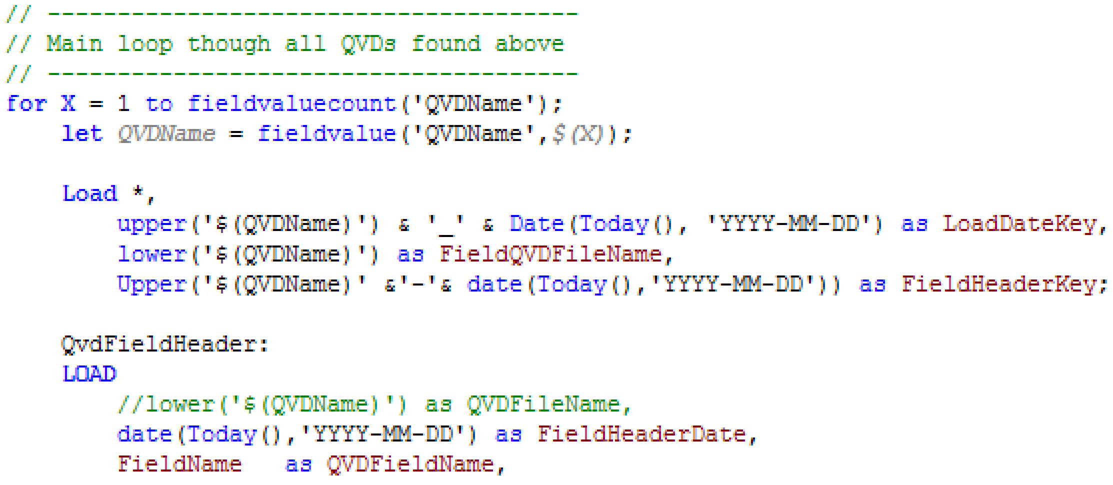
```
// ---------------------------------
// Load the Sessions table first
// ---------------------------------
Sessions:
LOAD
     MakeDate(LEFT(Timestamp,4), MID('
     Date(Timestamp, 'YYYYMMDD') &'_'
     Time(Timestamp)     as SessionsTi
     Timestamp           as Timestamp,
```

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



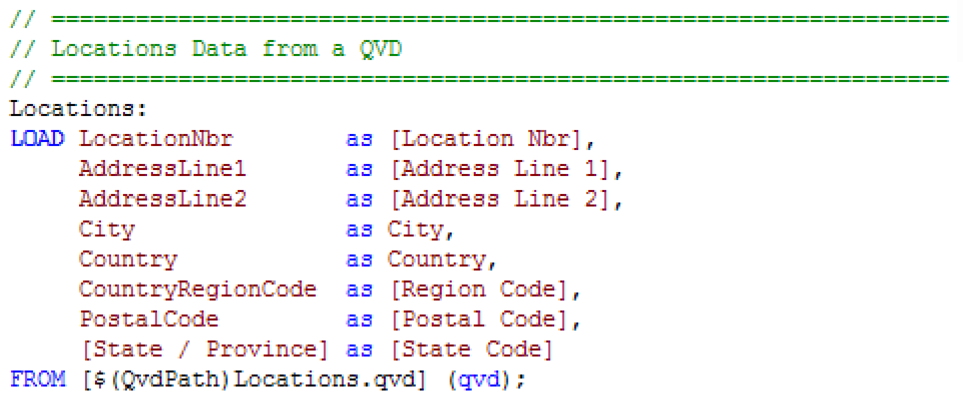
```
Looptable:
LOAD FileName as QVDName
//FROM $(MetaPath)FileList.qvd(qvd)
resident FileList //changed 2010-09-06
WHERE UPPER(Extention) = 'QVD';
```

- 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,
        FieldName   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:



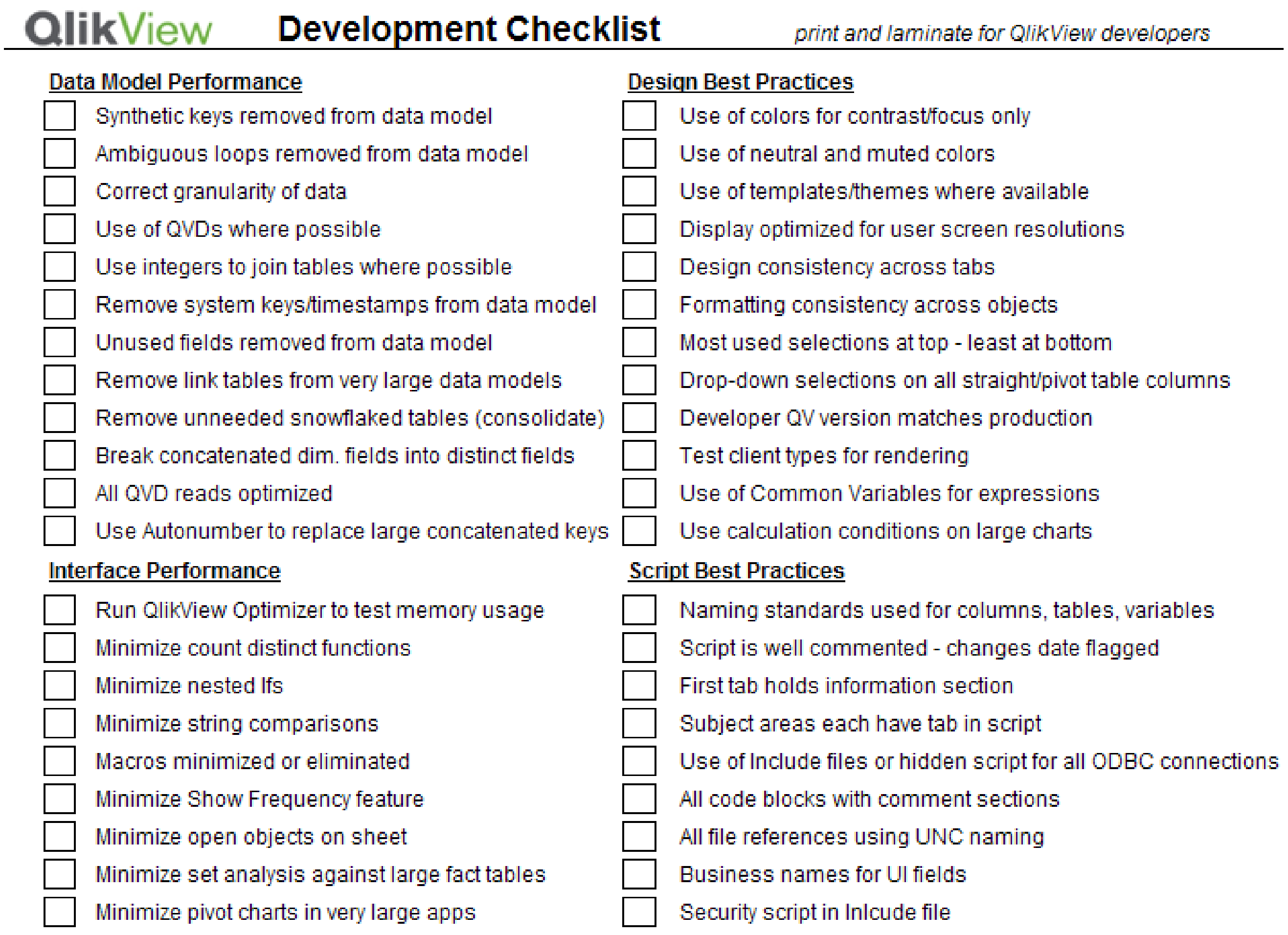
```
// ================================================================
// Locations Data from a QVD
// ================================================================
Locations:
LOAD LocationNbr        as [Location Nbr],
     AddressLine1       as [Address Line 1],
     AddressLine2       as [Address 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);
```

Development Checklists

QlikTech recommends the use of a developer checklist to highlight and reinforce development best practices.

Most enterprise clients develop this from a template or sample of best practices. Consult your Account Executive or Regional Services Director for a sample from QlikTech. One way to help promote the visibility and presence of the checklist is to limit it to one page and laminate it for each developer. This will make it easier to post the checklist and refer to it often. Some clients will use the checklist in code reviews to ensure that best practices were followed before releasing a QVW to Test or Production environments.

A screenshot sample of a checklist is below:



QlikView **Development Checklist** *print and laminate for QlikView developers*

Data Model Performance

- ☐ Synthetic keys removed from data model
- ☐ Ambiguous loops removed from data model
- ☐ Correct granularity of data
- ☐ Use of QVDs where possible
- ☐ Use integers to join tables where possible
- ☐ Remove system keys/timestamps from data model
- ☐ Unused fields removed from data model
- ☐ Remove link tables from very large data models
- ☐ Remove unneeded snowflaked tables (consolidate)
- ☐ Break concatenated dim. fields into distinct fields
- ☐ All QVD reads optimized
- ☐ Use Autonumber to replace large concatenated keys

Interface Performance

- ☐ Run QlikView Optimizer to test memory usage
- ☐ Minimize count distinct functions
- ☐ Minimize nested Ifs
- ☐ Minimize string comparisons
- ☐ Macros minimized or eliminated
- ☐ Minimize Show Frequency feature
- ☐ Minimize open objects on sheet
- ☐ Minimize set analysis against large fact tables
- ☐ Minimize pivot charts in very large apps

Design Best Practices

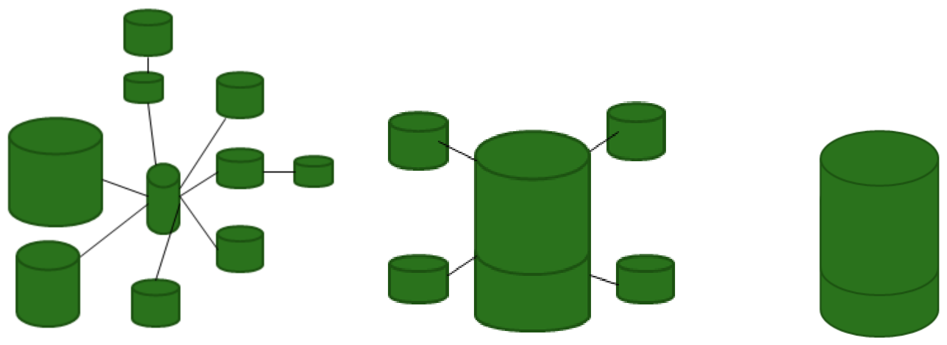
- ☐ Use of colors for contrast/focus only
- ☐ Use of neutral and muted colors
- ☐ Use of templates/themes where available
- ☐ Display optimized for user screen resolutions
- ☐ Design consistency across tabs
- ☐ Formatting consistency across objects
- ☐ Most used selections at top - least at bottom
- ☐ Drop-down selections on all straight/pivot table columns
- ☐ Developer QV version matches production
- ☐ Test client types for rendering
- ☐ Use of Common Variables for expressions
- ☐ Use calculation conditions on large charts

Script Best Practices

- ☐ Naming standards used for columns, tables, variables
- ☐ Script is well commented - changes date flagged
- ☐ First tab holds information section
- ☐ Subject areas each have tab in script
- ☐ Use of Include files or hidden script for all ODBC connections
- ☐ All code blocks with comment sections
- ☐ All file references using UNC naming
- ☐ Business names for UI fields
- ☐ Security script in Inlcude file

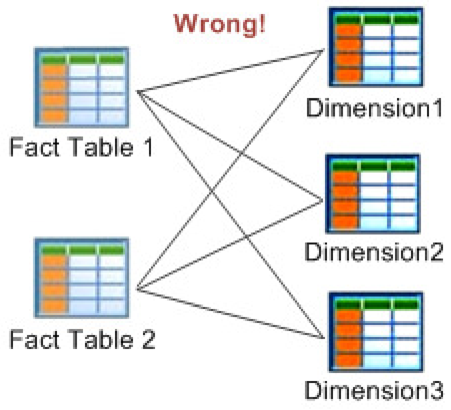
Data Models

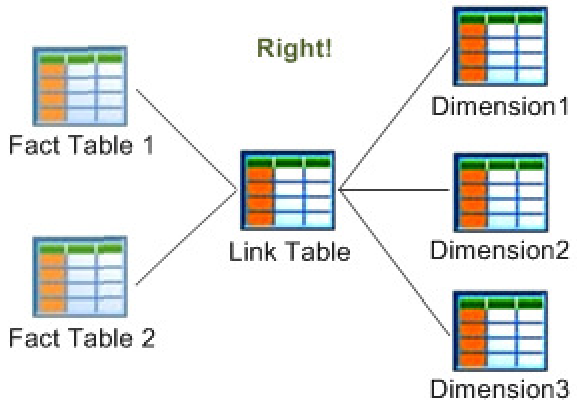
Represented below are diagrams of 3 basic data models that can be built in QlikView (along with many other combinations). Using these 3 examples we can demonstrate some of the differences in performance, complexity and flexibility between them.



	Option 1 Snowflake	Option 2 Star Schema	Option 3 Single Table
Response Time	😐😐	😐🙂	🙂🙂
RAM consumption	😐🙂	😐🙂	🙁🙁
Script run time	😐🙂	🙂🙂	🙁🙁
Flexibility Model	🙁😐	🙂🙂	🙂🙂
Complexity Script	🙁😐	🙂🙂	🙂🙂

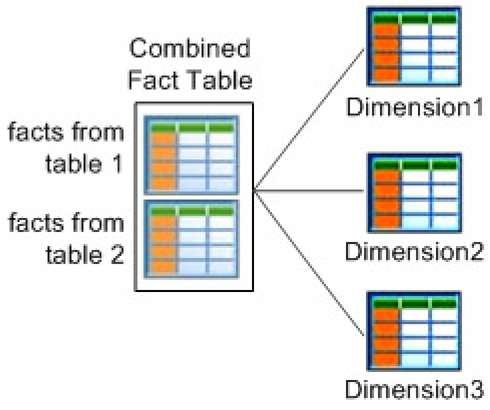
While star schemas are generally the best solution for fast, flexible QlikView applications, there are times when multiple fact tables are needed. Here are the wrong and right ways to join them:





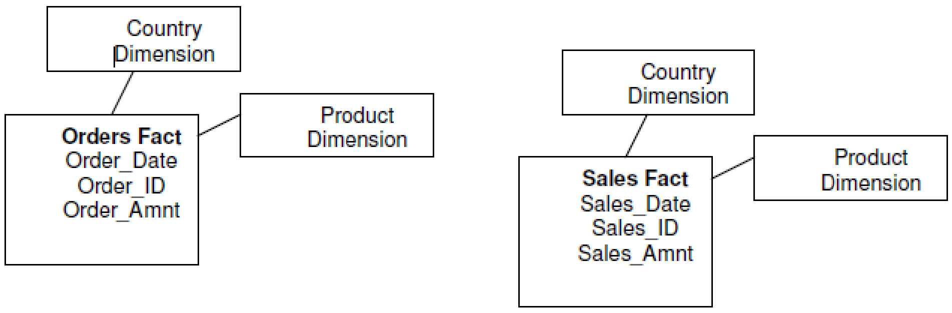
Further examples of how to build and use link tables are contained in QlikCommunity on line (http://community.qlikview.com/)

In addition to modeling for multiple fact tables, an alternative is to concatenate the two fact tables into a single fact table.   This is illustrated below.

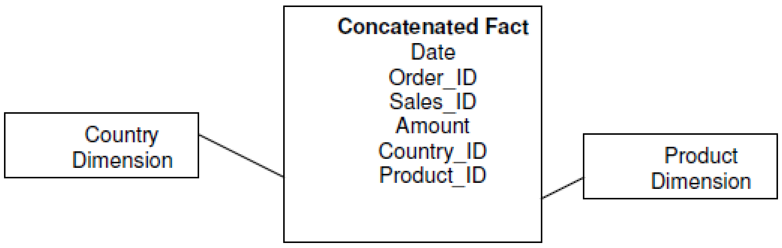


To show how this could be accomplished, the section below takes us through a scenario of two facts tables to be combined into one fact table.

Before



After

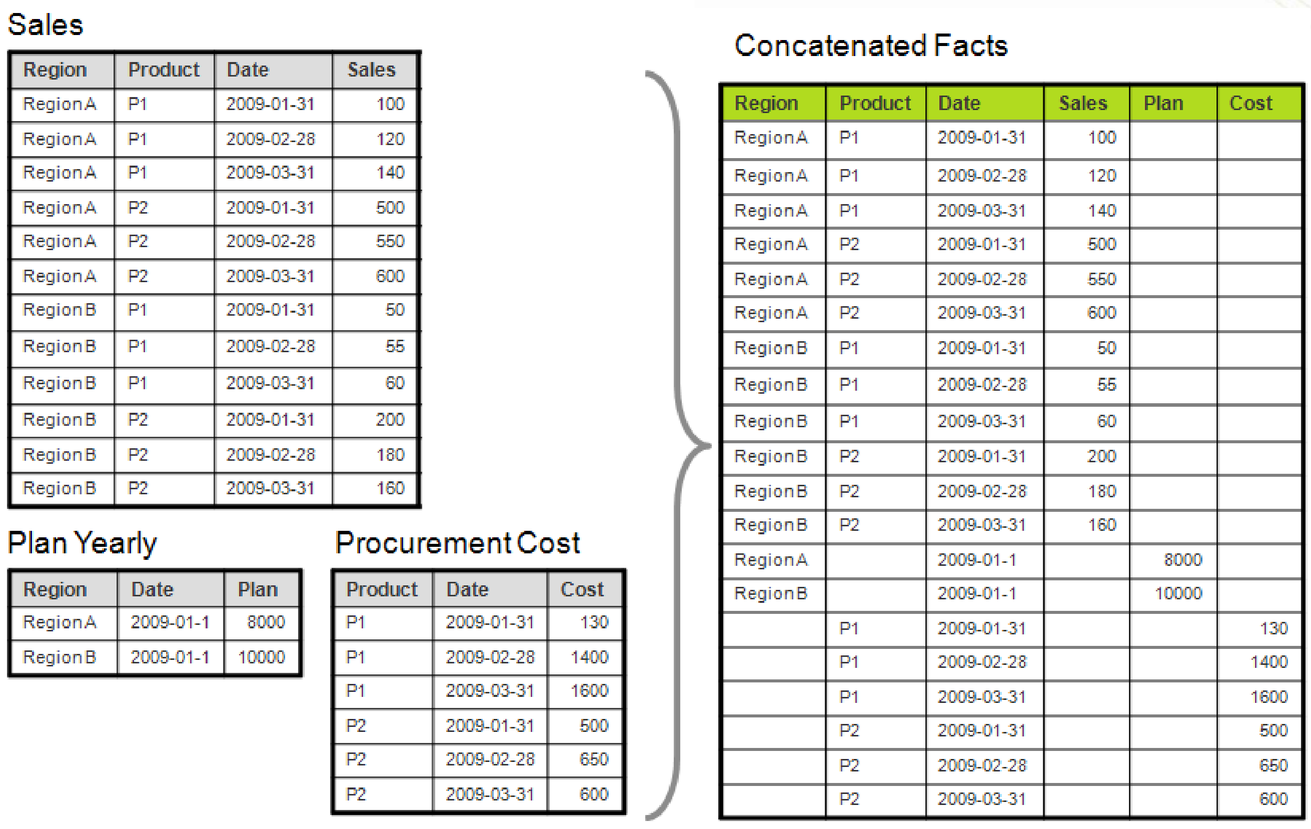


Script Example:

```
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.

A table example of this concatenation of fact tables is shown below.



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

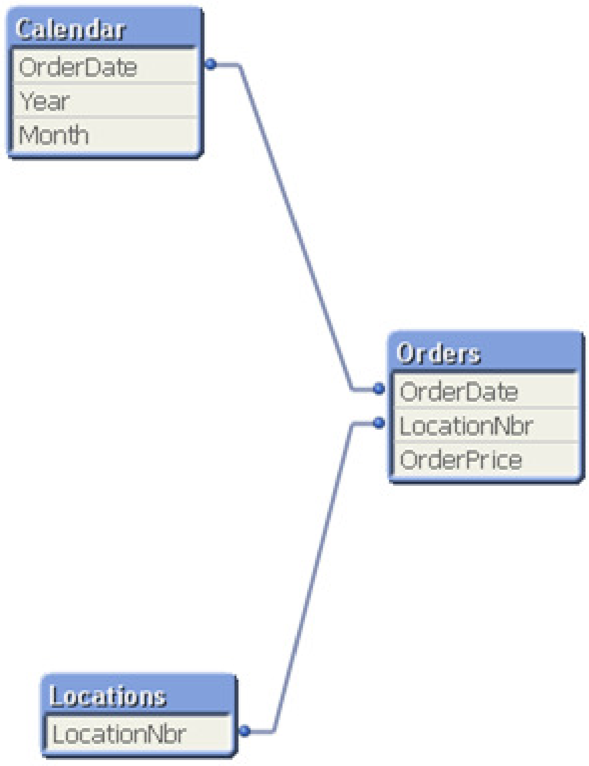
Region	Product	Date	Sales	Plan	Cost
RegionA	P1	2009-01-31	100		
RegionA	P1	2009-02-28	120		
RegionA	P1	2009-03-31	140		
RegionA	P2	2009-01-31	500		
RegionA	P2	2009-02-28	550		
RegionA	P2	2009-03-31	600		
RegionB	P1	2009-01-31	50		
RegionB	P1	2009-02-28	55		
RegionB	P1	2009-03-31	60		
RegionB	P2	2009-01-31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

Large Data Sets

QlikView can handle very large data sets and routinely does so. However, to optimize the user experience and hardware needed, you have options.

Consider the following scenario: You have a large orders data set (1 billion rows). You need to provide high level summary metrics for your executives, trending analysis for your Business Analysts, and detail tables and values for your Orders Processing team. You have many data design options with QlikView, but for demonstration purposes let's explore just 3 of them below:

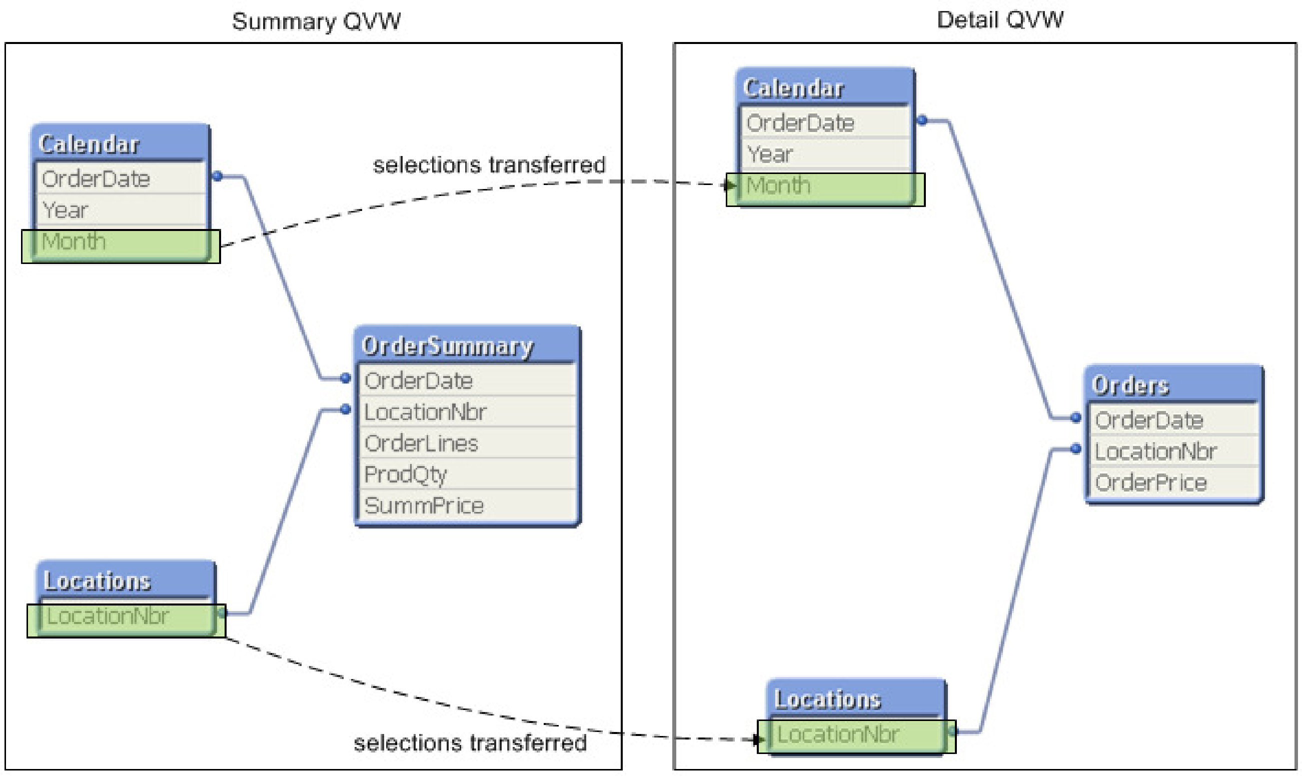
1) Detailed fact table only – allow QlikView to do all of the work to display the details and summarize metrics from the lowest level of detail to the highest summary needed.
    a. Advantages – simplicity. This is the easiest solution to code. You simply connect the Orders at a detailed level (perhaps SKU level) to the data model and design all of the high level metrics, trending charts and detailed tables and selections into the QVW.
    b. Disadvantages – QlikView will need to aggregate up to 1 billion rows of detail with every selection made. While QlikView is probably the only BI tool that can do this with acceptable performance, it will still result in a slower user experience than it needs to.



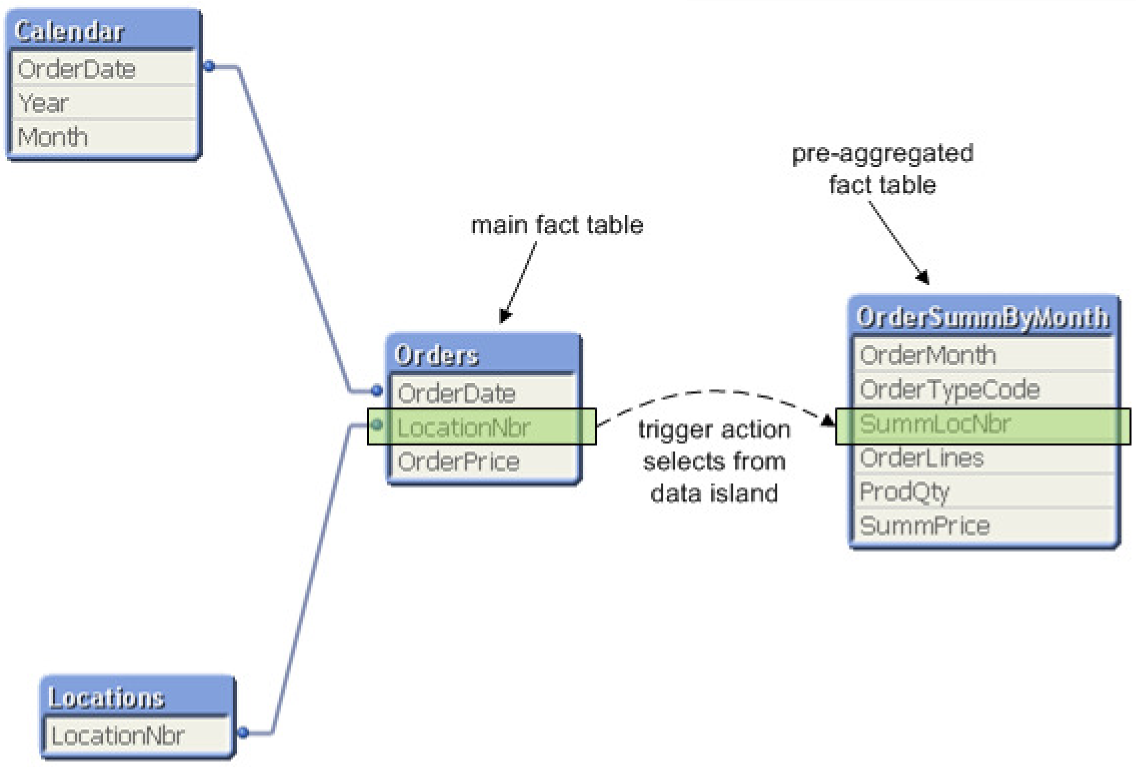
2) Document Chaining – 2 (or more) versions of the QVW are built. One of them has the detailed Orders table as the primary fact table, the others have pre-aggregated versions of the Orders table as their primary fact tables. Let's assume just 2 QVWs for this case. You have a diagram below showing the data model from the "summary" QVW and a data model from the "detail" QVW.   Note that the dimension values are largely the same between the two models.   The main distinction is the fact table in the data model. The users can start from the summary application, showing high level metrics and charts.

   If they want to drill into details you can use the Document Chaining feature in QlikView to transfer selections from one QVW to another QVW and open that second QVW. The user will see new charts and tabs show up and (if you design it as such) doesn't even need to know they have trasferred from one QVW to another. This means you will only be using the 1 billion row fact table ***when your users need it***. The rest of the processing will take place on the pre-aggregated version of the Orders table, which might be smaller than 100 million rows, for example. Document Chaining is discussed in detail in the QlikView Reference Manual and in several QlikView documents.

   a. Advantages – optimizes hardware and speed of response for QlikView navigation and charting.   Because the users' selections and navigation are specific to their needs, you don't waste CPU and RAM processing 1 billion rows of detail when the user didn't need things processed at that level.
   b. Disadvantages – tables (QVDs) need to be pre-aggregated and maintained for this approach.   While this is a one-time development effort, it is slightly more complex than option 1, where only one version of the Orders table is needed.



3) The 3rd option (and by no means the last) is to use a pre-aggregated summary table ***in addition to*** the detailed table in a single QVW data model. The diagram shown below is one way to use a pre-aggregated table in the same data model as the detailed version of the table. You would load the pre-aggregated table as a data island (not connected to the other tables in the data model). Then, as relevant selections in the detailed fact table are made you can transfer those selections to the pre-aggregated table using a triggered Action (QlikView version 9 and above).
    a. Advantages – this option doesn't require a second QVW and document chaining in order to use both detailed and summary versions of a large table.
    b. Disadvantages – this option will require some settings to be made in the QVW to trigger the actions that transfer selections from one table to another. As the QVW changes over time, you will need to keep track of where/when to make these actions trigger.



Please note: these are many more ways you could meet the needs described in the above scenario. These are just 3 mothods that call out the features and capabilities of QlikView to manage very large data sets. Please see the Architecture Best Practices Guide for more examples of ways to manage large data sets and large deployments of QlikView in an optimal way.

Key factors that affect the model:

- Distinct column data.
- Distinct key field information.

Both can affect the memory size of the Data Model and the user experience. By having many tables, the links can become a memory hog.
It has been known that you can reduce your memory foot print by fifty percent when modifying the data structure; and thus, additionally increasing the UI response.

See the Optimization section of this document for helpful hints on reducing the size and complexity of your data model.

Variables, Actions and Macros

Variables

(Following is taken from a blog post: http://www.quickqlearqool.nl/?p=902)

In this post I want to share with you a good practice in handling the various expressions that exist in a QlikView document. The most used expressions are the ones used in charts, where they hold measures such as Sum(Sales), Sum(Price*Quantity), etcetera. These are the ones more likely to be reused by other objects and in different sheets. There many other expressions including Chart Attributes, Color Expressions and Show Conditions, you can see them all by going to the menu Settings/Expression Overview.

The most used expressions are the ones used in charts, where they hold measures such as Sum(Sales), Sum(Price*Quantity), etcetera. These are the ones more likely to be reused by other objects and in different sheets. There many other expressions including Chart Attributes, Color Expressions and Show Conditions, you can see them all by going to the menu Settings/Expression Overview.

The use of expressions can be intensive in QlikView, especially when having a sophisticated user interface. There is a growing need to handle these expressions in a more efficient way, and this can be accomplished by the use of variables.

Reasons for holding expressions in variables:

- To achieve reuse: the formula for a measure such as Sales usually remains the same across a QlikView document, 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.

Variables can be created manually by going to the menu Settings / Variable Overview or by using the SET/LET statements in the script. They have a name and a value, which can hold any sort of strings or numbers, and they can be used as a reference from every sheet object. The Input Box is the object designed to show variables in the user interface.

If you want to start experimenting with moving your expressions to variables try the following:

1. Go to the Expressions tab on one of your charts and copy one of the formulas, for instance Sum(SalesValue)

2. Go to the menu Settings / Variable Overview and click on the “Add” button to create a variable. Give it a name such as vFormulaSales (it is a best practice to have all variable names starting with a v to help differentiate them from Fields).

3. Select your variable from the variable list an paste the formula from the chart in the “Definition” text box. If the formula starts with an = sign, remove it. Finally click on “OK” to save the changes.

4. Go back to the Expressions tab of your chart properties and replace the formula with the following: $(vFormulaSales)

The $ sign expansion indicates the string contained in the variable is a formula that needs to be calculated.

The final step is to replace replace the cloned formulas in all the other objects so they all refer to the same formula contained in the new variable. Every new object that needs to show Sum(Sales) should also refer to the variable.

You may already have quite a few QlikView documents where you didn’t apply this practice, but it’s never too late to get started. In the long term it’s really worth it.

< end blog content >

Variables are commonly used to help switch the database settings between environments without hard coding required in the QVW as it moves from environment to environment. See the sample code below for a best practice technique for doing this:

```
SET vEnvironment= ‘PROD’;

IF vEnvironment = ‘PROD’  THEN
···ODBC CONNECT TO MyOracleDBProd (XUserID is *****, Xpassword is ****)
                    SET vDBName = ‘MyOracleDBProd‘;
ELSEIF vEnvironment = ‘TEST’  THEN
···ODBC CONNECT TO MyOracleDBTest (XUserID is *****, Xpassword is ****)
                    SET vDBName = ‘MyOracleDBTest‘;
ELSE
···ODBC CONNECT TO MyOracleDBDev (XUserID is *****, Xpassword is ****)
                    SET vDBName = ‘MyOracleDBDev‘;
END IF
```

In your LOAD statements you now reference the vDBName as follows:

```
SQL SELECT *
FROM $(vDBName).MySchema.MyTable;
```

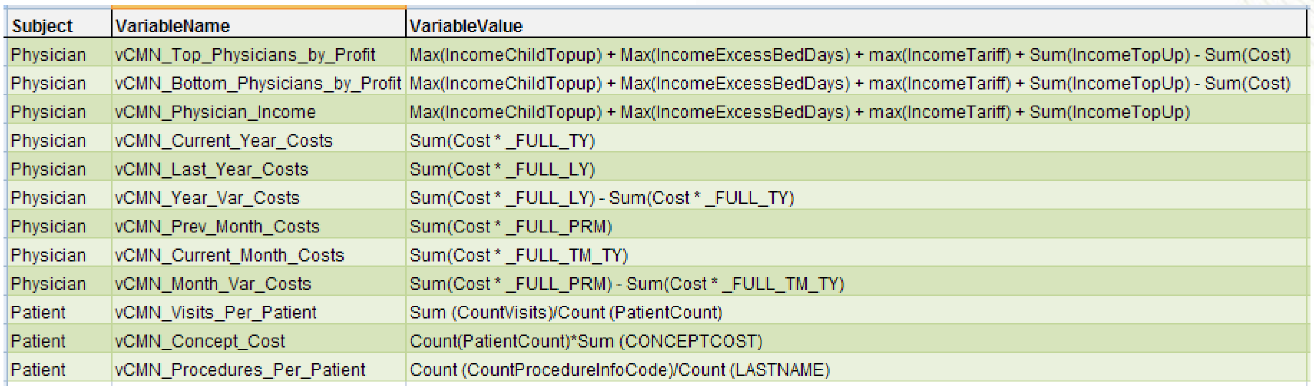
There are two simple methods for changing this variable value from environment to environment as the QVW gets promoted:

1) Force the developer or Admin to manually change the variable value in the script

2) Use an Include file with the SET vEnvironment…. statement in it. Each environment has its own Inlcude statement text file that stays in the environment. The QVW will load in the include file that exists in its directory, thereby always getting the proper variable set for its environment.

Variables can also be used to store common expression (metric) logic and used across many QlikView documents. The expression logic can be stored in Excel, a flat file, or in a database.

The sample below shows some expressions stored in an Excel file related to Physician metrics.

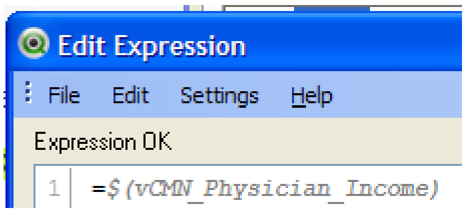


Subject	VariableName	VariableValue
Physician	vCMN_Top_Physicians_by_Profit	Max(IncomeChildTopup) + Max(IncomeExcessBedDays) + max(IncomeTariff) + Sum(IncomeTopUp) - Sum(Cost)
Physician	vCMN_Bottom_Physicians_by_Profit	Max(IncomeChildTopup) + Max(IncomeExcessBedDays) + max(IncomeTariff) + Sum(IncomeTopUp) - Sum(Cost)
Physician	vCMN_Physician_Income	Max(IncomeChildTopup) + Max(IncomeExcessBedDays) + max(IncomeTariff) + Sum(IncomeTopUp)
Physician	vCMN_Current_Year_Costs	Sum(Cost * _FULL_TY)
Physician	vCMN_Last_Year_Costs	Sum(Cost * _FULL_LY)
Physician	vCMN_Year_Var_Costs	Sum(Cost * _FULL_LY) - Sum(Cost * _FULL_TY)
Physician	vCMN_Prev_Month_Costs	Sum(Cost * _FULL_PRM)
Physician	vCMN_Current_Month_Costs	Sum(Cost * _FULL_TM_TY)
Physician	vCMN_Month_Var_Costs	Sum(Cost * _FULL_PRM) - Sum(Cost * _FULL_TM_TY)
Patient	vCMN_Visits_Per_Patient	Sum (CountVisits)/Count (PatientCount)
Patient	vCMN_Concept_Cost	Count(PatientCount)*Sum (CONCEPTCOST)
Patient	vCMN_Procedures_Per_Patient	Count (CountProcedureInfoCode)/Count (LASTNAME)

These variables are then read into QVW files using a simple LOAD statement and then converted to variables using this logic below:

```
// ---------------------------------------------------------------------------
// This code converts the variables from table values to document variables
// ---------------------------------------------------------------------------
Let RowCount = NumMax(NoOfRows('CommonVariables'),0)-1;
For i=0 to '$(RowCount)'
  Let TempVarName = peek('VariableName',$(i),'CommonVariables');
  Let TempVarValue = peek('VariableValue',$(i),'CommonVariables');
  Let $(TempVarName) = '$(TempVarValue)';
Next
```

Once this is done the variables can be used in any expressions in the QVW. An example of the expression logic to utilize the variable is shown below.



Expression OK

```
=$(vCMN_Physician_Income)
```

This method allows for central management of the logic in metrics. You can simply change and test logic enhancements in a spreadsheet or database and then allow the QVWs to reload the logic the next time they are triggered for reload.

Macros

The following are some reflections you should be aware of when you start including macro statements in your application.

Running a macro automatically deletes all caches, undo-layout buffers and undo logical operation buffers and this in general has a very large negative im-pact 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 mac-ros, 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 gen-eral 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 sta-bility, the macros will always maintain their poor performance and the gap be-tween genuine QlikView functionality and the macros will continue to in-crease, 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 functional-ity - 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 con-sider 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 applica-tion 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. Server-SideExport etc.) that is very expensive in terms of performance with each cli-ent 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 in-volved. 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 mac-ros 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!

Actions

Action is a new entity in 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.

The new 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 RunMacro action when loaded into QlikView.

Read more about Triggers in the QlikView Reference Manual.

Project Management

Overview

The recommendations in this section are intended to be explored and decided upon before the project starts, not during the implementation phase.

SCRUM Methodology works well with QlikView. Some important factors are

- Since QlikView projects are very rapid, SCRUM's methods of frequent project meetings work well with QlikView development

- A Notification method must be set up between concurrent developers when one of them are changing shared objects

- Define Processes for:
    - QA
        - What denotes an 'Error' when performing QA?
            - Incorrect data/totals are errors
            - Incorrect labels/descriptions are errors
        - What denotes an 'Enhancement'?
            - Changes to the layout (adding, changing items) are enhancements if the item/sheet already passed the initial acceptance by the end user
        - *It's important to denote between Errors and Enhancements because Errors must be fixed, Enhancements must get approved before they are implemented. We try to stay away from enhancements during QA since it may require us to re-QA a lot of work.
    - Change Requests
        - What should we do when a user asks to change items? When do we have to ask permissions?

- Communication and Execution plan
    - When will the key stakeholders meet to go over scope changes, or enhancement requests?

DSDM methodology

- Based on RAD methodology
- One of the Agile methods; part of the Agile Alliance
- Similar to SCRUM in process and concept, HOWEVER:
    - Less jargon than SCRUM
    - No education into SCRUM roles and titles
    - Fewer documents required
- Globally recognized Agile RAD methodology
- Iterative and incremental
- Emphasis on continuous user involvement

Best Practices Guidelines: Development

- Focus on “on time, on budget” time-boxed, scope consciousness
- Adjustments for changing requirements built in to schedule
- Easily folded into over-arching customer projects and PMO’s
- “Plain language” project effort, roles, and documentation.
- Cyclical back to additional sales & revenue opportunities

RAD/DSDM Methodology

- RAD = Rapid Application Deployment
- DSDM = Dynamic Systems Development Method
  - RAD and DSDM methodologies highly recognized in North America and Europe as part of Agile Project Management Alliance
  - RAD and DSDM methodologies fit the typical QlikView project profile with minimal modification
  - RAD and DSDM methodologies can be referenced and researched to aide project governance and templates
  - RAD and DSDM methodologies provide a foundation for knowledge transfers
  - RAD and DSDM methodologies are resource requirement and documentation lean yet complete

RAD/DSDM Elements

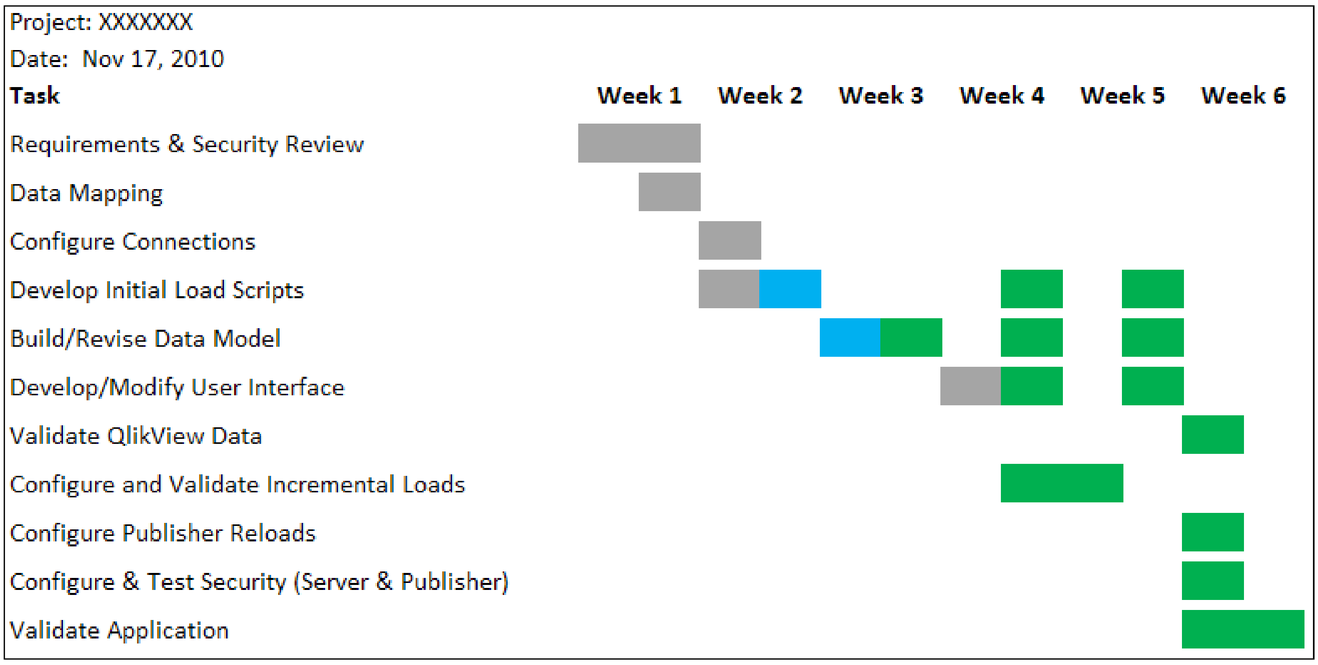
Project Phases	**3 Phases:** • **Pre-project** • **Project Development Life-cycle** • **Post-project**
Project Team Resource Roles	**6 Roles** • **Project Owner/Sponsor** • **Technical Analyst** • **Project Manager/Business Analyst** • **Expert Services Consultant** • **QlikView Service Partner Developer** • **Customer Project Team**
Documents Required	***8 Documents*** • **Project Charter with Scope** • **Requirements: Business, Functional, Non-functional, Technical** • **Test Plan & Summary** • **Project Schedule & Plan** • **Design & Development Summary** • **Knowledge Transfer & Support Summary** • **Team Post-project Interview Summary** • **Customer Satisfaction Interview Summary**

Engagement Document	***"Project Charter"*** **Single Document with tables** **Documentation:** • **Charter Purpose, Executive Summary, Project Overview, Scope with goals, objectives, and deliverables, Conditions with assumptions, communication plan, issue tracker, risk tracker, constraints, and escalation path, Structured Approach, Team Organization Plan, Team Contact Directory** • **Appendix Documents: project schedule (spreadsheet), SOW, change requests, milestone summaries (requirements, design & develop, test, deployment)**
Project Schedule	**Excel spreadsheet exportable to MS Project, et al project management software, based on MS Project formats**

QlikView

Best Practices Guidelines: Development

Shown below is a sample project plan for a QlikView project. QlikTech recommends that a project plan be created and followed for QlikView projects, and that the help of a qualified Project Manager be sought out for larger projects. Many more templates and samples are available via QlikTech Expert Services or online through QlikCommunity.



Project: XXXXXXX

Date: Nov 17, 2010

Task	Week 1	Week 2	Week 3	Week 4	Week 5	Week 6
Requirements & Security Review	▨▨					
Data Mapping	▨					
Configure Connections		▨				
Develop Initial Load Scripts		▨🟦		🟩	🟩	
Build/Revise Data Model			🟦🟩	🟩	🟩	
Develop/Modify User Interface				▨🟩	🟩	
Validate QlikView Data						🟩
Configure and Validate Incremental Loads				🟩	🟩	
Configure Publisher Reloads						🟩
Configure & Test Security (Server & Publisher)						🟩
Validate Application						🟩🟩

Security (Section Access)

Section Access can be set up in the QlikView script to handle security. It is contained within the .qvw file, meaning the one single file can be made to hold the data for a number of users or user groups. QlikView will use the information in the Section Access for Authentication and Authorization and dynamically reduce the data, so that a user only sees data that he/she is allowed to.

QlikView provides the following access levels:

ADMIN - can change everything in the document. Using the **Security** page in the **Document Properties** and **Sheet Properties** dialogs, a person with ADMIN access can limit the users' possibilities of modifying the document.
USER - cannot access the Security pages.
NONE - optionally used for clarity, always treated as "no access".

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 a userID, password or both. This is especially important if the QVW is going to be enabled for download from the AccessPoint or otherwise distributed to users.

QlikTech recommends that QVWs that will be distributed should be password protected, or at least validated against a userID with Section Access.

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
- Security in include files
- 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 'snowflaked' dimensions 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 degredation on the server.

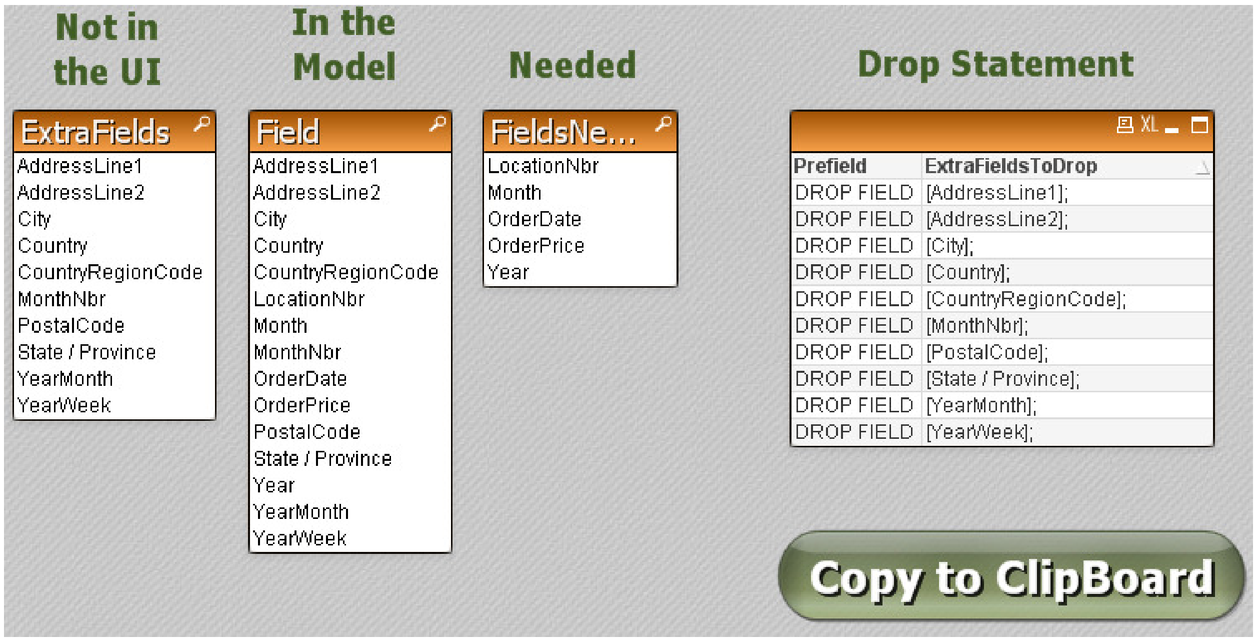
Optimization

Overview

When the development phase on an application is complete and the deployment phase begins, it's very important to consider the best practices for optimizing the application's footprint so that the end user experience is smooth and seamless. This section discusses the best practices optimizing data and the handling of expressions.

Data Load

After the load, drop any unnecessary fields. An unnecessary filed is one which isn't currently used in charts, list boxes, etc. Something which isn't currently being used. There are utility QlikView applications available on QlikCommunity that will identify the unused columns in your QVW and even generate the DROP statements needed to eliminate them from your data model. A screenshot of the UnsedFields.QVW is shown below. This QVW is available on QlikCommunity for download and free use.



ExtraFields	Field	FieldsNe...
AddressLine1	AddressLine1	LocationNbr
AddressLine2	AddressLine2	Month
City	City	OrderDate
Country	Country	OrderPrice
CountryRegionCode	CountryRegionCode	Year
MonthNbr	LocationNbr	
PostalCode	Month	
State / Province	MonthNbr	
YearMonth	OrderDate	
YearWeek	OrderPrice	
	PostalCode	
	State / Province	
	Year	
	YearMonth	
	YearWeek	

Prefield	ExtraFieldsToDrop
DROP FIELD	[AddressLine1];
DROP FIELD	[AddressLine2];
DROP FIELD	[City];
DROP FIELD	[Country];
DROP FIELD	[CountryRegionCode];
DROP FIELD	[MonthNbr];
DROP FIELD	[PostalCode];
DROP FIELD	[State / Province];
DROP FIELD	[YearMonth];
DROP FIELD	[YearWeek];

- If your data is huge, decide to break it up into multiple timeframes. For example: Have a "This year vs Last year" QVW, a second which has 5 years data (or more), since most people want to see this year vs. last year. (if that is the case). Note: this also results in a much faster end user experience. Let's assume that you discover that 80% of your users only look at the last 13 months of data, yet your QVW holds 60 months of data. If you create two versions of the QVW (one that holds only the last 13 months and another that holds all 60 months) you can allow users to first analyze the 13-month version of the QVW and then link to the other version only when needed. This will result in 80% of your end user sessions consuming a fraction of the RAM, CPU and processing time that they had before you split the application. This makes your end users' experience better and stretches your hardware further.

- Don't normalize data too much. Plan for 6 – 10 total tables in a typical QlikView application. This is just a guideline, but there is a balance to be struck with QlikView 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.

- Eliminate Count(Distinct x)'s They are very slow

- Eliminate Count Numbers, or Count Texts, they are almost as slow as Count(Distinct)

- Store anything possible as a number instead of a string

- De-normalize tables with small numbers of fields

- 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

- Use Incremental Load template to load incrementally and break the historical .qvd files into individual .qvds based on the incremental timeframe

- Always use relative paths when referencing for files

- Use UNC names, or automated tasks might not be able to reference the paths

- Local User Preferences in an include file
    - Include files for connections

- 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)

- (a – b) / b better: (a / b) – 1

- date(max(SDATE,'DD.MM.YYYY')) is factor xxx faster than max(date(SDATE,'DD.MM.YYYY'))

- Is the granularity of the source data needed for analysis? "sum() group by"

- Use numeric flags (e.g. with 1 or 0) , which are pre-calculated in the script

- sum(Flag * Amount) vs. sum(if(Flag, Amount))

- 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)

- De-activate Hyperthreading within server BIOS; Hyperthreading (only Intel-CPUs) can slow down script processing

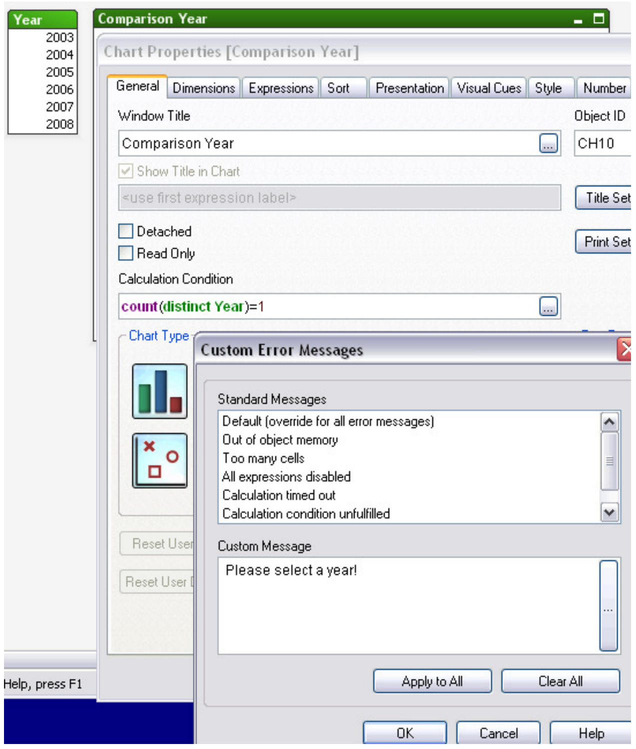
- Be very carefully using Macros!

- For very large QVWs you can further optimize by pre-caching selections in RAM. QlikView stores a result of a regular formula calculation within diagrams into the shared cache memory. The same user or also another user fetches the result from the cache when the formula and the filters are the same, i.e. the result is delivered instantly without any processing. The cache entries remain in the assigned cache memory till the QV-model is reloaded e.g. after an update. This means that the first users after a reload have to accept waiting time, because the cache is empty. This issued can be solved via a Visual Basic script (VBS) which simulates user selections in the application and which can be started automatically (via an external execution task in Publisher) after the update of the data model.

  This VBS sample below runs through all folders of the application, opens all diagrams and selects all fields in the dimension Region

```
set x = CreateObject("QlikTech.QlikView")
set doc = x.OpenDoc("qvp://BISERVER/xyz/Value_Management_Dashboard.qvw")

loop_through_objects(doc)
doc.CloseDoc
x.Quit

'I run through the application. Called by IOpenTheDocument
Sub loop_through_objects(doc)

  For i = 0 to doc.NoOfSheets - 1

      doc.ActivateSheet i
      ' Selection-loop thru the field Region
      Set val=doc.Fields("Region").GetOptionalValues
      For y=0 to val.Count-1

       INH=val.Item(y).Text
       doc.Fields("Region").select INH

       Objects = doc.ActiveSheet.GetSheetObjects

       For j = lBound(Objects) To uBound(Objects)
          set table = Objects(j)
          select case Objects(j).getObjectType
          case 1,4,10,11,12,13,14,16,20,21,27
           On Error Resume Next
            CellRect = doc.GetApplication().GetEmptyRect()
           CellRect.Top = 0
           CellRect.Left = 0
           CellRect.Width = table.GetColumnCount
           CellRect.Height = table.GetRowCount
           set CellMatrix = table.GetCells( CellRect )
           For RowIter = 0 to CellMatrix.Count-1
           Next
          End Select
       Next
      Next
      doc.Fields("Region").clear
  Next

End Sub
```

1. Use Calculation Condition expressions to limit the calculation of very large tables when it's not relevant. The calculation of diagram objects – especially with many complex formulas – can cause a significant system load and hence some waiting time, when no filter is set. It can make sense forcing the user to select a year, a product category or a region or all of these dimensions before the calculation of the diagram objects starts.



Year: 2003, 2004, 2005, 2006, 2007, 2008

Comparison Year

Chart Properties [Comparison Year]

General | Dimensions | Expressions | Sort | Presentation | Visual Cues | Style | Number

Window Title: Comparison Year

Object ID: CH10

[✓] Show Title in Chart

<use first expression label>

[ ] Detached

[ ] Read Only

Calculation Condition: count(distinct Year)=1

Custom Error Messages

Standard Messages:
- Default (override for all error messages)
- Out of object memory
- Too many cells
- All expressions disabled
- Calculation timed out
- Calculation condition unfulfilled

Custom Message: Please select a year!

Condition examples:

- Count(distinct *dim-field*)>1
- not isnull(only(*dim-field*))
- getselectedcount (*dim-field*) >0
- vSelectDim1 and vSelectDim2
  where vSelectDim1 = not isnull(only(dim-field1))
  and vSelectDim2 = not isnull(only(dim-field2))

2. Utilize variables and/or Set Analysis instead of complex data calculations in expressions. Using time functions within expression (1-3 below) results in waiting time 3-15 times longer compared to a simple comparison (4 below) or to Set Analysis (5 below).

1. sum(if(inmonth (Date,date(max(total Date)),-12), Sales)) ☹
2. sum(inmonth (Date,date(max(total Date)),-12) * -1 * Sales) 😐
3. sum(if(inmonth (Date, vPYMonthEnd,0), Sales)) ☹
4. sum(if(Date>= vPYMonthStart and Date <= vPYMonthEnd, ☺ Sales))
5. sum({$<Date={">=$(vPYMonthStart) <= $(vPYMonthEnd)"}>} ☺ Sales)

These optimization best practices are demonstrated and practiced in the QlikView Developer and Designer training courses. QlikTech strongly urges clients to take advantage of this training in order to optimize QlikView deployments and maximize the return on investment they can realize with QlikView.

QlikView

Best Practices Guidelines: Development

Code Management & Migration Guidelines

Code Management

In order to provide an overview of Code Management best practices, it's helpful to look at a fictitious case study that examines a QlikView application change process for a company. The fictitious company's name is Acme Industries.

Report: *Acme Industries' Change Request Guide*

Changing the QlikView App –Release Process

We expect that end users will have enhancement requests and/or defect reports for the QlikView application. We will manage such change requests using standard Acme Industries' software change management procedures where possible.

Several change requests can be implemented and rolled up into a new release. The frequency of new releases is not yet determined, but once the application is stabilized, new releases are more likely to be a quarterly or monthly event, rather than a daily event.

Opening a Change Request

The change request should first be entered into Acme Industries change request tool or bug tracking system. The change request should then be prioritized by the QlikView Project Manager. As more change requests come in, the Project Manager assigns the highest priority change requests to a QlikView Developer for implementation.

Implementing a Change Request

1. Check-out of QVW File (s)

The QlikView Developer then checks the appropriate QVW file(s) from Acme Industry's revision control tool. (*Alternatively, instead of checking out the QVW file(s), the script and layout files can be exported from the QVW(s) via QlikView Developer for management by the preferred revision control tool*).

2. Changing the QlikView application

The QlikView application developer now extends and/or repairs the QlikView application, using skills from the QlikView Designer and Developer courses. The QlikView application can be modified and extended locally on a developer's desktop, or remotely on the test server. The following table compares the two different approaches:

Development Location	Desktop Software Required	Server Software Required	Notes
Locally on Desktop	QlikView Desktop	n/a	Server-based development is useful for larger data volumes.
Remotely on Test Server	Windows Remote Desktop Connection Client	QlikView Desktop	

*Typically, it is good practice to use separate hosts for **test** and **production** purposes for large deployments. However, one machine can serve in both roles if need be, especially in smaller deployments. In the case of one machine serving in both roles, **test** and **production** files can be stored in separate directory trees, and the systems administrator can control access to the **production** directory tree using Windows NTFS file permissions. The rest of this document discusses the use of separate test and production hosts; however, the same concepts could be applied when using separate test and production directory **trees** on a single machine.*

3. Test Deployment on Test Server

The QlikView developer now copies the updated QVW file to the test server's QlikView server folder and validates that the application works as desired when accessed by the preferred client type (ex QlikView's Internet Explorer plug-in).

4. Test Reload on Production Server

The QlikView developer works with the server administrator to validate that the updated QVW file can reload data successfully on the production server. This test is especially critical if the QlikView developer has added calls to any new data sources in the updated QVW file, as those data sources might not be available from the production environment.

5. Data Reduction before Check-In

In software development, the goal of a revision control system is to control changes to software application artifacts themselves, and not to the data processed by the application. As such, before checking in the new version of the QVW file(s), all data should be removed from the QVW files, via the QlikView Desktop **File > Reduce Date > Remove All Values** command.
This will reduce the size of the QVW file from about 2 MB down to 200 kb. We are not worried about checking in this file with no data, as the QVW file will be re-populated with data after being deployed to the production server, when its load script is run.

This step is only relevant if Acme is performing revision control on the QVW file(s); it is not relevant when using the alternative approach of performing revision control on the script and layout files.

6. Check-In and Deployment

The developer should check in the updated, reduced QVW file to the revision control system, and then alert the system administrator to the availability of the updated file. The system administrator then copies the updated QVW file(s) to the **Production** directory on the server**.**

If Acme is using the alternative approach of performing revision control on the layout and script files (instead of the QVW), then the developer should check in the modified script and layout files, and the system administrator would then import those updated script and layout files into the production QVW files.

Closing the Change Request
The change request should be marked as closed in the Acme Industries change request tool or bug tracking system.

This concludes the typical report on implementing a change control solution for QlikVIew.

Migration Guidelines

This section describes the migration path of changes to production QlikView documents from the developer and professional users into production. This will need to be a coordinated effort between several people to make the process work correctly.

Following is a discussion of the various roles that are used in this section:

Roles

Role	Responsibilities
Administrator	Publisher nightly jobs (Creating, modifying, maintaining) Acting as a gateway for changes (source control) Integrating New changes from Developers/Professional users into Production
Developer	This is generally a person who develops the data model and works with the source systems
Professional User	This is generally a person who creates/modifies the charts/reportsand layout
QA User	This is the end user who helps define report requirements and are the final validation that the reports are correct. This could be the developer or a person outside of the development team.

Best Practices Guidelines: Development

Definitions

Document : A QlikView QVW file. AKA 'QVW'

Production Candidate: A document which has all of the latest changes applied to it. This is generally a document which is in the hands of the administrator. It's the administrators responsibility to keep track of which document is the production candidate and there should only be one production candidate per document at any time.

Definitions of changes

Changes would include anything which requires a developer to open the document, edit it and save the changes. This includes changes in layout (moving things around), changing colors, adding/modifying properties on anything (list boxes, charts, graphics, etc), changes in scripting, cycle and drill groups, macro code.
This reference of changes do not include updating the data nightly through an automated process (such as publisher)

Overview

The basic process is that a developer will make a few changes to the layout of the document and want to merge those changes into production. Once the developer has verified the changes are correct, they will notify the administrator. The administrator will integrate the changes from the developer (and possibly multiple developers) and put the document in QA for the developer to review.

If the developer and end user has verified the document is correct, then the administrator can move the changes into production.
If a large number of changes are occurring, the administrator may (in coordination with the all parties involved) develop a release schedule and release many changes at once.

End User Communication Process

A Process will need to be developed to notify end users users that new features are available in a specific application. In other words, this should be a process to notify the end users that new features are available once the development has been finished and the application is promoted to production. This could vary depending on the size of the change. If it's a simple new report, then no notification may be necessary. If it's a change to a report which is used all of the time by many people, then you may want to notify everyone with email or have a training session with the changes.

Release Strategy

If there will be several changes from multiple developers going into production on the same document, then a release strategy should be developed to coordinate the changes to minimize the number of promotions to production and additional overhead of promoting new features to production (QA, administrative time to merge changes, possible chance for errors during promotion, etc)

Details for Developers

These details are the general steps each developer would take to develop new changes and have them added to production.

Once a developer receives a request to develop a change to a production document there are 2 ways to proceed:

1. If there are a large number of changes which affect a large number of areas of the document
   a) The developer or professional user would contact the administrator and request the most current version of the document be given to the developer for exclusive access.
   b) The developer would make the changes by
      i. Copying the document which the administrator gives you to your working directory
      ii. Making the changes to the document
      iii. Copying the new document to the e:\QlikView Development Server Directory using the appropriate name (See the "Development Processes.doc") to allow QA users to validate the changes.
      iv. When the QA users have validated the changes, return the document to the administrator with
         i. 1. A general list of changes. (An all-inclusive list of changes wouldn't be necessary since the administrator would take the document from the developer, validate some standards (See Details for Administrator Section of this document) and put the document directly into production.)
         ii. 2. A list of QA users who should QA the document once it's placed into the QA directory. The administrator will grant access to the document for the specific list of QA users.
      v. The administrator would put the document into production after some basic checking that the application work correctly in the production environment and that the proper standards are being met (script isn't broken, all of the reports work correctly)

2. If there are small changes, or the changes are limited to a specific subset of the document
   a. The developer would copy a version of the document from the "E:\QlikView Copy of Production" to their working directory and develop changes. It's very important to keep a list of all of the changes which were made.
   b. The developer must copy the document to the directory "e:\QlikView Development Server Directory" (On the Development Server) using the appropriate name (See the "Development Processes.doc") and allow QA users to validate the changes.
   c. At this point the QA user would validate the changes and the developer would notify the administrator of

i. Where the document is located
ii. The name of the document (exact file name)
iii. Changes the administrator would need to merge into production
iv. Which users should see the document once it's placed into QA

d. The administrator would merge the changes of one or more developer's changes into a production build; place the build in the "E:\QlikView QA Server Directory" for QA by the developer and QA users and notify the developers and QA users of the name of the document and that it's ready for QA (The name could be different than what the developer had originally since the administrator may have merged in changes from several developers)
e. Once the developer (and optionally the QA user) has had time to QA the document, the administrator would promote it into production

Details for Administrator

Your role will encompass keeping the production environment organized and also to merge new changes into production. These changes include changes to script, reports and Publisher jobs. To perform this role, you will need to have a firm grasp on all of the QlikView products since you will be managing the flow of data through all of them.

General principles for all tasks

- Traditional SDLC practices should apply where the development process goes through a Development – QA – Production sequence to promote changes to production. Since QlikView doesn't have features for showing the difference between 2 different documents, comparing changes and promoting changes to production is a manual process.

- When merging in new changes from developers you'll need to make sure you keep track of which document has all of your most recent changes and merge developer's changes into that most recent document.

- You will need to avoid having multiple versions of the same document in QA at the same time. The reason for this is that if you have multiple versions of the same document in QA then when you migrate the changes to production you will have to re-merge the changes which means you need to change things which means additional QA (since changes could affect other changes)

Admin Responsibilities

- Coordinate changes into production
- Fix system problems
- Operate with good software design principles (backups, quality checks, etc)
- Enforce coding standards/practices

- Coach developers as to design principals to promote a better system
- Monitoring/tuning the QlikView Server
- Monitor changes to security groups
- Managing User CALs
- Raising issues with navigation (although they could be overruled if there is a business need)

Admin Responsibilities do not include

- Validating the numbers/data for accuracy
- Managing the QlikView projects
- Deciding what 'looks' good

Merging in new Reports (Layout changes):

When a developer notifies you of changes they should provide you with

- A location and name of the document which has the changes for you to promote to production
- A list of the changes. These changes could include
  - Layout objects (Graphs, reports, list boxes, etc)
  - Variables which the reports reference
  - Cycle and Drill objects
- A list of people who will QA the document

The process is to have a production candidate version of the document which you will work with and the newly enhanced document which the developer gave you (possibly multiple developers)

- The admin would go through the list of changes and move the changes to the new production candidate Many times this can be a simple copy/past of the new object to the production candidate and deleting the old object.
- Once you migrate the changes to the new document you will need to validate that it looks correct.
- Check all fields – In charts and graphs, you'll want to check all fields to make sure they exist in the production candidate. Any fields which don't exist will generally turn red, or have a red X next to them in the chart properties (if it's a chart).
- Check all expressions (expressions everywhere, not just in the expression tab)
- Check that they are valid
- Check that they are efficient (additional data model changes may be necessary if the expressions could be made to be more efficient)
- Drill/cycle groups – There is no way to copy/paste these, they will need to be rebuilt. Also look at the sort order in the groups, the sort order could be important to the developer
- Possibly any other supporting architecture such as Island Tables, new fields, etc.

Merging in new Script:

- Identify the new script and validate the script is going to work with the coding standards which have been adopted.
- Make sure it executes correctly.
- Merge the new script into the existing script. It may be possible to merge the new script with some existing script to improve efficiency instead of continually adding new script to the document

Modifications to Publisher:

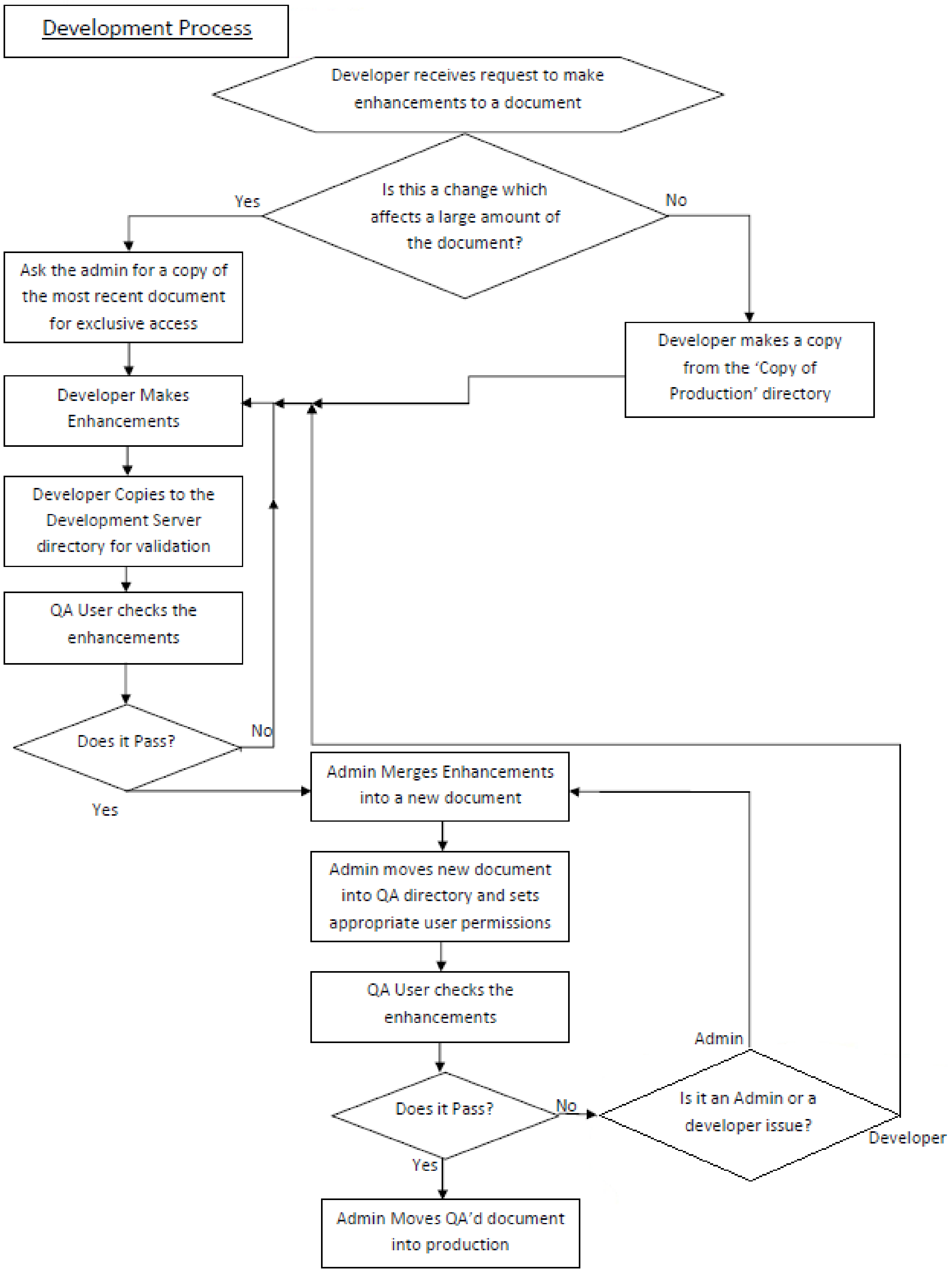
Users won't be giving you actual publisher items to move into production. Instead, publisher changes will come with the requirements of other enhancements.

For example, if a new request is made to reload data on a new schedule, that would be a publisher change which would need to be executed. If a new script module was added which utilized additional documents, publisher would have to be adjusted to accommodate the additional documents.

Publisher changes will be up to your discretion as to what needs to be accomplished and what timing they need to be accomplished. There are a couple of guidelines which you'll want to follow though:

- Don't 'Distribute' large documents to the QlikView Server Resource during times when users are utilizing the system. Doing this will slow the QlikView Servers dramatically Small documents (which take 30 seconds or less) are fine.
- When validating new data model changes (script changes and such), it's best to develop the publisher jobs and let them execute from publisher to validate that the data movement is occurring as it should. Without doing this, you won't know if publisher has the proper permissions and such defined to execute the jobs in their actual production environment

Development Process


Development Process
Developer receives request to make enhancements to a document
Is this a change which affects a large amount of the document?
Yes
No
Ask the admin for a copy of the most recent document for exclusive access
Developer makes a copy from the 'Copy of Production' directory
Developer Makes Enhancements
Developer Copies to the Development Server directory for validation
QA User checks the enhancements
Does it Pass?
No
Yes
Admin Merges Enhancements into a new document
Admin moves new document into QA directory and sets appropriate user permissions
QA User checks the enhancements
Does it Pass?
No
Yes
Is it an Admin or a developer issue?
Admin
Developer
Admin Moves QA'd document into production

Naming Standards

Publisher

- It's generally considered bad practice to put the time of an action (I.E. 'Daily', 'Nightly') in the task's name since that task can be scheduled by many different jobs which all can be scheduled differently
- It's generally considered good practice to put the time of an action in a Job's name.
- If you have a mixed QA/Dev environment, you will want to prefix anything (Except for an individual .qvw) with either a Dev_ or QA_ Production would either have a Prod_ prefix, or no prefix (your choice as long as everyone follows the same standards)
- Always use log files for any application promoted to production. Without log files, publisher can't capture the details of the success/failure of an application's execution
- Use UNC names, or automated tasks might not be able to reference the paths

Abbr.	Abbr. Type	Meaning
All	Environment	Applicable to all environments
DEV	Environment	Development Environment
PRD	Environment	Production Environment
TST	Environment	Test Environment
APR	Publisher Item	Publisher Access Point Resource
JOB	Publisher Item	Publisher Job
SDF	Publisher Item	Publisher Source Document Folder Resource
TSK	Publisher Item	Publisher task
DSR	Publisher Item	Directory Service Resource

Scripting and Layout

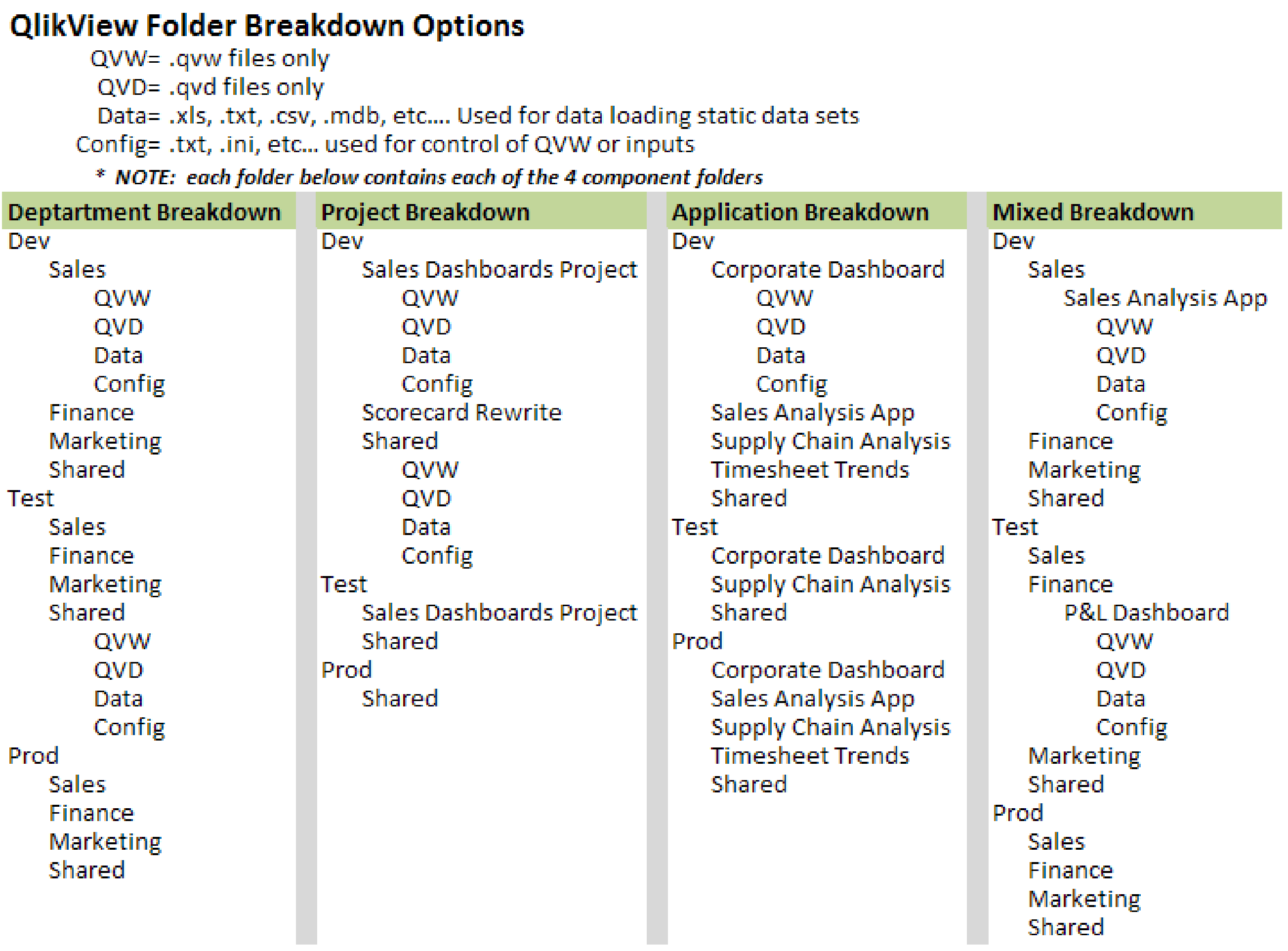
Come up with a naming standard:

- Use business names for data fields e.g., Customer Nbr instead of CustNo
- All abbreviations are a standard type. Get a list of abbreviations and use it (I.E. always use Desc for Description, as specified in the abbreviations list)
- Utilize a Prefix
  - Variables = starts with a "v" e.g., vCurrentYear
  - Key fields = starts with a "%" e.g., %CustomerKey
  - Flag fields = starts with a "_" e.g., _YTDFlag
  - / Cycle Group = starts with a "<" e.g., <ProductCycle
  - Drilldown Group = starts with a ">" e.g., >GeographyDrilldown
  - Key Field Separator = separated by "_" e.g., Company&'_'&Nbr as Key
  - Temp Fields/Tables = ends with "_tmp" e.g., Daily_Trans_tmp

Folder Structures

Overview

Folders to hold your production QlikView files are also important. The structure of your folders should allow developers to easily locate and read the files for which they have access. Below are some strategies for the location of QlikView files.



QlikView Folder Breakdown Options

QVW= .qvw files only
QVD= .qvd files only
Data= .xls, .txt, .csv, .mdb, etc.... Used for data loading static data sets
Config= .txt, .ini, etc... used for control of QVW or inputs

* *NOTE: each folder below contains each of the 4 component folders*

Deptartment Breakdown

```
Dev
    Sales
        QVW
        QVD
        Data
        Config
    Finance
    Marketing
    Shared
Test
    Sales
    Finance
    Marketing
    Shared
        QVW
        QVD
        Data
        Config
Prod
    Sales
    Finance
    Marketing
    Shared
```

Project Breakdown

```
Dev
    Sales Dashboards Project
        QVW
        QVD
        Data
        Config
    Scorecard Rewrite
    Shared
        QVW
        QVD
        Data
        Config
Test
    Sales Dashboards Project
    Shared
Prod
    Shared
```

Application Breakdown

```
Dev
    Corporate Dashboard
        QVW
        QVD
        Data
        Config
    Sales Analysis App
    Supply Chain Analysis
    Timesheet Trends
    Shared
Test
    Corporate Dashboard
    Supply Chain Analysis
    Shared
Prod
    Corporate Dashboard
    Sales Analysis App
    Supply Chain Analysis
    Timesheet Trends
    Shared
```

Mixed Breakdown

```
Dev
    Sales
        Sales Analysis App
            QVW
            QVD
            Data
            Config
    Finance
    Marketing
    Shared
Test
    Sales
    Finance
        P&L Dashboard
            QVW
            QVD
            Data
            Config
    Marketing
    Shared
Prod
    Sales
    Finance
    Marketing
    Shared
```

Department Breakdown

This strategy is to separate the files into folders for specific subject areas under the environment folders, including a Common folder to hold all files that are candidates to be used across subject areas. This is a good strategy for established deployments that don't have many new development projects running at any given time. It can be less desirable if many new development projects are running, since there are not project-specific folders in this strategy. That means new development files that may or may not be in Production will be present in the Dev/Test/QA folders mixed in with established Production files.

Project Breakdown

This strategy is common where several development projects are running concurrently. It houses the files specific to a project within a folder named for the project. This makes it easy for developers and administrators to quickly identify new files related to a project and isolate them for testing and code promotion. Once files make it to Production they are all housed under the “Shared” folder since there are no development projects in Production.

Application Breakdown

This strategy is common where large QlikView applications exist which do not greatly overlap each other in file use. Separating the files by application makes it easier to identify files needed for enhancements or additions to an application. Note that there is still a Shared folder that will house all files which are common to more than one of the applications. Over time, this approach may be less desirable if more and more common files are used.

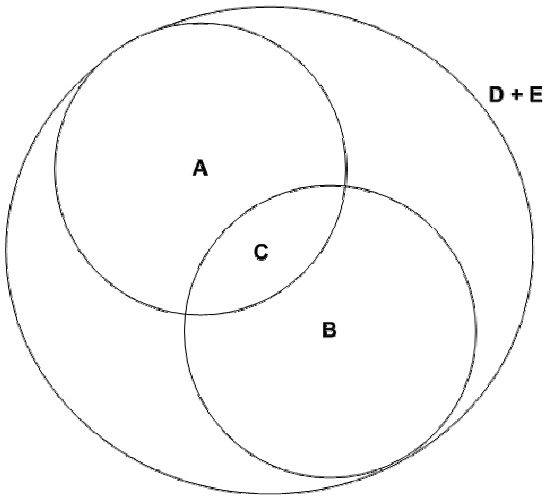
Mixed Breakdown

This strategy mixes the Department Breakdown with either the Project Breakdown or the Application Breakdown. The picture above shows the Department/Application combination for demonstration purposes. This approach is common for larger QlikView deployments where several departments are developing applications that may not overlap each other greatly.

It’s important to note that any approach above or combination of approaches will work, but consistency is the key to folder strategy. You can, of course, switch strategies as your QlikView deployment grows, but remain consistent and vigilant *within* the strategy you are using in order to maintain control and governance over code management and deployment.

Folder Security

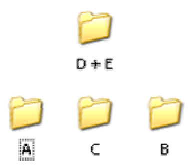
This is an example how to secure your QlikView Source files. There are many ways to do this, but as a best practice QlikTech recommends that you incorporate a security strategy that matches your development strategy. This will allow you to isolate development files from people that shouldn’t access them, while still allowing them access to shared files that allow for re-use and consistency.



Use different groups to match your source data file structure.

A) Department1/Development group 1
B) Department2/Development group 2
C) Shared company data (Shared_Folders)
D) QlikView Administrator has access to all groups and distributed documents
E) Service Account for QlikView service, are member of group D

This service must have read access to databases, file system and Active Directory



Testing & Certification

Tests to Perform

1. Reload Testing
   a) Reload locally
   b) Reload from QVS (manually)
   c) Reload automated from an isolated Publisher task

2. Publisher Testing
   a) Test tasks individually for reload, distribution, statuses
   b) Build dependencies (if needed) and test individually

3. User Testing (UI)
   a) Individual user testing
   b) Private bookmarks
   c) Exports (all formats)
   d) Concurrent user testing (unstructured)
   e) Concurrent user testing (simultaneous function testing)
   f) Concurrency testing (multiple apps)

4. User Testing (Collaboration)
   a) Testing new collaboration objects
   b) Testing shared collaboration objects
   c) Server Bookmarks

5. Performance Testing
   Simultaneous tests of 10+ users on QVW

6. Regression Testing
   If this is an enhancement to a QVW, perform a regression test using the test cases from initial deployment of the QVW

7. Cycle Testing
   Run 4-6 daily full cycle reloads to refresh the QVW from source to end user interface

Environments

a. Single environment – local testing

   This approach is used when a client only has one server (PROD) and as such, needs to locally test code on developer machines until the code is ready for production.   This can be done with local unit testing on the developers' machines, then a limited availability version of the QVW can be placed in PROD to acceptance test. Once that version of the QVW has passed testing it can be made available as the PROD version.

b. 2-Server environment – promotion

This approach involves performing all testing on the Dev/Test server and then, once passed, the code is promoted to PROD. Unit testing can still be performed locally by developers, but an official acceptance test should be performed on the Dev/Test server once the developer has completed unit testing.

c. 3-Server environment – promotion

This is basically the same as the prior approach, except that DEV, TEST and PROD are used. DEV is first used for unit testing. Then TEST is used for acceptance testing, and once it passes tests the QVW is promoted to PROD.

End User Testing

1. Individual – Unstructured

   These tests would be performed by end users at their convenience without structured plans or directions as to what and how they would test. Individual end users are simply granted access to the v9 Testing environment and told to test anything they would like.

2. Individual – Structured

   These tests would involve directions on what and how to test, but they would not dictate when to test or coordinate the tests among user groups. They would likely have directions indicating which QVWs to test and what functions to test in a particular order. Results would be gathered to compare among users that completed the tests.

3. Group – Unstructured

   A group of end users are all granted access to the v9 Testing environment and simultaneously test for a period of time. This might be conducted over a series of n-hour time blocks or days, or could be one continuous test window. The end users are not given structured directions on what and how to test, but are simply told to test everything.

4. Group – Structured

   A group of end users are all granted access to the v9 Testing environment and simultaneously test for a period of time. This might be conducted over a series of n-hour time blocks or days, or could be one continuous test window. They are given specific directions on what and how to test. They are also coordinated (preferably in the same room or over a conference call line) so that they can perform concurrency tests to simultaneously initiate functions or access documents. The results would be gathered to compare to similar tests conducted in the 8.5 environment as a control.

Test Adoption

It can be difficult to gain access to end users' time and effort for testing. Some of these approaches can help to promote testing and ensure good results.

1. Coordinate test windows for users in advance. Let end users know that you will have "test windows" where the environment will be monitored and you will be available to help with any questions or needs. This helps ensure that their time will not be wasted and they will not be waiting for help or guidance if needed.

2. Have a prize for the end user or group that discovers the largest amount of bugs with the new version's documents. This is a great motivator for end users and will ensure that they stress test the QVW's in a thorough manner.

3. Co-locate users for tests. Find a large conference room and have end users come to that room for the testing. Training labs work well if users cannot bring laptops.

4. Have users coordinated over the phone while testing. Have an open conference line available during any test windows so that end users can converse with each other, can ask questions, and can take direction for simultaneous testing events.

5. Give users a check list of documents to test or functions to test. Sometimes users cannot remember all of the features available to them from within a QVW. Give them a checklist that includes bookmarks, reports, exports, conditionally visible objects, collaboration, etc…

6. Coordinate test windows for users in advance. Let end users know that you will have "test windows" where the environment will be monitored and you will be available to help with any questions or needs. This helps ensure that their time will not be wasted and they will not be waiting for help or guidance if needed.

7. Have a prize for the end user or group that discovers the largest amount of bugs with the new version's documents. This is a great motivator for end users and will ensure that they stress test the QVW's in a thorough manner.

8. Co-locate users for tests. Find a large conference room and have end users come to that room for the testing. Training labs work well if users cannot bring laptops.

9. Have users coordinated over the phone while testing. Have an open conference line available during any test windows so that end users can converse with each other, can ask questions, and can take direction for simultaneous testing events.

10. Give users a check list of documents to test or functions to test. Sometimes users cannot remember all of the features available to them from within a QVW. Give them a checklist that includes bookmarks, reports, exports, conditionally visible objects, collaboration, etc…

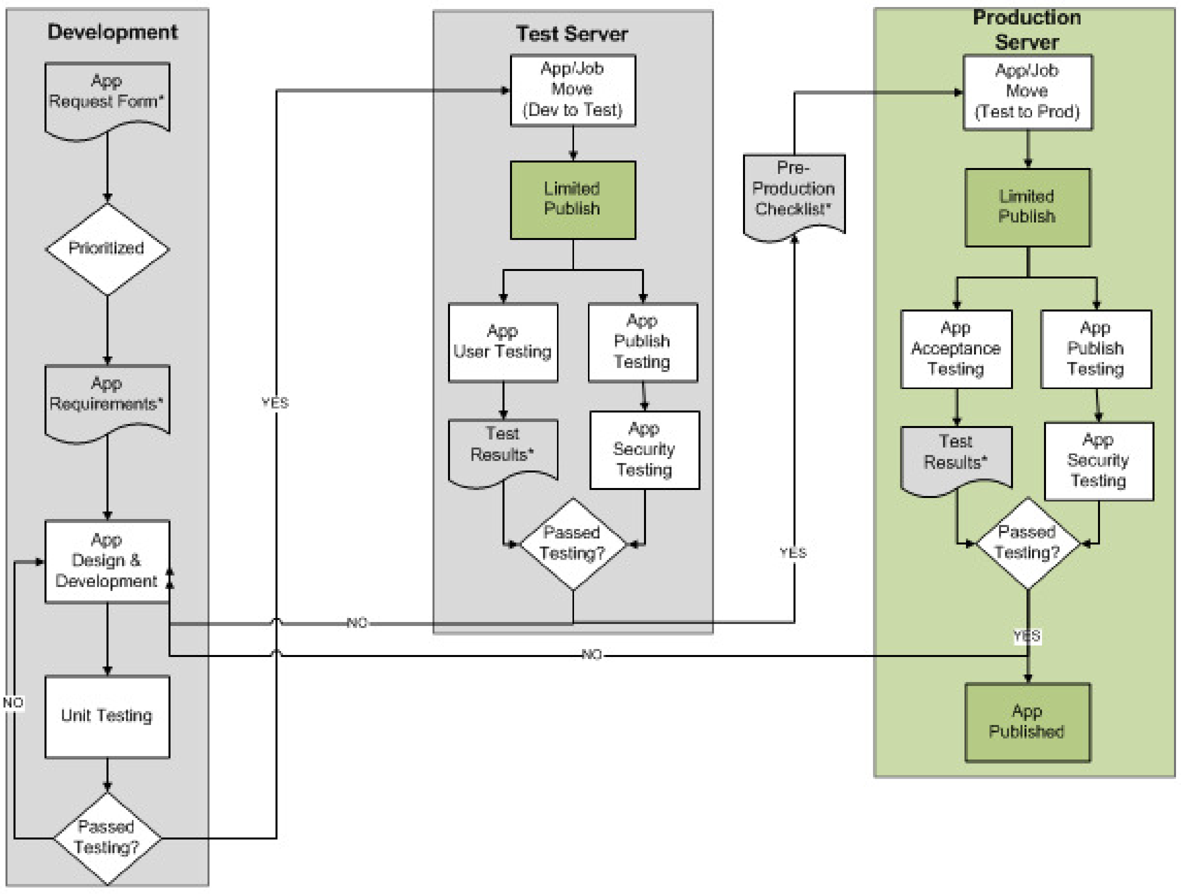
There are many ways to test code in QlikView. The two most important success factors in effective testing have been:

1. Planning.
   Have a plan for the testing that allows you to break up the QlikView responsibilities and test them individually (development, reloading, publishing, security, usage, monitoring, troubleshooting, etc…). Then, bring those responsibilities together and cycle test them where possible. Document the test plan and repeat the steps that can be repeated.

2. Time Commitment. Secure serious time commitment from developers, admins and end users of QlikView. A common benchmark is that the testing should involve at least 10% of the development time for a QVW. This means that if a QVW took 160 hours to develop it should get 16 hours of testing time prior to its promotion to Production. This could be 16 users for an hour each or any combination of users and hours that total to 16 (or 10% of development time). Make those hours count by giving some structured and some unstructured time to the end users, as well as making sure they have some motivation to uncover issues.

Workflows

QlikTech recommends the creation of a Development Workflow document that diagrams the steps involved in code promotion throughout QlikView environments. The diagram below shows an example of a development workflow:

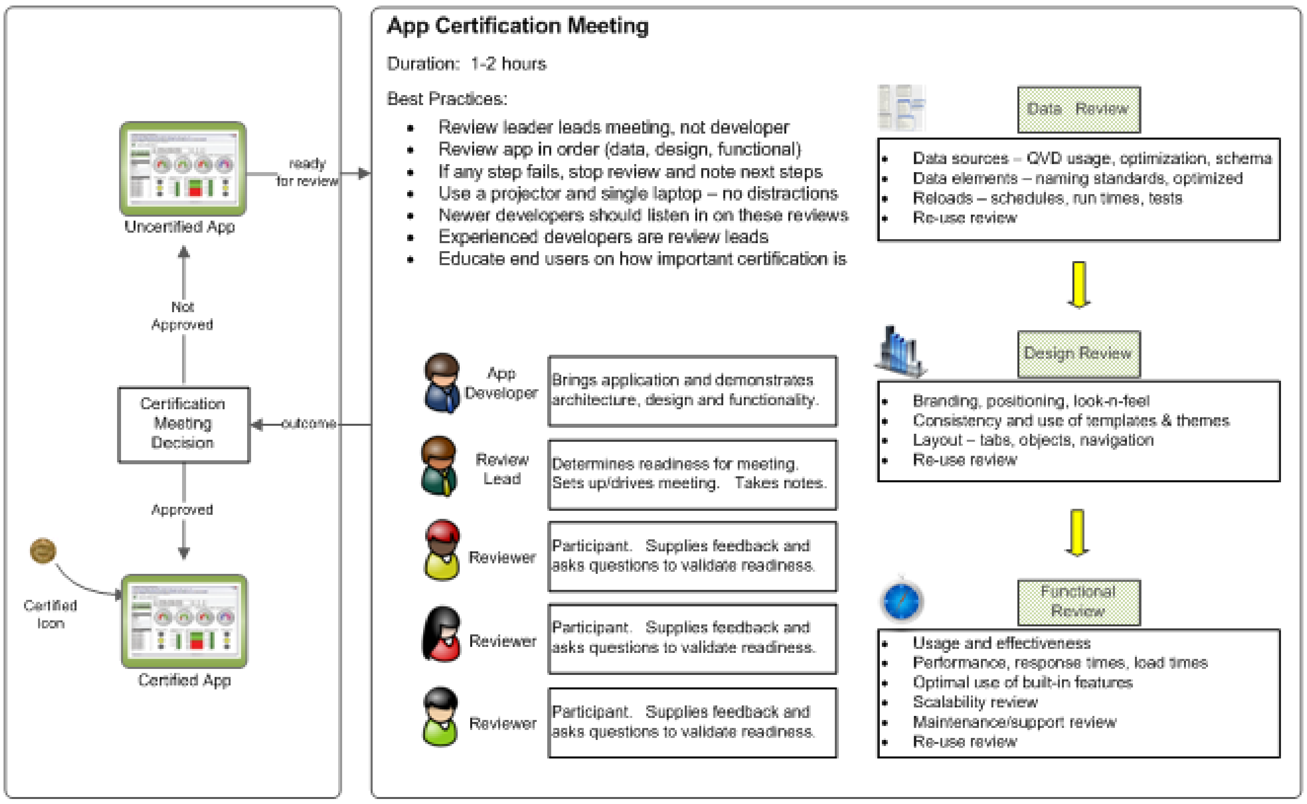


Certification

Many companies benefit from the speed and flexibility of QlikView development, but also wish to retain a more rigorous process for some QlikView development of very important or high profile applications. In order to distinguish between "go fast" applications and "high rigor" applications many QlikView clients use a Certification process.

This process serves as a tollgate to getting a QlikView application "certified". Certification means an application has gone through this process and been approved. A "Certified" icon is then placed in the title section of the application so that users and support teams know which applications are certified and which ones are not. This allows teams to place emphasis on this process by not providing the same level of support for non-certified applications.

The diagram below shows what a sample Certification meeting might look like.



App Certification Meeting

Duration: 1-2 hours

Best Practices:

- Review leader leads meeting, not developer
- Review app in order (data, design, functional)
- If any step fails, stop review and note next steps
- Use a projector and single laptop – no distractions
- Newer developers should listen in on these reviews
- Experienced developers are review leads
- Educate end users on how important certification is

App Developer	Brings application and demonstrates architecture, design and functionality.
Review Lead	Determines readiness for meeting. Sets up/drives meeting. Takes notes.
Reviewer	Participant. Supplies feedback and asks questions to validate readiness.
Reviewer	Participant. Supplies feedback and asks questions to validate readiness.
Reviewer	Participant. Supplies feedback and asks questions to validate readiness.

Data Review

- Data sources – QVD usage, optimization, schema
- Data elements – naming standards, optimized
- Reloads – schedules, run times, tests
- Re-use review

Design Review

- Branding, positioning, look-n-feel
- Consistency and use of templates & themes
- Layout – tabs, objects, navigation
- Re-use review

Functional Review

- Usage and effectiveness
- Performance, response times, load times
- Optimal use of built-in features
- Scalability review
- Maintenance/support review
- Re-use review

Troubleshooting & Support

Support Types

Supporting QlikView applications and environments can be done in several ways. As a best practice, QlikTech recommends that support levels and services be identified for the following areas:

- QlikView Applications (QVWs)
- QlikView Interface (end user support)
- QlikView Server/Publisher
- QlikView Data Architecture (QVDs and QlikView data, in general)

Many QlikView clients utilize certified QVWs for application support of high importance apps. This can help especially when business teams are creating their own QVWs and your support team is only responsible for supporting the certified applications that it had a chance to code/interface/data review. See the section called Testing & Certification in this document for more details on the certification process.

QlikView Development Teams

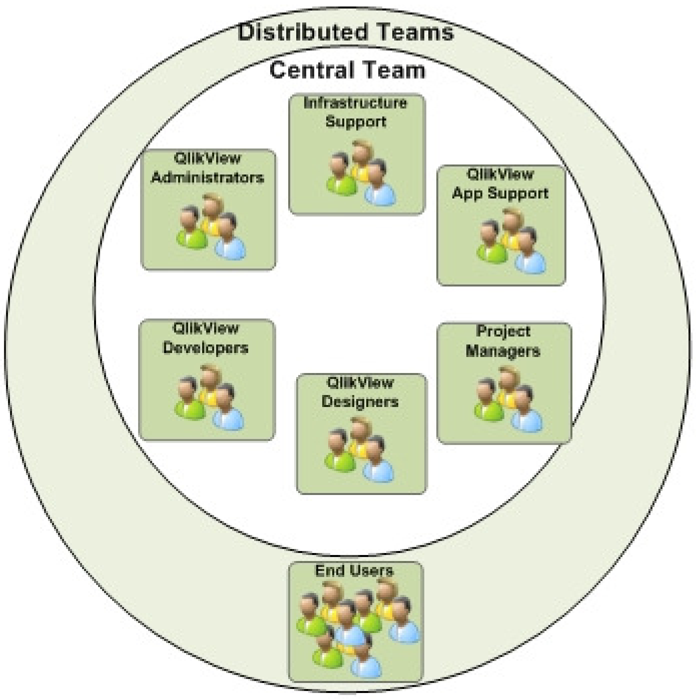
QlikView is an extremely flexible and easily adapted BI tool. As such, development teams can organize around several models for support, administration, development, training and management. These scenarios help guide discussions about possible configurations for QlikView roles in a development environment.

It is recommended that the client consult its own IT standards for development, as they may drive this decision, or at least narrow the allowed choices. QlikTech does not expressly promote one of these scenarios over the others, but asks that clients determine for themselves which of these configurations might work best, given the nature of the QlikView development and the skills sets that exist.

On a continuum from Fully Centralized to Fully Decentralized, the following are 5 options for QlikView team structures:

1) Fully Centralized

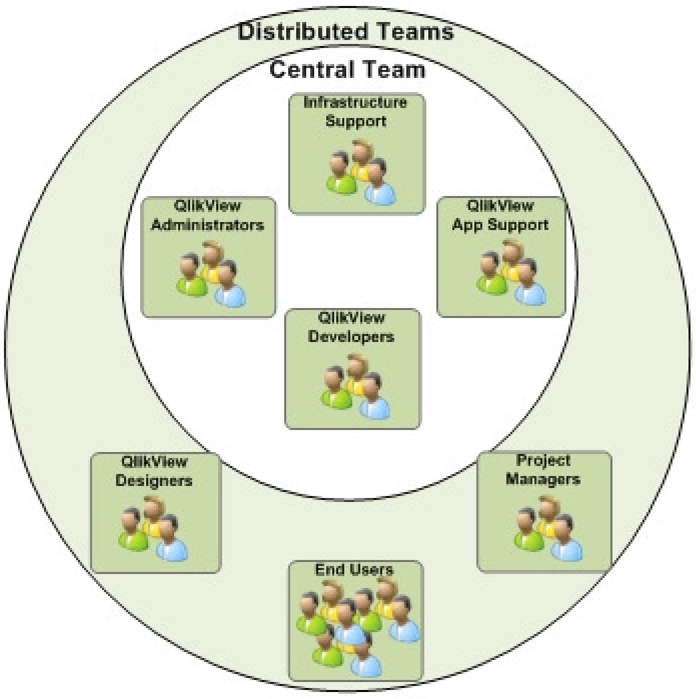
Central Team	Distributed Team(s)
Infrastructure Support	End Users
QlikView Administrators	
QlikView Application Support	
QlikView Developers	
QlikView Designers	
Project Managers	



In this option, departments don't need to supply developers, support personnel or administrators to use QlikView applications. They request new applications and then consume them along with central QlikView services. Strengths in this approach are control, skill set sharing, consistency and governance, since all services are contained to one team.

2) Co-Development (v1)

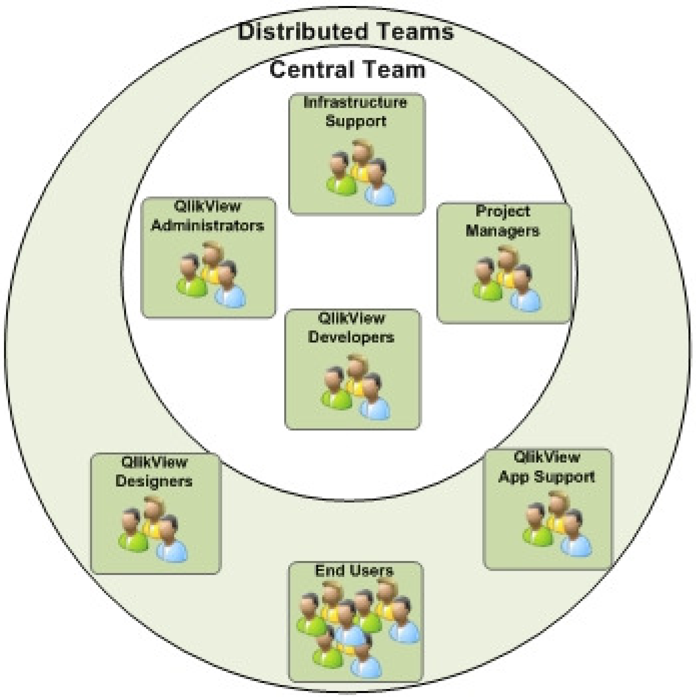
Central Team	**Distributed Team(s)**
Infrastructure Support	End Users
QlikView Administrators	QlikView Designers
QlikView Application Support	Project Managers
QlikView Developers	



In this option, enterprise development is retained as a central function, allowing for the scripting and data modeling to be handled by expert QlikView developers and data professionals. Departments are responsible for all training, project mgmt, application design, testing and support.
The strengths of this approach are that the back-office BI work is still centralized, but the design and project management are done in the business teams, so that they can move at a faster pace, partially independent from IT resources.

3) Co-Development (v2)

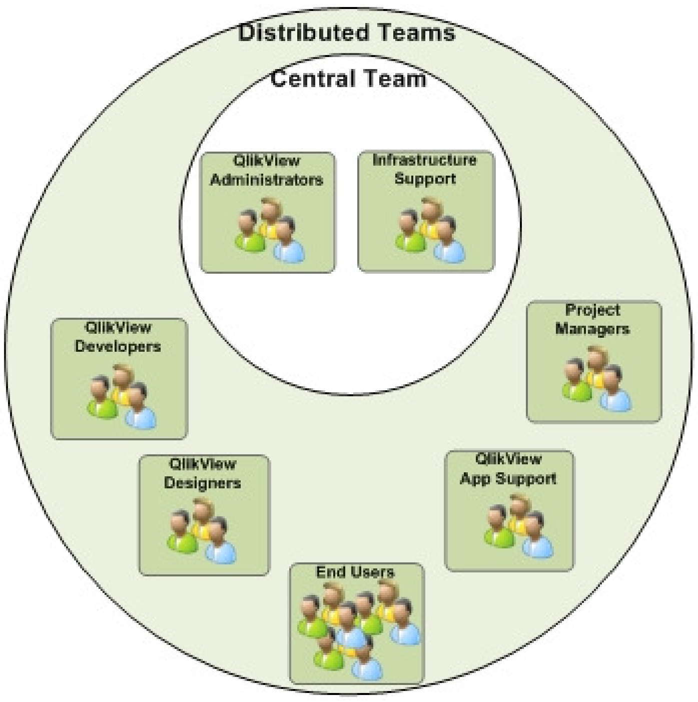
Central Team	**Distributed Team(s)**
Infrastructure Support	End Users
QlikView Administrators	QlikView Application Support
QlikView Developers	QlikView Designers
	Project Managers



In this option, support has been moved to departments, but Project Mgmt is retained in the central team to better allow for QlikView expertise and control of designs. Departments are responsible for all training, application design, and support. Strengths of this approach are similar to the v1 Co-Development model, with the exception of the QlikView Application Support now also being distributed to the business teams. This frees up more IT resources and places some responsibilities on the distributed teams to provide support and training to their end users.

4) Mostly Decentralized

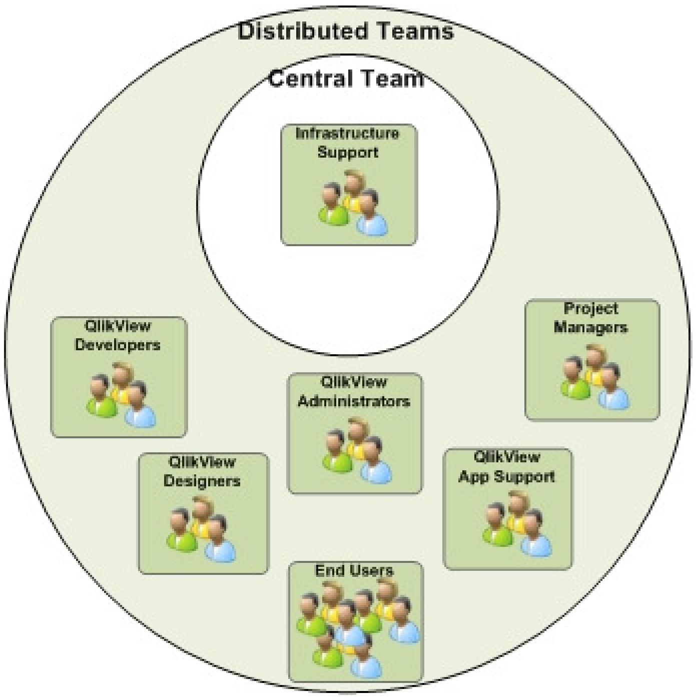
Central Team	**Distributed Team(s)**
Infrastructure Support	End Users
QlikView Administrators	QlikView Application Support
	QlikView Developers
	QlikView Designers
	Project Managers



In this option, departments are responsible for all training, project mgmt, application design, scripting, development, testing and support. The Central team is still providing infrastructure support and QlikView Administration. This allows for a small IT team (usually less than 2 people) to administer rights, server settings, and batch processing (reloads). Strengths of this approach are that the Centralized (IT) team is very small.

5) Fully Decentralized

Central Team	**Distributed Team(s)**
Infrastructure Support	End Users
	QlikView Administrators
	QlikView Application Support
	QlikView Developers
	QlikView Designers
	Project Managers

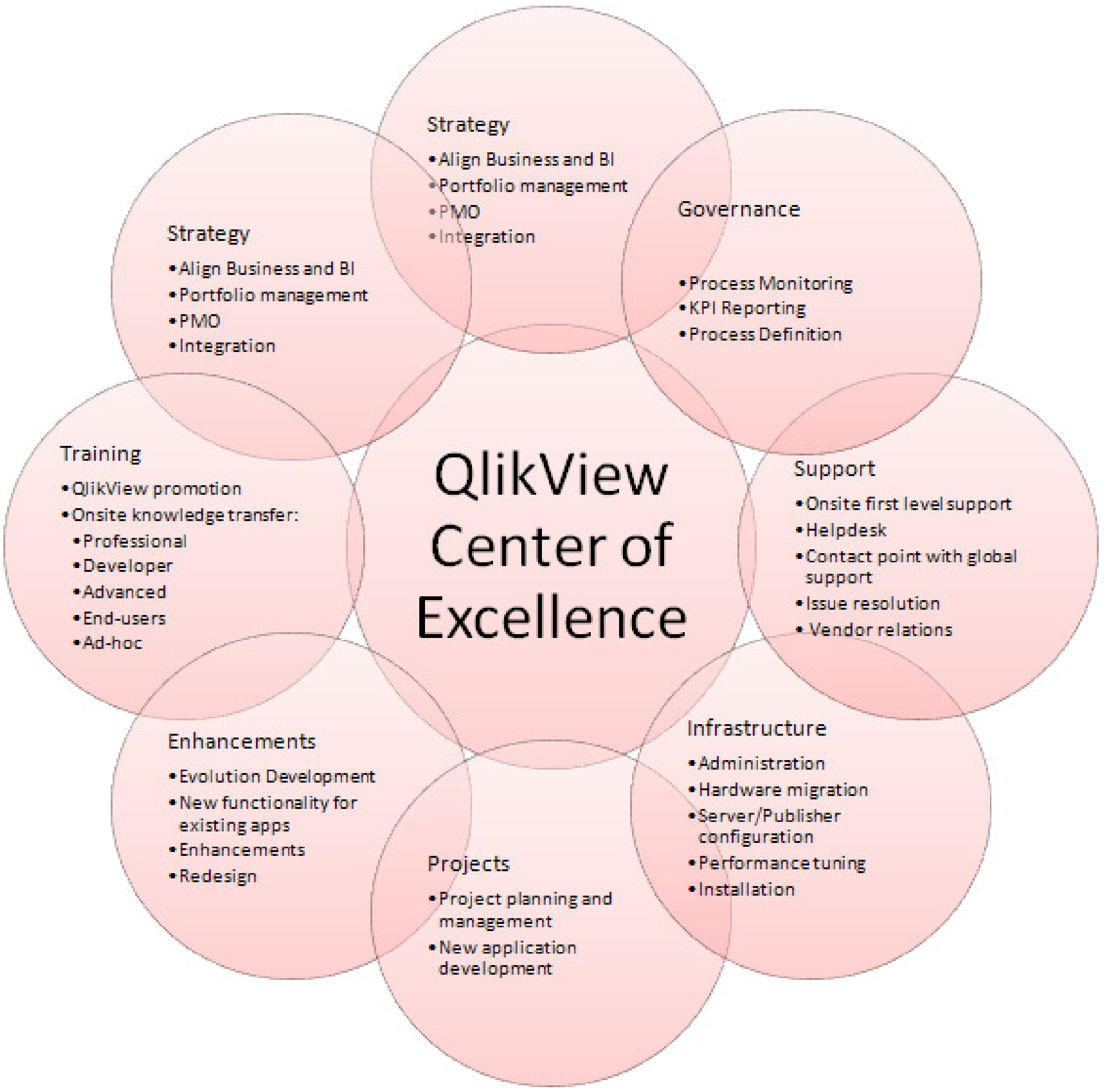


In this option, infrastructure is still maintained centrally, but all other aspects of QlikView development, testing, support, training a usage are distributed to departments. This requires distributed teams to be trained on all aspects of QlikView. The strength of this option is that there is no software-specific resource needed in a central team. However, the challenge of this approach is that those software-specific resources will need to be present in several distributed teams, possibly overlapping.

Choosing a development team structure is an important step in an enterprise deployment of QlikView. While the Co-Development options (#2 & #3) are the most popular today, the right option for each client is the one that matches their needs and strengths.

QlikView Centers of Excellence

QlikTech recommends that clients implementing enterprise deployments of QlikView strongly consider the forming of a QlikView Center of Excellence (or Competency Center). This can be as formal or informal as needed, but the services and synergies shared in a center of excellence provide significant savings and coverage for enterprise deployments.



QlikView Center of Excellence (CoE)

Integrated by a mixed team from IT and business users, the QVCoE centralizes organizational knowledge and best practices to standardize, implement, and manage business intelligence solutions in the organization.

Objectives:

1. Develop and share QlikView best practices

2. Ensure functionality and operation of QlikView applications
3. Design and implement a common BI architecture
4. Promote the use of BI throughout the organization

Organization Structure

- Centralized or decentralized, depending on company culture, skill sets of teams involved and policies
- Can be dedicated or virtual
- Should “think big” but start small
- Allow for and plan for growth of QlikView and company

Benefits

- Increase credibility and confidence in corporate information
- Rollout the use of BI for the whole company
- Accelerate decision making process
- Optimize resources and reduce costs
- Business process innovation through BI insights
- Continuously evolve QlikView BI applications to support changing business requirements

Steps:

1. Executive Sponsor
2. COE mandate and objectives
3. Funding
4. Organizational structure/reporting lines
5. Functional areas
6. Required roles
7. COE KPIs

QlikTech can provide materials and templates to help you define the centralized services, KPIs, structures and best practices for creating a QlikView Center of Excellence. This should be sized and scoped to be commensurate with your QlikView deployment. Meaning, it can start small and then grow as your QlikView deployment grows.

Training

An effective training strategy is essential in the creation, on-going upgrade or deployment of any QlikView application. Whether you are about to embark on a small sales dashboard for a single manager or a worldwide deployment of thousands of users, the training strategy will greatly affect the success of the project and how people utilize the system to efficiently perform their jobs.

The four key areas' and their QlikView training offerings are listed below:

Developer

This role is responsible for developing and documenting the QlikView document according to the specified requirements.

- ✓ Build Your First QlikView
- ✓ QlikView Developer 1
- ✓ QlikView Developer 2
- ✓ QlikView Developer 3
- ✓ Set Analysis
- ✓ SAP Connector

UI Designer

This role focuses on usability and develops the User Interface of the QlikView document according to the requirements.

- ✓ Build Your First QlikView
- ✓ QlikView Designer 1
- ✓ QlikView Designer 2

System Admin

This role is responsible for hardware, operating systems and job schedules.

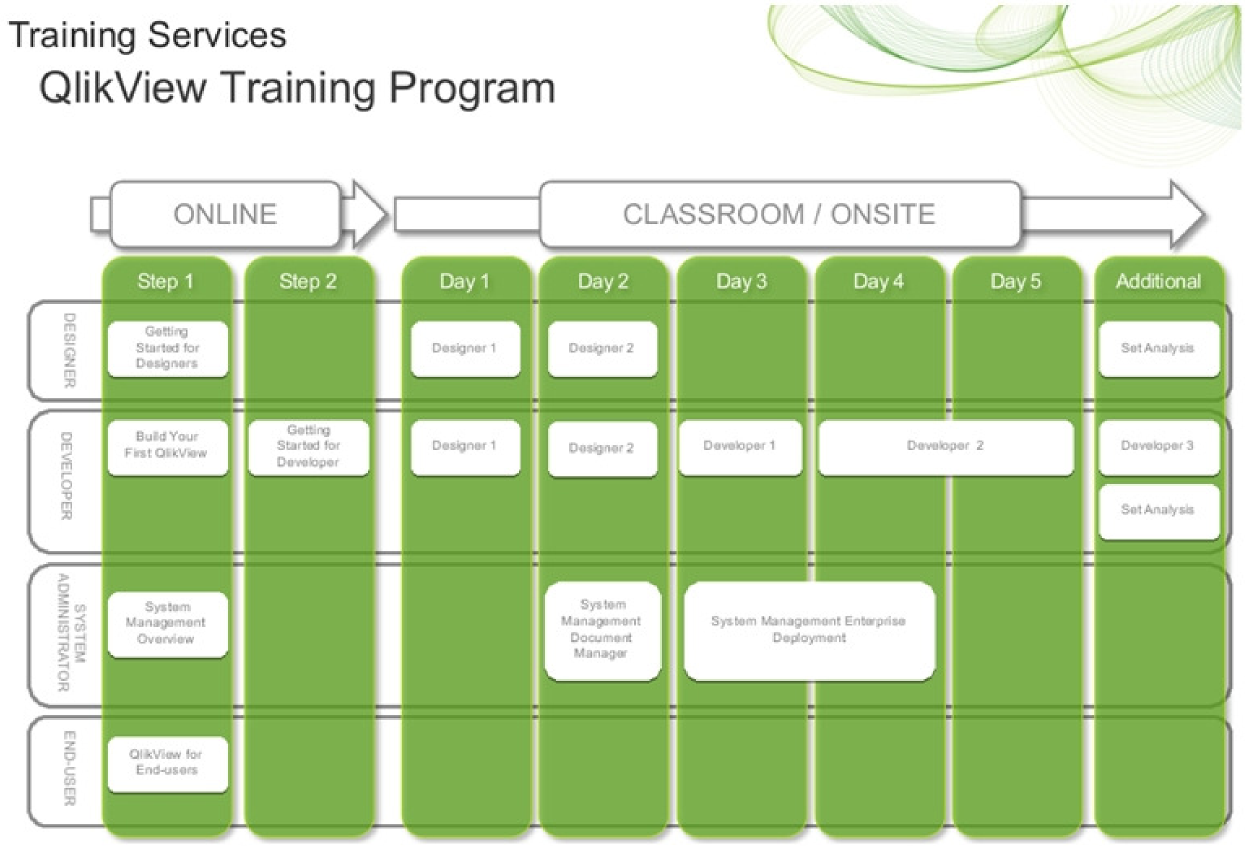
- ✓ System Management – Overview
- ✓ Document Manager
- ✓ Enterprise Deployment

End User

This role is a consumer of QlikView applications.

- ✓ QlikView Version for End Users

The following table shows the potential training for several roles, including those listed above:



Training Services

QlikView Training Program

	ONLINE		CLASSROOM / ONSITE					
	Step 1	Step 2	Day 1	Day 2	Day 3	Day 4	Day 5	Additional
DESIGNER	Getting Started for Designers		Designer 1	Designer 2				Set Analysis
DEVELOPER	Build Your First QlikView	Getting Started for Developer	Designer 1	Designer 2	Developer 1	Developer 2		Developer 3 Set Analysis
SYSTEM ADMINISTRATOR	System Management Overview			System Management Document Manager	System Management Enterprise Deployment			
END-USER	QlikView for End-users							

QlikTech strongly recommends that clients attend Developer, Designer and Systems Admin training for QlikView in order to maximize the return on investment in QlikView hardware and software. The depth of courses to be taken will depend on the needs of your deployment. Please consult your Regional Services Director at QlikTech or your Account Executive to discuss options.

Summary

QlikView is a very fast and flexible BI solution. Although there are few moving parts (QVDs and QVWs) to manage for development, there are endless combinations or resources, processes, environments, uses, delivery platforms and designs that can be applied. With speed comes the need for some governance and consistency, and QlikView is no exception to that.

These best practices are not fully comprehensive. A good strategy for staying current with best practices is to augment the use of documentation such as this with other forms of input. Some of these are:

- QlikCommunity – QlikView's online community with 10,000's of users
- QlikTalks – local and regional QlikView gatherings where best practices, new features, showcase solutions and vendor experts are highlighted.
- QlikTech Expert Services – an array of services are designed to help a QlikView team come up to speed very rapidly with best practices in development, architecture and administration. Contact your Account Executive or Regional Services Director for information on the services available from QlikTech and its certified partners around the world.
- Developer groups – groups local to a client or organized by several clients are forming in many cities. Check to see if there is a local user group in your city through QlikCommunity.

QlikTech strongly urges clients to take time to learn best practices and techniques that optimize deployments and speed time to delivery for QlikView. Consider the implementation of these best practices and others into your deployment. They are a great way to accelerate the successes of QlikView even further!