QlikView

QlikView Best Practices – Metadata Management

Best Practices in Managing QlikView Metadata

Version: 1.0
Date: 2009-03-24
Author(s) BPN

”A best practice is a technique or methodology that, through experience and research, has proven to reliably lead to a desired result.”

Table of Contents

Overview

Three Kinds of Metadata

There are three types of metadata to consider when managing applications:

1) ***Descriptive Metadata*** describes an application's origin and status. This can include information about the author, purpose, uses and history of an application

2) ***Administrative Metadata*** provides information on the "who, what, where and when" of an application. This includes information about who can access the app, who has accessed it, how was it used, when was it used.

3) ***Structural Metadata*** describes the components of an application. What is it made of (QVDs, tables, columns, expressions, charts, etc…).

This paper describes some techniques and best practices for managing metadata across these three categories. There is frequent overlap between Metadata Management best practices and several other categories of best practices, particularly in the areas of Code Management, Monitoring and Optimizations. Where possible, this document will reference those best practices instead of repeating the content.

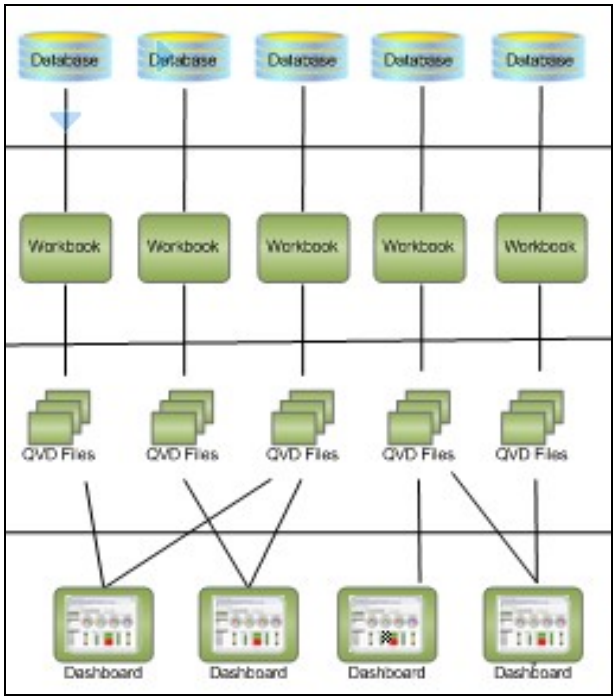
A special note on Structural Metadata

As you will see in the section on Structural Metadata management, the use and management of a QVD-based architecture (twp-tiered architecture) is key to most advantages to be gained in this area, as they allow for a data layer that can be more easily represented than direct queries in QlikView applications. There are methods for recording and displaying structural metadata from direct query applications, but they are much harder to employ in an enterprise implementation and are not the focus of this document.

2-Tiered QlikView Development – Explained

This refers to a development methodology where QVDs are generated distinct from the applications that consume them. For example, QlikView "workbook" applications are created to generate the QVD files that become the sources for the dashboard and analytics applications (QVWs) that are loaded later. This 2-tiered approach is very popular, especially in larger QlikView environments or those where re-use and optimization of data extraction is necessary.

(diagram1) two-tiered architecture



Workbook applications extract data from databases and write out QVD files.

These QVD files then become the main inputs to the dashboards and analytics applications that are built later.

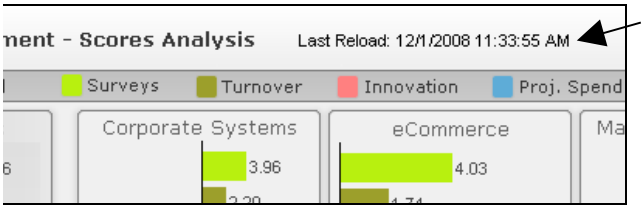
Descriptive Metadata

Descriptive Metadata includes data about the QlikView document and its origins. This includes data like application author, purpose, topics, search terms, history and intended usage. The collection and management of these values provides rich context to the makeup of a QlikView application. This is useful when comparing applications, shopping for examples or code to re-use, and evaluating trends and coding practices.

QVW Metadata

Several key pieces of application metadata are available from within the application for developers and designers. They include:

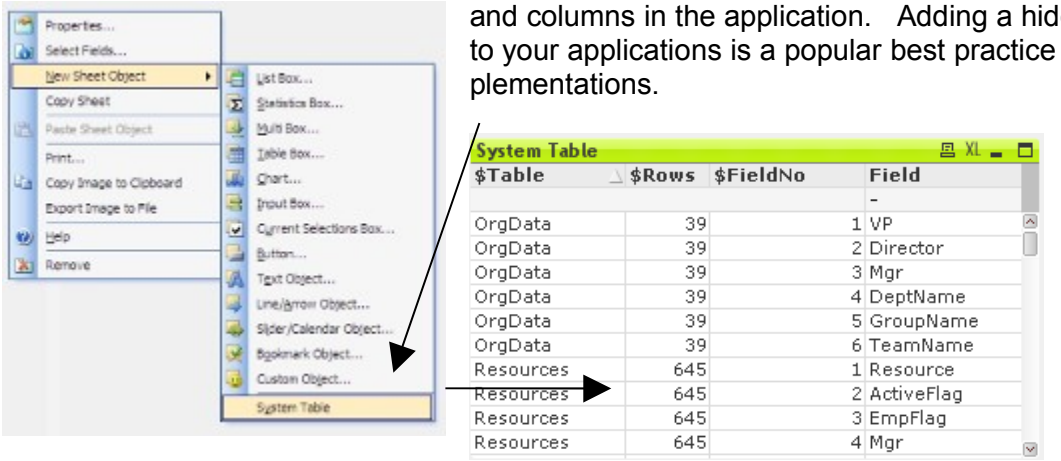
Last Reload Date/time:



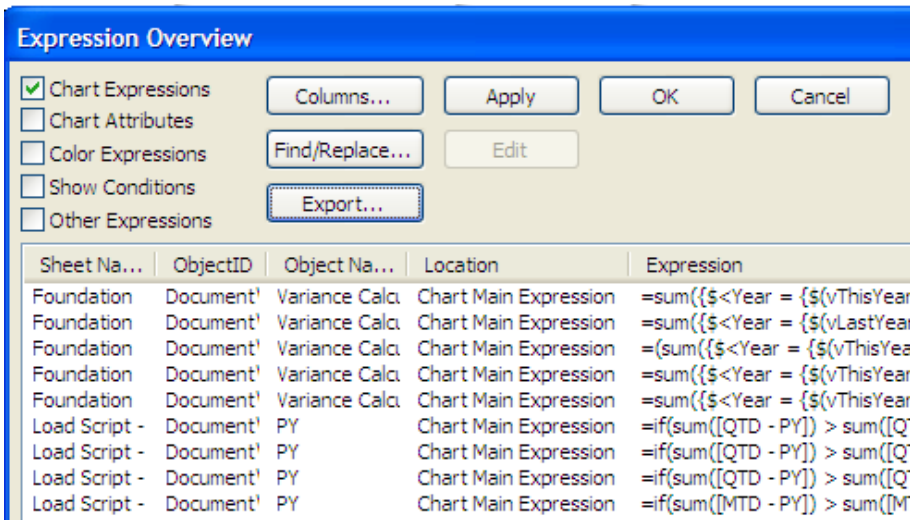
Place this code in any text box:
='Last Reload: ' & ReloadTime()

Embedded System Tables:

You can place add a system table to your applications, which contains metadata about the tables and columns in the application. Adding a hidden support tab to your applications is a popular best practice for QlikView implementations.

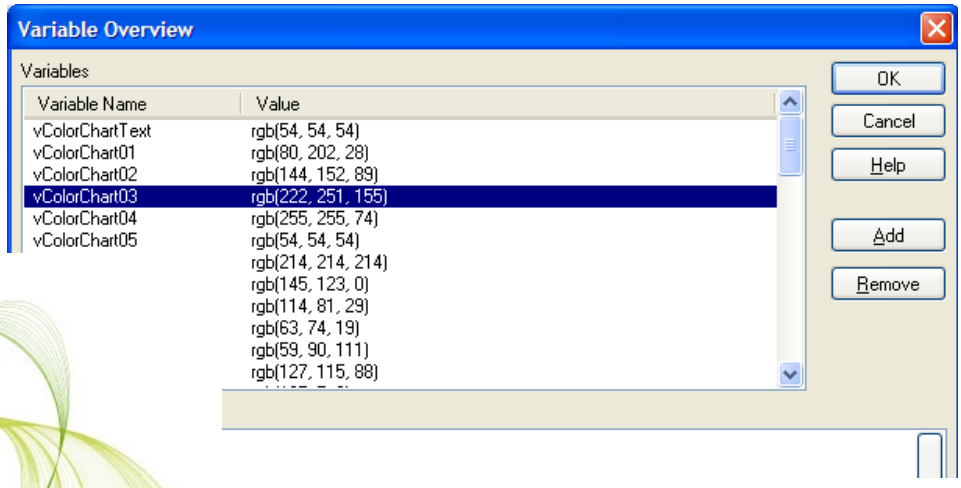


Expression Code:



The expressions overview window allows you to see expressions and conditions added to an applications charts and tables. These can be exported and saved as well.

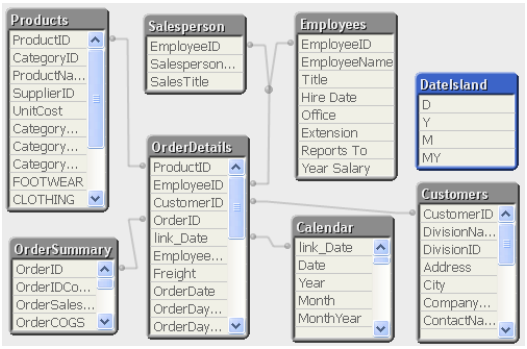
Variables:



The variables overview window shows you all variables and their values in an application. You can add/remove and edit from this window as well.

Data Model:

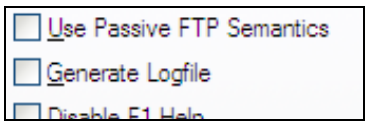
[4]



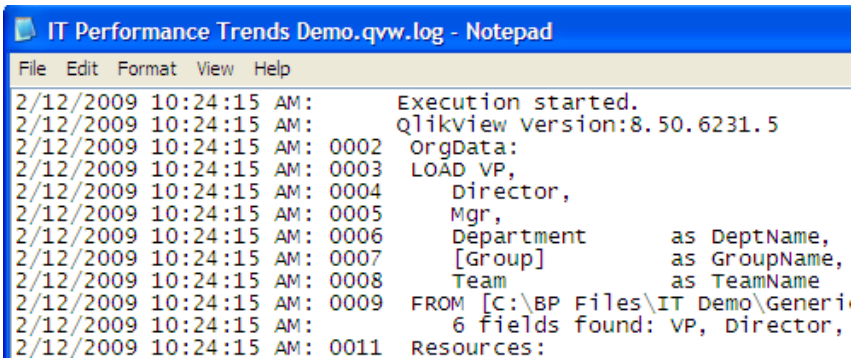
In addition to table diagrams that show the QlikView application's data model, you can export the diagrams as well as the data structure metadata. Once exported you will see three files like those below:

Tables.tab - this file holds metadata on the tables in the data model (number of columns, records, etc..)
Mappings.tab – this file holds metadata on the mapping of columns to tables in the data model
Fields.tab – this file holds metadata on the columns themselves (names, value counts, cardinality, etc...)

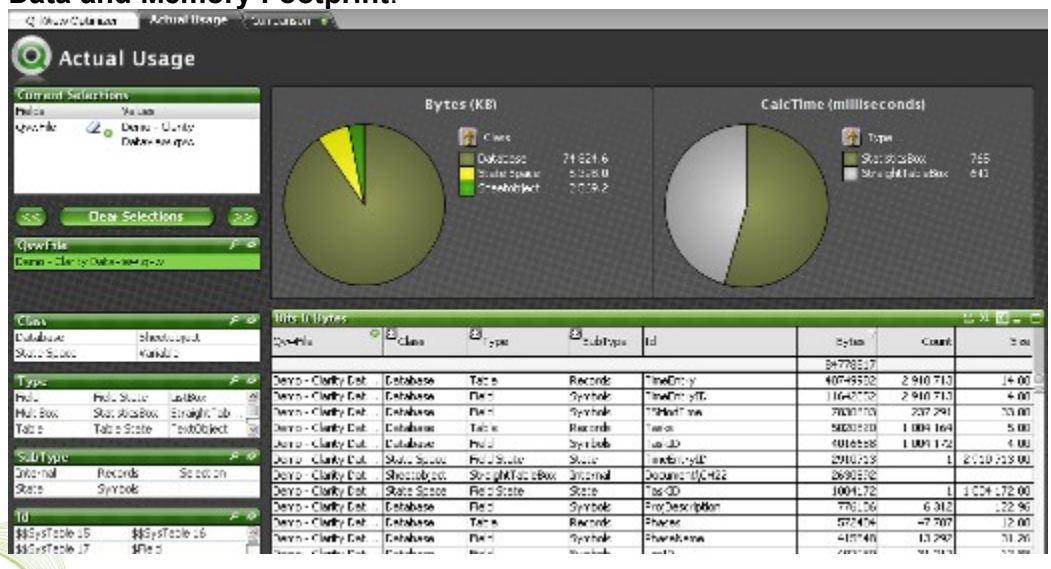
Script Log Data:



Setting the Generate Logfile option will generate a detailed log of all lines performed in the script of the application. From this metadata you can track the duration of tasks and capture any warnings or errors that occurred.



Data and Memory Footprint:

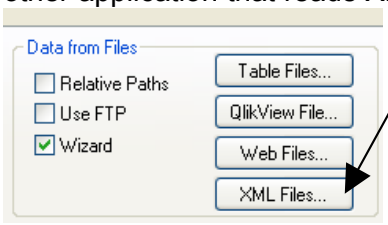


Download and use the QlikView Optimizer application (a QlikView app) that analyzes the memory output file that is optionally generated when you run a QlikView application. This application shows you the behaviors and footprints of the objects and spaces in your application with actual usage recorded. This application metadata can be very useful when optimizing applications or monitoring application behaviour for troubleshooting purposes.

Structural Metadata

QVD files have several pieces of embedded metadata that can easily be extracted as XML data. In addition, best practices in QVD architecture are to use a separate QlikView application to generate QVDs. This will open up more possibilities for creating, storing and retrieving additional QVD metadata for analysis, control and monitoring.

Embedded QVD File Metadata:
Each QVD file contains embedded XML data that can be read into a QlikView application or any other application that reads XML formatted data.



Simply use the XML Files button in the scripting window and instead of selecting an .XML file you select a .QVD file. QlikView will do the rest! What you will get will be a two-table structure that holds the XML header data from the QVD file. See the example below.

QVD Table Data

TableName	Length	NoOfRecords	QvBuildNo	CreateUtcTime	Offset
FinalCustomers	6156	684	3804	2007-04-21 09:42:05	32135

QVD Field Data

TableName	FieldName	NoOfSymbols	Length	Offset	Type	Bias	BitOffset	BitWidth	nDec	UseThou
FinalCustomers	-	-	6,156	32,135	-	-	-	-	-	-
-	Address Number	683	6,830	10,683	1	0	22	10	0	0
-	Business Family	6	24	14	0	0	10	3	0	0
-	Business Unit	1	7	32,094	0	0	0	0	0	0
-	Category Code 5	1	5	83	0	0	0	0	0	0
-	Category Code 7	2	8	133	0	0	65	1	0	0
-	Channel	8	33	50	0	0	46	3	0	0
-	Consolidated	482	4,999	27,075	0	0	13	9	0	0
-	Customer	684	10,353	330	0	0	0	10	0	0
-	Customer Number	683	9,562	17,513	0	0	32	10	0	0
-	Customer Type	3	12	38	0	0	42	2	0	0

There are quite a few helpful pieces of metadata in these XML tables, including record counts, data types, lengths, distinct values and more. Since this header data has its state maintained with the QVD data, it does not store historical values. In order to retain this information and add it to subsequent refreshes of this QVD data we need to add some simple data capture logic, as described in the next section.

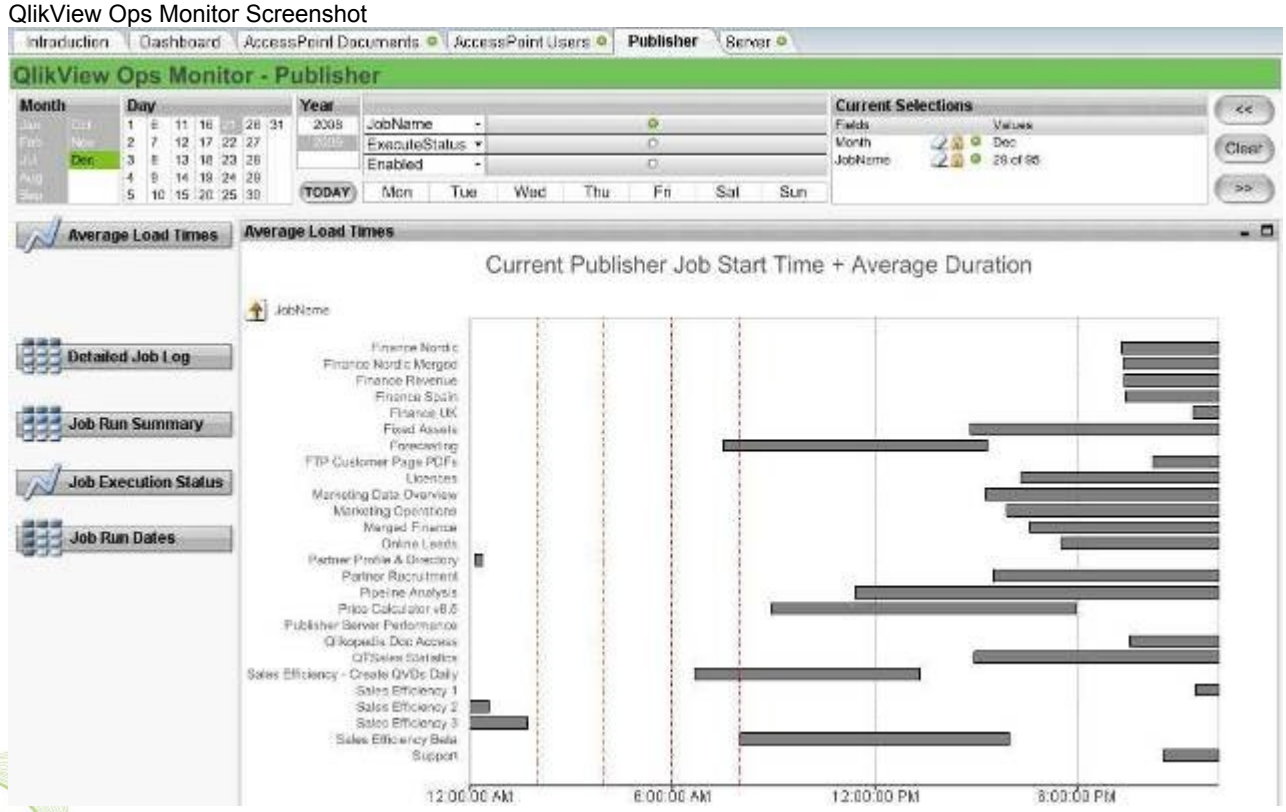
Administrative Metadata

QlikView Ops Monitor Application:
This application connects data points from QlikView Server, Publisher and Active Directory to allow for monitoring of a QlikView environment in a holistic way. Once connected to active data sources it can give current day and historical views of performance, usage and alerts. There are 5 tabs, including Dashboard, Access Point Documents, Access Point Users, Publisher and Server.

Note the setup tab in the main script. This is where you will connect to your QVS directories that contain log and session files. Also note that this application makes use of archived QlikView Session and Log files in QVD files. You can eliminate these QVDs and just use the Session and Log files but you may see performance hits over time as these files become larger. The "Users" tab contains the connections to Active Directory and other proprietary organizational data. You can remove this tab but will need to remove some selections and field references in the application. Consider fitting this logic to your clients directory services and/or organizational data. The Publisher tab also makes use of QVD archives of jobrun data from Publisher. This can be skipped if no archiving to QVD is being done.

Consider using this application if the monitoring of usage and users is beneficial to your QlikView implementation, especially if an Administrator is assigned the task of monitoring many QlikView applications and jobs, across many users.

QlikView Ops Monitor Screenshot



QVD Workbook Audit Application:
While the embedded QVD data is very useful, it is also sometimes useful to know exactly how long each QVD file is taking to reload and how many rows are being loaded. These pieces of metadata are very helpful in forming an optimized data load strategy. They can also be used to troubleshoot batch reload issues that may occur when source table indexes break or significant data growth exceeds planned growth. In order to accomplish this, a few steps are needed to help format the additional data to be stored. Note: this approach requires the two-tiered architecture referred to earlier.

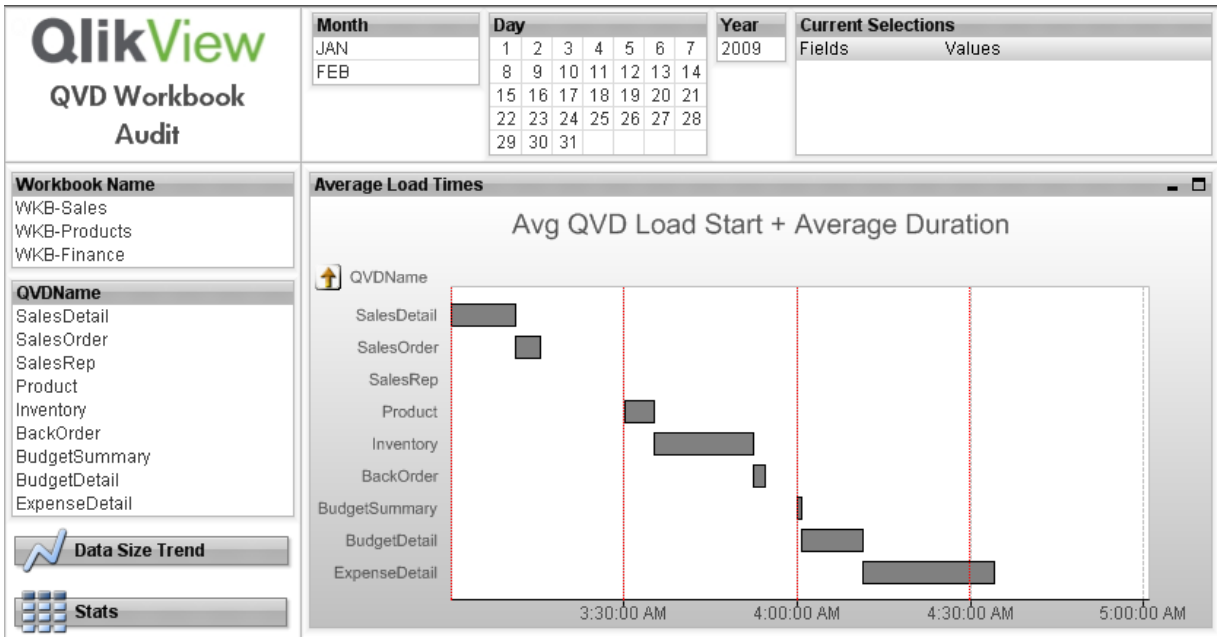
As mentioned previously, we use the term "workbooks" for QlikView applications that are used to retrieve data from source systems and write out QVDs. These applications are not end-user applications. They simple feed the data layer of your QlikView architecture. These are part of the best practices for two-tiered QlikView development. Below is an example of the code that can be used to help collect metadata at the time of retrieval. This code would be added to each of the QVD's load statements in your workbooks. You may have several workbooks in your final architecture, each with several QVD load statements that hold this code.

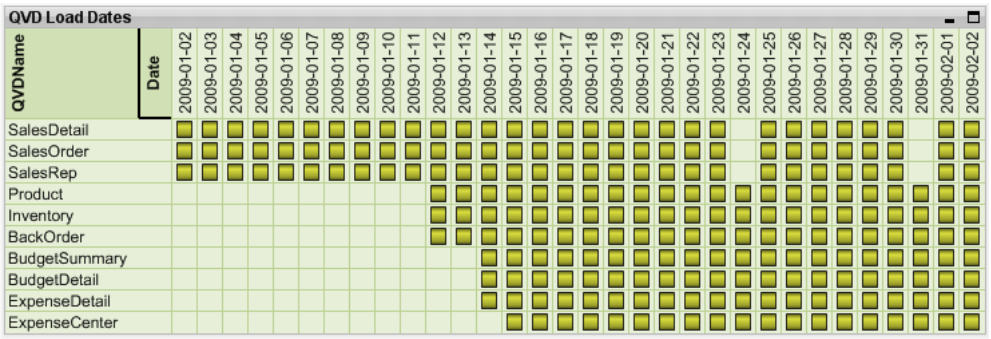
```
LET vStart = Now();

//=======================================
// Sales Budget data from MS Access DB
//=======================================
ODBC CONNECT TO [MS Access Database;DBQ=C:\BP Files\IT Demo\sales budget.mdb];
SalesBudget:
Select *
 from [Sales Budget]
 WHERE [Prod Line code] <> '406'
   AND [Prod Line code] <> '046';

LET vEnd = Now();
LET vRows  = NoOfRows('SalesBudget');

// Now add the metadata for this QVD to the WB_Audit table
CONCATENATE (WB_Audit)
LOAD * INLINE [
WorkbookName, QVDName, StartTime, EndTime, Rows
WBTest, SalesBudget, $(vStart), $(vEnd), $(vRows)];
```

At the end of each workbook's scripting simply use the STORE command to store the new values into a QVD called *workbookname*.QVD (which were concatenated since we loaded this QVD at the beginning of the workbook script).

You will end up with one of these workbook QVD files for each workbook. They are added to for each load of the workbook so you get an accumulated history of the QVD metadata that can be used for trending and analysis. A sample application called QVD Workbook Audit shows you how to read in these QVDs and create your own QVD audit analysis application. See the screen shot below.

QVD Workbook Audit Screenshots



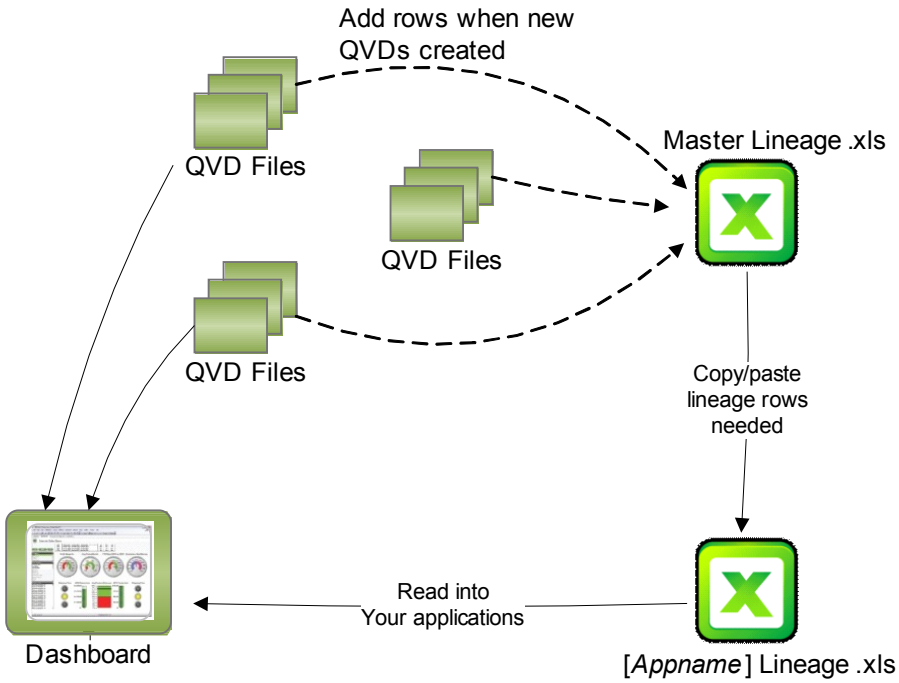


Data Lineage Demo Application:
This application demonstrates a technique for documenting the data lineage of the fields within an application. It starts with creating the data lineage fields for each QVD file as you create new QVDs. Add these to a master QVD spreadsheet that tracks the sources of all QVD fields. This should only take 20-30 minutes per QVD built, but it pays off repeatedly as you develop applications so it's a great investment. Refer to the QVD Audit Workbook application to see one use of this valuable metadata.

Once you have a master QVD lineage spreadsheet built (see the Master QVD Lineage.xls example in the Metadata folder for an example) you can easily copy/paste the fields you need into an "ApplicationName Lineage.xls" file to hold an application's data lineage information. See the Demo Lineage.xls spreadsheet for an example of this. You will need to add any fields that were directly retrieved in the application script to this spreadsheet, but if you're following a good two-tiered architecture there shouldn't be many (if any) fields to add manually.

Once that is built you connect your application to its newly built lineage spreadsheet and copy/paste the tab from the Data Lineage Demo.qvw as a starting point for your lineage tab.

Here's a picture of the architecture of this solution:



Alternate Architecture:
You can also reduce this architecture to a single spreadsheet by using an "App ID" column for each application that uses a column in the master spreadsheet. This would require you to build a crosstable of the master spreadsheet and then select only the rows with your App ID in your application. Note: this can be cumbersome if applications also retrieve non-QVD columns, requiring the list of custom fields in the master spreadsheet to grow considerably.

Summary

QlikView supplies you with application, usage and data model metadata needed to effectively tell the story of your BI environment. QlikView does not provide a separate middleware layer to track metadata and direct you on how to use the metadata. This can be very liberating to clients who simply want the metadata without a prescribed solution for analyzing it and integrating it into monitoring applications and contextualized formats.

In order to utilize metadata down the road in your QlikView deployment, consider the use of a two-tiered development architecture, taking advantage of QVDs and workbooks to organize their retrieval. Also consider documenting the QVD data lineage so that it can easily be loaded into an application-based spreadsheet or table and used within your QlikView applications for data lineage and traceability. Finally, utilize the embedded QVW metadata at your disposal from within the application.