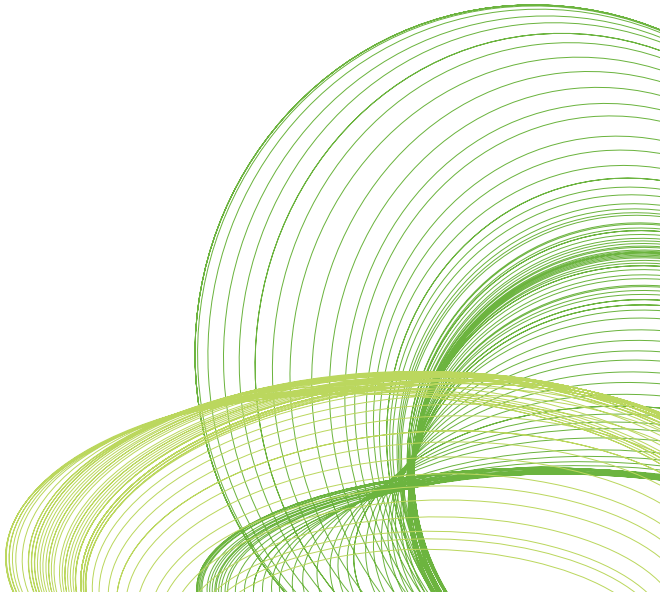
QlikView

METADATA DRIVEN QLIKVIEW APPLICATIONS AND POWERFUL DATA INTEGRATION

QlikView Expressor

A QlikView Technical Brief Document



October 2012

qlikview.com

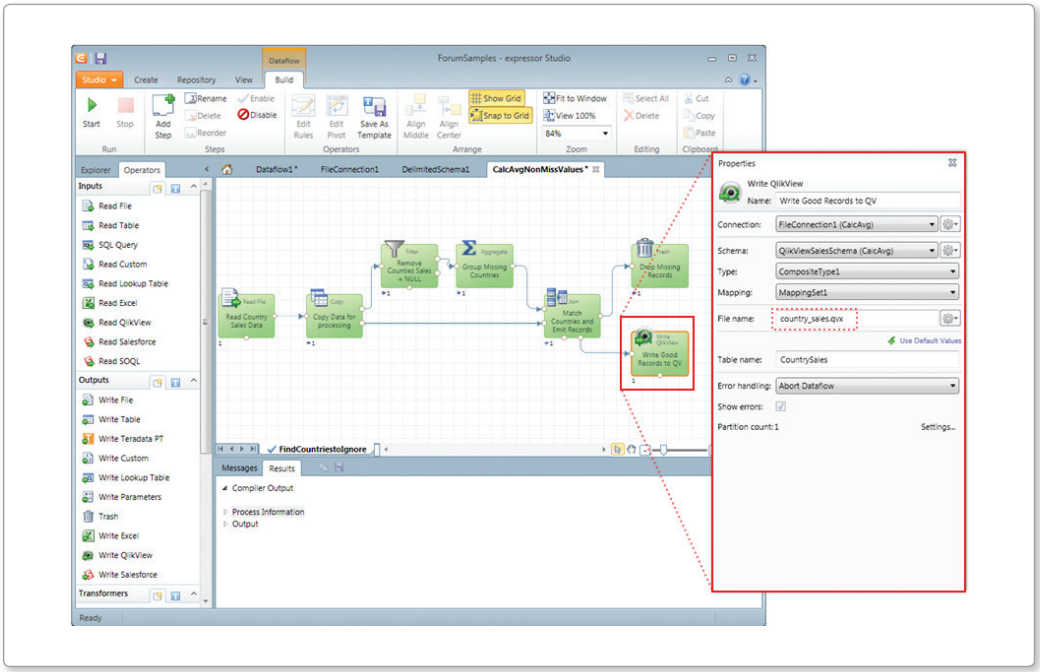
Introduction

This technical brief highlights a subset of capabilities and approaches that are to be considered when developing *metadata-driven* QlikView applications using QlikView Expressor. The information provided will cover core value and benefits when using QlikView Expressor for data preparation used by QlikView applications as compared to accessing, transforming and loading data natively using QlikView. The technical brief is strictly to inform QlikView developers of an alternative approach and additional functionality available to them when developing QlikView applications for large scale QlikView deployments. It is not meant to compare or contrast QlikView Expressor data provisioning versus native QlikView scripting nor recommend any specific one approach.

QlikView Expressor Dataflows

Simply stated, QlikView Expressor provides a rich development studio to provision data for QlikView and other data targets. Creating a QlikView Expressor Dataflow from a list of graphical operations that access, transform and load data - makes it easy to see where the data is coming from, how it is being transformed and where it is going. Property panels guide the developer in selecting available options allowing simple configuration of most operators. While developing a Dataflow, a layer of active metadata is defined about the data it is describing, resulting in a reusable collection of common business definitions and rules that can be applied across all integration and QlikView application projects. QlikView Expressor calls this reusable collection of active metadata a Semantic Type.

Figure 1: QlikView Expressor Desktop and Dataflow loading data into QlikView

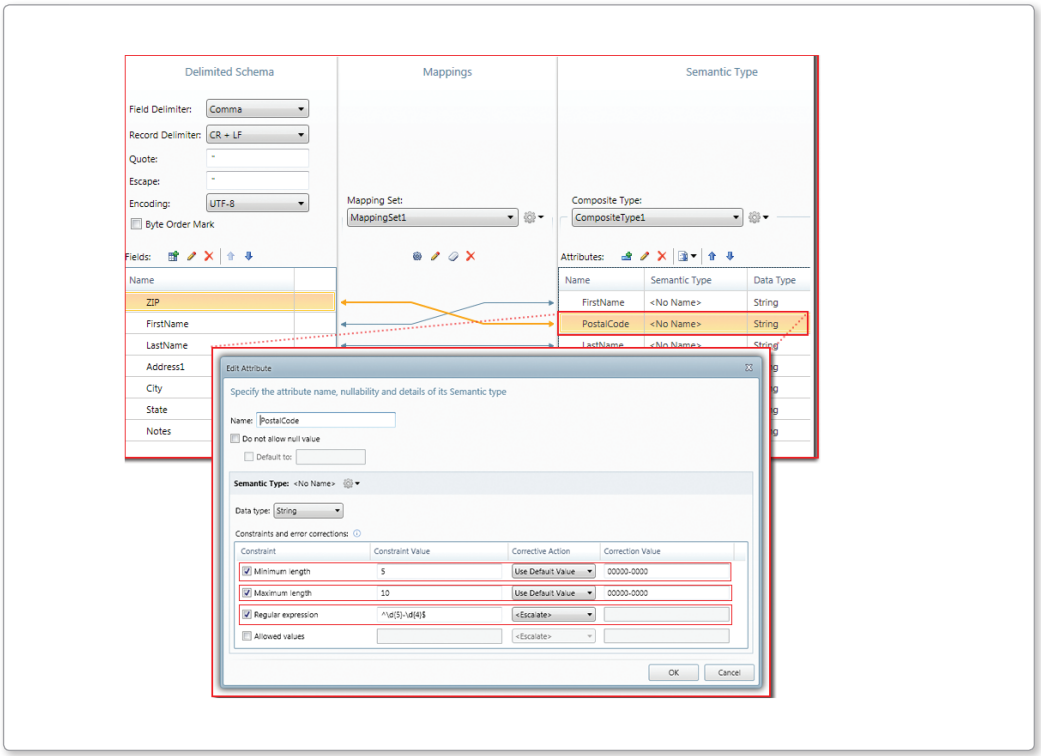


Active Metadata – Semantic Type

Metadata is defined as data about data. Active metadata can be defined as metadata in action. This active metadata layer used in QlikView Expressor is called a Semantic Type. Traditional Business Intelligence metadata is mostly static and limited to only describing a few properties about the data it refers to such as data type, column name, length and format. QlikView Expressor metadata is not static, it's actionable. It describes common properties on the source and target data while actively respecting data validation rules that have been defined on its attributes.

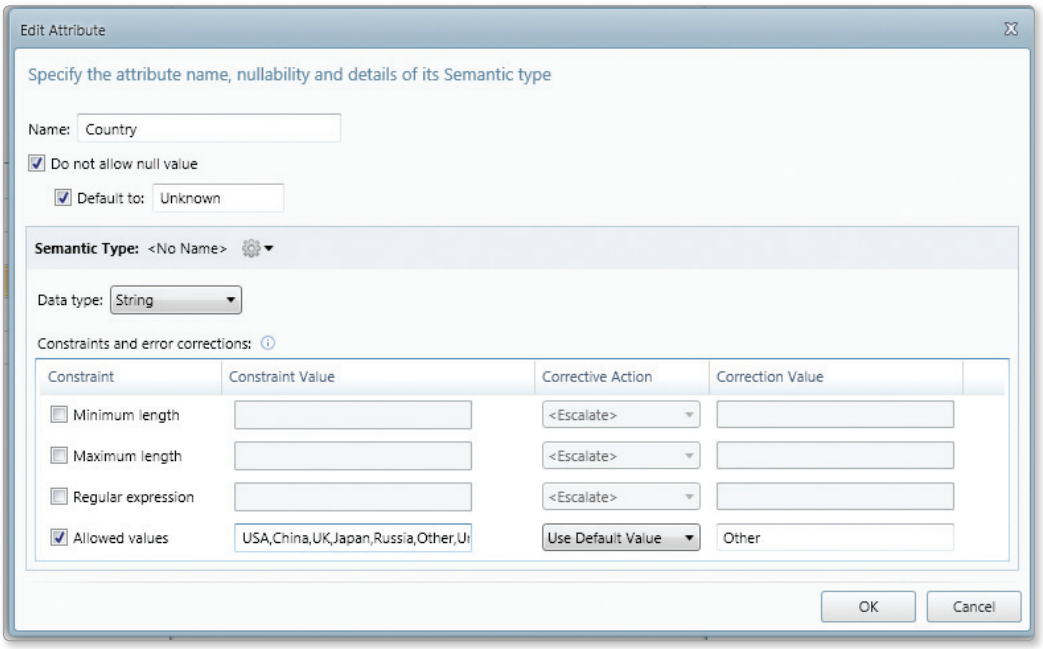
Let's take a simplistic example using a string column named ZIP used to store a 5 digit +4 zip code. Not only can the metadata describe the attribute using a common business term such as PostalCode, but it can also specify a minimum and maximum string size constraint to ensure the data length only falls within a specified range. Furthermore, it can provide a pattern match rule using a regular expression such as ^\d{5}-\d{4}$. This will ensure that the data flowing to QlikView will absolutely match the XXXXX-XXXX pattern. - What if the value fails the rule(s) you might ask? A choice to set a corrective action or redirect the record to another part of the flow is available - allowing more control on how and where the data is to be processed.

Figure 2: QlikView Expressor Semantic Type and Edit Attribute dialog showing constraints and corrective actions



Other examples include using an allowable list of values, setting rounding / min / max / precision / scale / constraints on numeric values, date range validation, date formatting and string padding / truncation. The benefit to creating active metadata such as this is that instead of defining individual validation and redirect rules within script or within each individual application, active metadata can be defined once and reused across all applications that require these attributes for decision making. And by storing the metadata in the QlikView Expressor version control repository it becomes reusable for other QlikView Expressor projects and deployments which is especially handy in a multi-developer environment.

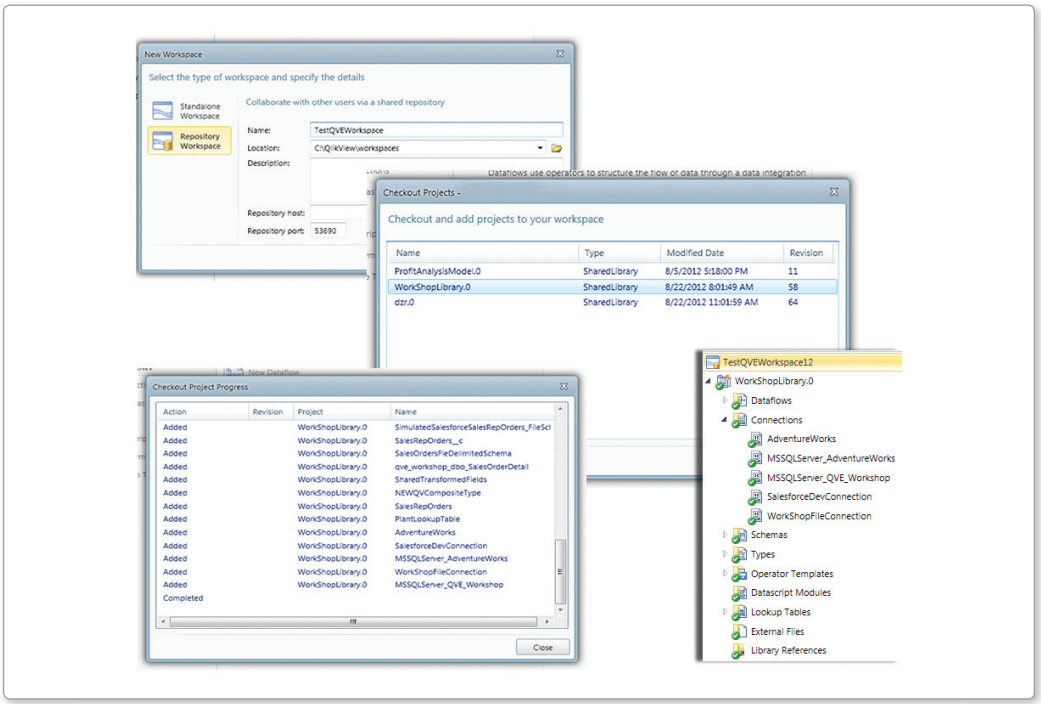
Figure 3 - A validation rule allowing only certain values



Reuse, Storage and Sharing

In order to speed up a project's development and reduce its implementation time, it is important to leverage common work components that have already been developed. Within QlikView Expressor – common project components that can be reused across all applications are Semantic Types, Schemas, Lookup Tables, Connections, Operators and Datascript modules. If there are projects that use common data sources, columns, expressions and business rules it makes absolute sense to create those elements once and share them with other developers. QlikView Expressor provides a combination of project storage options for its Dataflow building blocks known as artifacts. Local developers can simply choose to use a file system workspace where all projects and artifacts are stored locally for reuse on their system. They can enable simple sharing of their projects components by exporting them to an archive that can be imported by another developer. A more streamlined approach would be to use the QlikView Expressor Repository Workspace. This offers a centralized storage and version control system for storing QlikView Expressor artifacts. Developers can securely connect to their repository server from anywhere and check out the libraries with the artifacts they need. Furthermore, if certain updates are needed – they can occur in one place and be applied to all the applications easily with a simple update or check in. A great use of the centralized Repository Workspace is to check out and reuse pre-configured Operator Templates to apply a common set of business rules to any QlikView application.

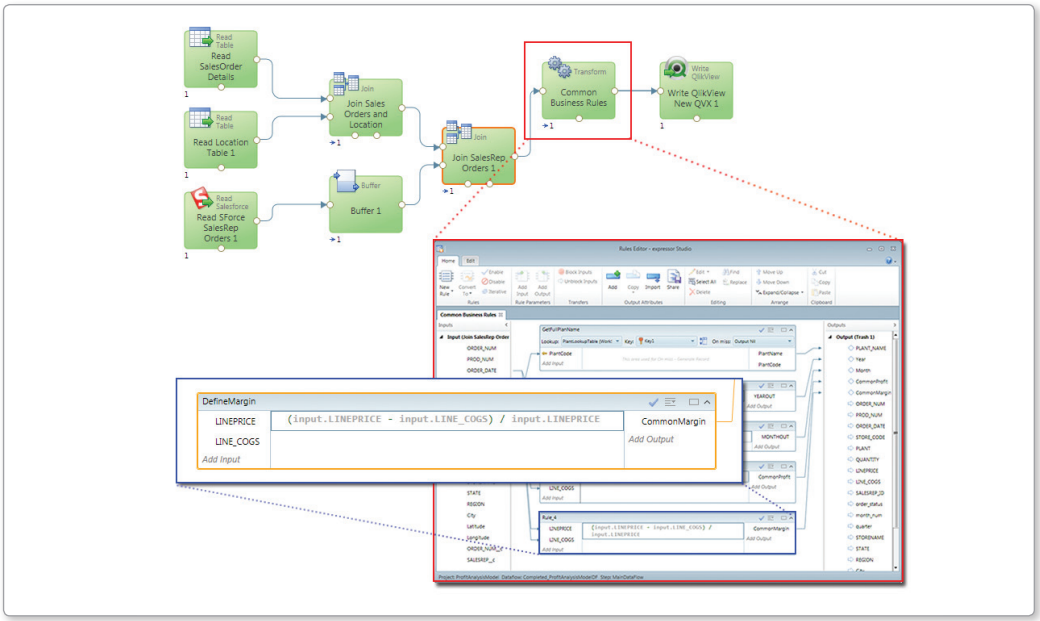
Figure 4: Checking out a library and its artifacts from the QlikView Expressor Repository Workspace



Reusable Business Rules and Operator Templates

Another way to provide greater time to value when using QlikView Expressor is to create a reusable library of common business rules exposed as Operator Templates. QlikView Expressor Dataflows use a graphical operation to build business and transformation rules called a Transform operator. The Transform operator is an all-inclusive component used to create new and augment existing data. It can create a collection of reusable rules that accept incoming data attributes as inputs. Transformed data is defined in expressions using Expression, Function and Lookup Rules created within the Transform Operator's Rules Editor. Rules enrich data with a variety of string, math, decision, utility and many more functions, including the ability to lookup data from Lookup Tables. Use of the Transform operator can be compared to creating expressions within QlikView load scripts or within UI sheet objects; however the operator and its already created rules can be made available as a reusable Operator Template that can be included in any QlikView Expressor dataflow. Some examples of common transformation rules may include - definitions of metrics such as profitability, churn rate or margin; data that combines multiple fields such as FNAME and LNAME to make FullName; standardized formatting and masking to properly format dates, phone numbers and social security numbers. All of the appropriate expressions to create those desired results can be include as rules in the Transform Operator, parameterized and made available as an Operator Template. The next developer to use that template would simply map the input attributes to the rule parameters and map the output attributes. Operator Templates are also commonly used to enrich incoming data with other disparate data from Lookup Tables. A clear example of this looking up location data such as geocoding an incoming state code; country, city, zip address etc.

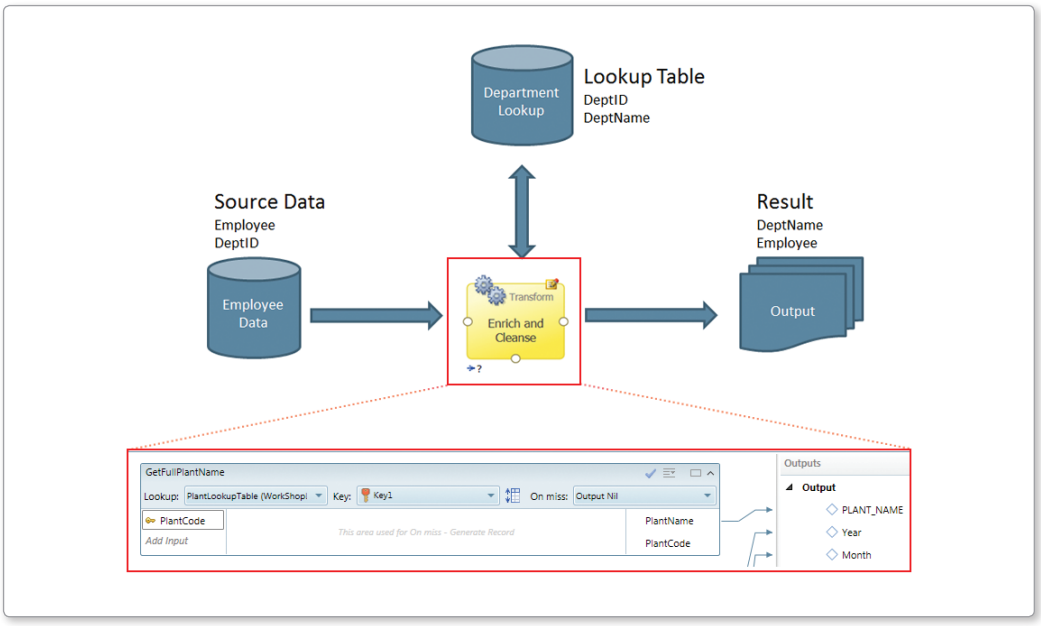
Figure 5: Dataflow, Transform Operator, Rules Editor and Rule that defines a common metric



Lookup Tables

A Lookup Table is a database table designed to serve a special, limited function within a data integration application or group of applications. Lookup Tables are usually created from a subset of data from a larger table or from a source designed to add data that an application can use. For example, a Lookup Table might be created to provide department names to data from a source that contains only department numbers. During the process of integrating data, the Lookup Table could be read to add department names to department numbers. The advantage of Lookup Tables is that they are stored within an Expressor Project and are included in Deployment Packages. Access to them is thereby made easier and faster. When their function and size are limited, accessing their data is also easy and fast. In QlikView Expressor, Lookup Expression Rules are similar to using the ApplyMap() function that is used with a MAPPING table created with a MAPPING LOAD statement in a QlikView load script. However with the Lookup Expression Rules, the developer is not limited to just two fields. Lookup Tables can have both single and composite keys and return multiple columns from them. The developer can also specify a default value if the value being looked up is not present. It is also possible to branch to another part of the Dataflow to perform another step, without having to use IF THEN ELSE scripting logic. An additional benefit is the Range Lookup feature, it makes it possible to define lookups that are constructed as numeric ranges.

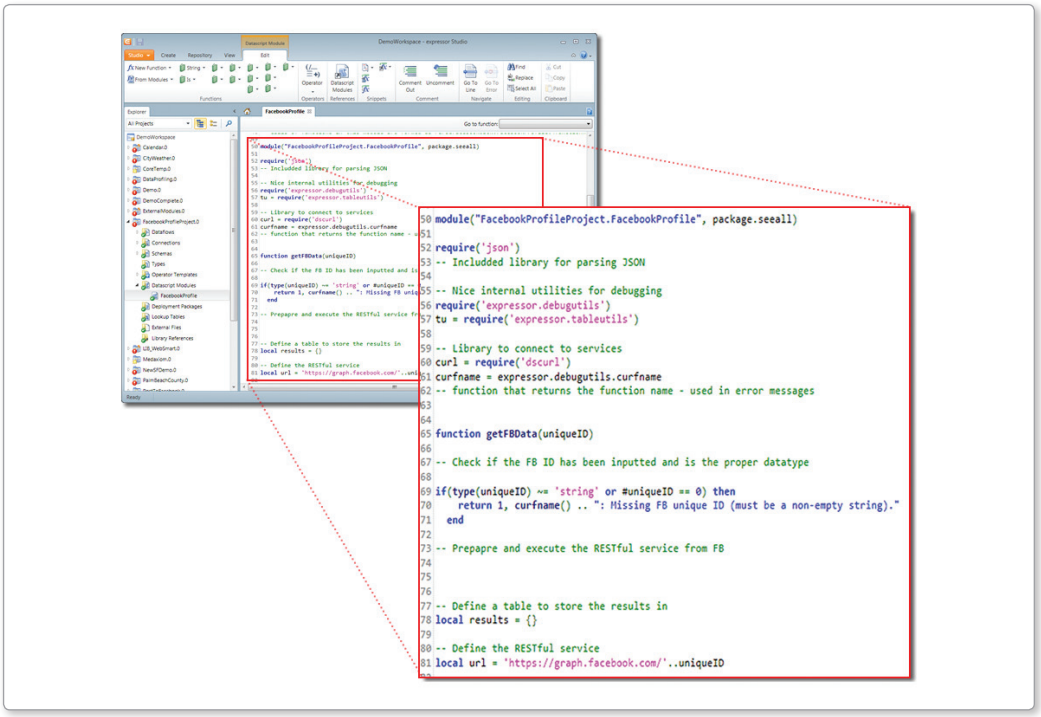
Figure 6: Lookup Table flow and Lookup Expression Rule



Extending Functionality with QlikView Expressor Datascript

Business Intelligence solutions should always provide a software development kit to help organizations create new components and extend existing functionality that is not always available inside the box. It is impossible to have *everything* that *everyone* wants available in a graphical business user or development interface. Leveraging a SDK allows customers and partners the ability to create what they need when they want it without the reliance on the software vendor. With QlikView Expressor there is an option to use a script editor depending on the complexity of the data access or transformation needed. This functionality is comparable to using QlikView script or building extensions for a QlikView application. QlikView Expressor Datascript is a lightweight, fast interpreted scripting language based on Lua. It is easy to learn and provides an extensible library of additional modules that enable custom functionality to be defined and shared across all QlikView Expressor applications. Datascript can also come in handy when certain file management and job control flow capability is needed as file system, ftp and web service type modules can be made available.

Figure 7: A Datascript module that accesses, parses and flattens Facebook generic profile data



References

QlikView Expressor Manual
http://documentation.qlikview.com/expressor/3.7/expressor_3.htm

QlikCommunity QlikView Expressor Documents
http://community.qlikview.com/community/qlikview_expressor?view=documents

Lua
http://www.lua.org/

http://en.wikipedia.org/wiki/Lua_(programming_language)