QVX File Format

and QlikView Custom Connector

Contents
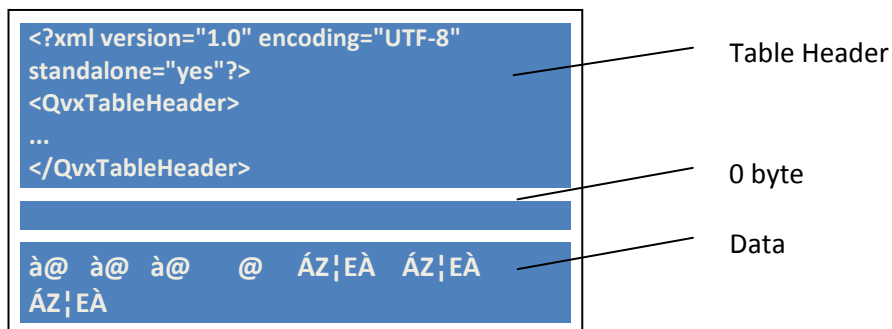
1   QVX and QlikView Custom Connector

QVX (QlikView data eXchange) is a new file/stream format for high performance data input into QlikView. A QVX formatted file contains metadata describing a table of data and the actual data. In contrast to the QVD format, which is proprietary and optimized for minimum transformations inside QlikView, the QVX format is public and requires a few transformations when exporting data from traditional data base formats.

QlikView Custom Connector (Connector) is an interface developed by customers that enables data retrieval from a custom data source. Earlier versions of QlikView supported Connectors implemented as dynamic-link libraries (DLL). Starting with QlikView 10 it is possible and preferred to implement Connectors as separate applications that reply to QlikView's requests to connect to a data source and to retrieve data, as well as stream QVX formatted data to QlikView. Connectors are launched as separate processes when QlikView needs to retrieve data from custom data sources.

2   QVX File Format

A QVX formatted file describes a single table followed by the actual table data. The file includes:

- An XML formatted table header (QvxTableHeader) written in UTF-8 character set. The header describes the fields in the table, the layout of the subsequent data together with some other meta-data.
- 0 byte written at the end of the table header, i.e. directly after </**QvxTableHeader**>, and before writing the actual data.
- The actual data formatted according to the preceding table header.

<?xml version="1.0" encoding="UTF-8" standalone="yes"?> <QvxTableHeader> ... </QvxTableHeader>

Table Header

0 byte

Data

à@ à@ à@   @   ÁZ¦EÀ ÁZ¦EÀ ÁZ¦EÀ

2.1 QvxTableHeader XML Schema

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
 <xs:simpleType name="QvxFieldType">
   <xs:restriction base="xs:string">
    <xs:enumeration value="QVX_SIGNED_INTEGER"/>
    <xs:enumeration value="QVX_UNSIGNED_INTEGER"/>
    <xs:enumeration value="QVX_IEEE_REAL"/>
    <xs:enumeration value="QVX_PACKED_BCD"/>
    <xs:enumeration value="QVX_BLOB"/>
    <xs:enumeration value="QVX_TEXT"/>
   </xs:restriction>
 </xs:simpleType>
 <xs:simpleType name="QvxFieldExtent">
   <xs:restriction base="xs:string">
    <xs:enumeration value="QVX_FIX"/>
    <xs:enumeration value="QVX_COUNTED"/>
    <xs:enumeration value="QVX_ZERO_TERMINATED"/>
   </xs:restriction>
 </xs:simpleType>
 <xs:simpleType name="QvxNullRepresentation">
   <xs:restriction base="xs:string">
    <xs:enumeration value="QVX_NULL_NEVER"/>
    <xs:enumeration value="QVX_NULL_ZERO_LENGTH"/>
    <xs:enumeration value="QVX_NULL_FLAG_WITH_UNDEFINED_DATA"/>
    <xs:enumeration value="QVX_NULL_FLAG_SUPPRESS_DATA"/>
   </xs:restriction>
 </xs:simpleType>
 <xs:simpleType name="FieldAttrType">
   <xs:restriction base="xs:string">
    <xs:enumeration value="UNKNOWN"/>
    <xs:enumeration value="ASCII"/>
    <xs:enumeration value="INTEGER"/>
    <xs:enumeration value="REAL"/>
    <xs:enumeration value="FIX"/>
    <xs:enumeration value="MONEY"/>
    <xs:enumeration value="DATE"/>
    <xs:enumeration value="TIME"/>
    <xs:enumeration value="TIMESTAMP"/>
    <xs:enumeration value="INTERVAL"/>
   </xs:restriction>
 </xs:simpleType>
 <xs:complexType name="FieldAttributes">
  <xs:all>
   <xs:element name="Type" type="FieldAttrType" />
   <xs:element name="nDec" type="xs:integer" minOccurs="0" />
   <xs:element name="UseThou" type="xs:integer" minOccurs="0" />
```

```
    <xs:element name="Fmt" type="xs:string" minOccurs="0" />
    <xs:element name= "Dec" type="xs:string" minOccurs="0" />
    <xs:element name= "Thou" type="xs:string" minOccurs="0" />
  </xs:all>
 </xs:complexType>

 <xs:element name="QvxTableHeader">
  <xs:complexType>
   <xs:all>
    <xs:element name="MajorVersion" type="xs:integer" />
    <xs:element name="MinorVersion" type="xs:integer" />
    <xs:element name="CreateUtcTime" type="xs:dateTime" minOccurs="0" />
    <xs:element name="TableName" type="xs:string" />
    <xs:element name="UsesSeparatorByte" type="xs:boolean" " minOccurs="0" />
    <xs:element name="BlockSize" type="xs:integer" minOccurs="0" />
    <xs:element name="Fields">
       <xs:complexType>
        <xs:sequence>
         <xs:element name="QvxFieldHeader" maxOccurs="unbounded">
          <xs:complexType>
           <xs:all>
            <xs:element name="FieldName" type="xs:string" />
            <xs:element name="Type" type="QvxFieldType" />
            <xs:element name="Extent" type="QvxFieldExtent" />
            <xs:element name="NullRepresentation" type="QvxNullRepresentation" />
            <xs:element name="BigEndian" type="xs:boolean" minOccurs="0" />
            <xs:element name="CodePage" type="xs:integer" minOccurs="0" />
            <xs:element name="ByteWidth" type="xs:integer" minOccurs="0" />
            <xs:element name="FixPointDecimals" type="xs:integer" minOccurs="0" />
            <xs:element name="FieldFormat" type="FieldAttributes" />
           </xs:all>
          </xs:complexType>
         </xs:element>
        </xs:sequence>
       </xs:complexType>
      </xs:element>
   </xs:all>
  </xs:complexType>
 </xs:element>
</xs:schema>
```

2.1.1 QvxTableHeader Element

The QvxTableHeader element specifies data formatting information for a single table.

Child Element	Description
MajorVersion	Version number for the QVX format. Value - 1.
MinorVersion	Version number for the QVX format. Value - 0.
CreateUtcTime	Data formatting time specified in Coordinated Universal Time (UTC) time standard.
TableName	String defining source of the data, e.g. executed SQL statement.
UsesSeparatorByte	Flag specifying if extra byte for separating records is used. When record separator is used: • Record separator byte RS (0x1E) is written before every record. • File separator byte FS(0x1C) is written after the last record and indicates the end of file (EOF). • Null byte NUL(0x0) is used to pad to the next block boundary (used when BlockSize is given). Record separators have to be used together with BlockSize, otherwise their use is optional.
BlockSize	The specified block size (>1, _int64) together with UsesSeparatorByte define a block structure to store formatted data. The use of block structure is optional and enables parallel load of the data. Blocks, except of the first one, are aligned to the whole file.
Fields	Contains QvxFieldHeader specified for every table field. See **QvxFieldHeader**.

2.1.2 QvxFieldHeader Element

The QvxFieldHeader element describes a single table field. It specifies the field name, format used to write the field values in the QVX formatted file, as well as how QlikView should interpret the values.

The elements Type, Extent, NullRepresentation, BigEndian, CodePage and ByteWidth describe how the field data values are written in the QVX formatted file.

FixPointDecimals and FieldFormat can be used to specify for QlikView on how to interpret the read data values.

Child Element	Description
FieldName	Field name.
Type	Data type in which field values are written in the QVX formatted data stream. See **QvxFieldType**.
Extent	Method used to define field value length. See **QvxFieldExtent**.
NullRepresentation	Method used to handle Null values. See **QvxNullRepresentation**.
BigEndian	Order of bytes (with respect to significance) in data stream. It may be used for binary fields, i.e. QVX_SIGNED_INTEGER, QVX_UNSIGNED_INTEGER and QVX_REAL, also it may be used for byte count in QVX_COUNTED fields. If not specified LittleEndian is assumed.
CodePage	Character encoding used to write field values in the QVX data stream. 1200/1201 implies UTF16. Byte order mark (BOM) is not allowed. BigEndian has no effect on UTF16. If not set then UTF8 is assumed.
ByteWidth	In case of QVX_FIX fields, it is a size of the actual data: • 1,2,4 or 8 for QVX_INTEGER_SIGNED and QVX_UNSIGNED_INTEGER. • 4 or 8 for QVX_IEEE_REAL. • Arbitrary size for other QVX field types. In case of QVX_COUNTED fields, it is a size of the byte count value, which is used to store a size of the actual data.
FixPointDecimals	Fixed number of digits to the right of the radix point. Used together with QVX_SIGNED_INTEGER, QVX_UNSIGNED_INTEGER, QVX_PACKED_BCD. For instance, value 1234 is interpreted as 12.34 with FixPointDecimals = 2 and is interpreted as 123400 with FixPointDecimals = -2.
FieldFormat	Format specifies how QlikView should interpret the streamed data values. See **FieldAttributes Type**.

2.1.3 QvxFieldType Type

Data type in which field values are stored in QVX formatted data stream.

Type Value	Description
QVX_SIGNED_INTEGER	An integer value is passed in normal 2-complement binary representation. The exact layout of the value is determined by the BigEndian flag and a ByteWidth (of 1,2,4 or 8).
QVX_UNSIGNED_INTEGER	An unsigned integer value is passed in normal binary representation. The exact layout of the value is determined by the BigEndian flag and a ByteWidth (of 1, 2, 4 or 8).
QVX_IEEE_REAL	A floating point number is passed in IEEE 754-2008 standard formats (binary32 or binary64). The exact layout of the value is determined by the BigEndian flag and a ByteWidth (of 4 or 8).
QVX_PACKED_BCD	Data is passed as a packed BCD (Binary Coded Decimal) number - two digits per byte. Low nybble (4 bits) of last byte of 0xB or 0xD means negative, 0xA, 0xC, 0xE, 0xF means positive, 0-9 is extra digit. Extra leading digit positions are 0-filled.
QVX_BLOB	Binary data interpreted as BLOB. Limited use in QlikView.
QVX_TEXT	Data is a text string that may be interpreted as a number by QlikView depending on the supplied **FieldAttrType**. CodePage defines the encoding. 1200/1201 means UTF-16 and also defines its byte order. When the extent is QVX_FIX, binary trailing zero-padding should be used.
QVX_QV_DUAL	A flag byte, possibly followed by a number and /or a string. First comes one flag byte of type QvxQvSpecialFlags. Depending on the flag, there can follow a number and/or a string. The number may be an integer or a double. The string is always in UTF-8 or UTF-16 format and zero terminated. **Its recommended to use one of the other types if possible.**

QvxQvSpecialFlag	**Only the listed flag combinations are allowed.**
	
	QVX_QV_SPECIAL_NULL = 0, // No data follows after this
	QVX_QV_SPECIAL_INT = 1, // An integer after this
	QVX_QV_SPECIAL_DOUBLE = 2, // A double after this
	QVX_QV_SPECIAL_STRING = 4, // String is zero terminated and in UTF-8 or UTF-16
	QVX_QV_SPECIAL_INT_AND_STRING = 5, // Integer before string
	QVX_QV_SPECIAL_DOUBLE_AND_STRING = 6, // Double before string
	
	The possible combinations are limited to QVX_QV_SPECIAL_INT_AND_STRING and QVX_QV_SPECIAL_DOUBLE_AND_STRING. The string must be a zero terminated UTF-8 or UTF-16.

2.1.4   QvxFieldExtent Type

Method used to define field value length.

Type Value	Description
QVX_FIX	The byte count for each field value is constant and given by ByteWidth.
QVX_COUNTED	Each field value is preceded by a (unsigned) byte count of the actual data. The layout of the count is determined by the BigEndian flag and a ByteWidth.
QVX_ZERO_TERMINATED	Can be used for text fields. Means that the extent of the field length is a terminating zero byte (or 16-bit entity for UTF-16).
QVX_QV_SPECIAL	Used with QVX_QV_DUAL.

2.1.5   QvxNullRepresentation Type

Method used to specify how Null values are handled.

Type Value	Description
QVX_NULL_NEVER	Field cannot return NULL.
QVX_NULL_ZERO_LENGTH	For field values that use QVX_COUNTED together with QVX_NULL_ZERO_LENGTH, a zero length means NULL (rather than an empty string).
QVX_NULL_FLAG_WITH_UNDEFINED_DATA	A single byte (with binary value 1 or 0) in front of the field value signifies NULL or not. A 1 means it is NULL and the field value is not used.
QVX_NULL_FLAG_SUPPRESS_DATA	A single byte (with binary value 1 or 0) signifies NULL or not. A 1 means it is NULL and no field value is passed. A 0 byte means a normal value follows.

2.1.6   FieldAttributes Type

The FieldAttributes format specifies how QlikView should interpret the streamed data values. For instance, QlikView can interpret QVX_IEEE_REAL or QVX_TEXT type data as DATE.

Child Elements	Description
Type	Data type (see **FieldAttrType**).
nDec	Fixed number for decimals for FIX type data and fixed number of significant digits for REAL type data. Allowed range(0..15).
UseThou	Flag indicating if thousand separator is used. Allowed range(0,1).
Fmt	Format code that can be used to specify format for numbers, dates, time, timestamps and time intervals

9

Dec	Decimal separator.
Thou	Thousand separator.

2.1.7   FieldAttrType Type

For more details read QlikView Reference Manual Section "Number Interpretation Functions".

Type Value	Description
UNKNOWN	Data type is unknown. If possible values are treated as numbers, otherwise as text.
ASCII	Forces values to be treated as text.
INTEGER	Values will be interpreted as integers.
REAL	Values will be interpreted as numbers with a fixed number of significant digits defined by FieldAttributes.nDec.
FIX	Values will be interpreted as numbers with a fixed number of decimals defined by FieldAttributes.nDec.
MONEY	Values will be interpreted as money according to the format string specified by FieldAttrType.Fmt.
DATE	Values will be interpreted as dates according to the format string specified by FieldAttrType.Fmt. Values streamed as QVX_IEEE_REAL should be implemented as OLE Automation date, i.e. floating-point value counting days since midnight, 30 December 1899.
TIME	Values will be interpreted as time according to the format string specified by FieldAttrType.Fmt.
TIMESTAMP	Values will be interpreted as time stamps according to the format string specified by FieldAttrType.Fmt.
INTERVAL	Values will be interpreted as time intervals according to the format string specified by FieldAttrType.Fmt.

2.2   QvxTableHeader Element Example

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<QvxTableHeader>
        <MajorVersion>1</MajorVersion>
        <MinorVersion>0</MinorVersion>
        <CreateUtcTime>2010-03-25 10:12:52</CreateUtcTime>
        <TableName>SELECT ProductID, Name, ListPrice FROM
                AdventureWorks.Production.Product
        </TableName>
        <UsesSeparatorByte>0</UsesSeparatorByte>
        <BlockSize>0</BlockSize>
        <Fields>
        <QvxFieldHeader>
                <FieldName>ProductID</FieldName>
                <Type>QVX_SIGNED_INTEGER</Type>
                <Extent>QVX_FIX</Extent>
```

```
                    <NullRepresentation>QVX_NULL_NEVER</NullRepresentation>
                    <BigEndian>0</BigEndian>
                    <Codepage>1201</Codepage>
                    <ByteWidth>4</ByteWidth>
                    <FieldFormat>
                                <Type>INTEGER</Type>
                                <nDec>0</nDec>
                    </FieldFormat>
            </QvxFieldHeader>
            <QvxFieldHeader>
                    <FieldName>Name</FieldName>
                    <Type>QVX_TEXT</Type>
                    <Extent>QVX_COUNTED</Extent>
                    <NullRepresentation>QVX_NULL_NEVER</NullRepresentation>
                    <BigEndian>0</BigEndian>
                    <Codepage>1201</Codepage>
                    <ByteWidth>4</ByteWidth>
                    <FieldFormat>
                                <Type>UNKNOWN</Type>
                                <nDec>0</nDec>
                    </FieldFormat>
            </QvxFieldHeader>
            <QvxFieldHeader>
                    <FieldName>ListPrice</FieldName>
                    <Type>QVX_IEEE_REAL</Type>
                    <Extent>QVX_FIX</Extent>
                    <NullRepresentation>QVX_NULL_NEVER</NullRepresentation>
                    <BigEndian>0</BigEndian>
                    <Codepage>1201</Codepage>
                    <ByteWidth>8</ByteWidth>
                    <FieldFormat>
                                <Type>MONEY</Type>
                                <nDec>0</nDec>
                    </FieldFormat>
            </QvxFieldHeader>
        </Fields>
</QvxTableHeader>
```

2.3   Reading Data from QVX Files

Similarly to QVD and text files, QVX files can be referenced by a load statement.

For instance:

Load * FROM C: \qvxsamples\xyz.qvx (qvx);
Load Name, RegNo FROM C: \qvxsamples\xyz.qvx (qvx);

3 QlikView Custom Connector

QlikView communicates with QlikView Custom Connector (Connector) to retrieve data from its data source. This section provides details on how the Connector is launched and on how the communication between the applications is performed.

3.1 Connector File Properties

For QlikView to recognize an exe file as Connector, the files version-information resource should include the property "QlikView Connector". Its value is used as a Connector display name and has to be specified.

There are a few ways to set file version information, e.g.:

- Version information can be defined in a resource file for a C++ program. Search for "VERSIONINFO Resource" to read more about this.
- Using third party tools, e.g. Version Resource Tool: http://www.codeproject.com/KB/install/VerPatch.aspx.

3.2 Launching Connector

QlikView starts Connector process and passes two ***command-line arguments***: parent window handle and command pipe name.

A connector process is launched:

- During script reload when a connect statement to the Connector is found. The process is terminated when another connect statement is found or script reload is completed.
- On open Edit Script dialog box. QlikView checks if the Connector has Custom Caption button and if so, then gets the caption. The process is terminated immediately afterwards.
- When the Connector is selected in the Databases list box in Edit Script dialog box. The Connector process is used for handling operations related to Connect, Select and Custom dialogs. The process is terminated on Edit Script dialog box exit or when another data source is selected.

3.3 Interprocess Communication via Named Pipes

Two types of named pipes are used for exchanging messages and data between QlikView and Connector:

- ***Command pipe*** is a named pipe that is used for exchanging XML formatted messages, i.e. QlikView requests, which describe commands to be performed by the Connector, and the Connector replies, which describe status of the executed commands. In certain cases, replies may include result data.

- **Data pipe** is a named pipe that is used by the Connector to return QVX formatted data, e.g. SQL statement result. For every QVX_EXECUTE command request a new data pipe is created.
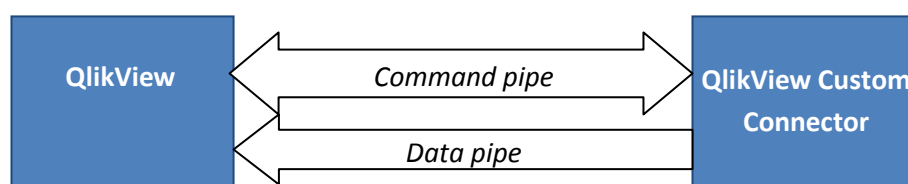
Below are example scenarios on how QlikView and the Connector exchange messages to execute
- Connect statement:
  1. QlikView sends a connect request (QVX_CONNECT) via the command pipe and waits for the Connector reply.
  2. The Connector executes the connect command and sends the reply if the connection to the data source was successful or not (QVX_OK or QVX_CONNECT_ERROR).
- Select statement:
  1. QlikView sends an execute request (QVX_EXECUTE) and waits for the Connector reply. The request includes the SQL statement and the name of a data pipe, via which QVX formatted result should be returned.
  2. After executing SQL statement, the Connector sends the reply on the execute command execution status, e.g. QVX_OK or QVX_TABLE_NOT_FOUND.
  3. The Connector starts writing SQL statement result data to the data pipe.
  4. QlikView, after receiving QVX_OK reply to the execute request, starts reading data from the data pipe.

3.4   Command Pipe

Command pipe is used to exchange QlikView requests (QVXRequest) to the Connector and the Connector replies (QVXReply) to QlikView. After sending a request QlikView waits for the Connector reply before proceeding with other actions.

This section describes format of QVXRequest and QVXReply messages.

3.4.1   QlikView Request
Data stream of QlikView request if formatted as follow:

- 4-byte that specify the length of the following string.
- Zero terminated string that is formatted according to QvxRequest XML schema.

15

3.4.1.1 QvxRequest XML Schema

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
 <xs:simpleType name="QvxCommand">
   <xs:restriction base="xs:string">
    <xs:enumeration value="QVX_CONNECT"/>
    <xs:enumeration value="QVX_EXECUTE"/>
    <xs:enumeration value="QVX_EDIT_CONNECT"/>
    <xs:enumeration value="QVX_EDIT_SELECT"/>
    <xs:enumeration value="QVX_GENERIC_COMMAND"/>
    <xs:enumeration value="QVX_DISCONNECT"/>
    <xs:enumeration value="QVX_TERMINATE"/>
    <xs:enumeration value="QVX_PROGRESS"/>
    <xs:enumeration value="QVX_ABORT"/>
   </xs:restriction>
 </xs:simpleType>

 <xs:complexType name="QvxConnectOptions">
  <xs:all>
   <xs:element name="Provider" type="xs:string" />
   <xs:element name="LoginTimeoutSec" type="xs:integer" />
   <xs:element name="ConnectTimeoutSec" type="xs:integer" />
   <xs:element name="AutoCommit" type="xs:boolean" />
   <xs:element name="ReadOnly" type="xs:boolean" />
   <xs:element name="AllowPrompt" type="xs:boolean" />
  </xs:all>
 </xs:complexType>

 <xs:element name="QvxRequest">
  <xs:complexType>
   <xs:all>
    <xs:element name="Command" type="QvxCommand" />
    <xs:element name="Parameters">
     <xs:complexType>
      <xs:sequence>
       <xs:element name="String" type="xs:string" maxOccurs="unbounded"/>
      </xs:sequence>
     </xs:complexType>
    </xs:element>
    <xs:element name="Options" type="QvxConnectOptions" minOccurs="0" />
   </xs:all>
  </xs:complexType>
 </xs:element>
</xs:schema>
```

QvxRequest Element

Child Element	Description
Command	Command to be executed by the Connector. See **QvxCommand**.
Parameters	Array of strings. Holds command specific parameters.
Options	Connection options. See **QvxConnectOptions**.

QvxCommand Type

Type Value	Description
QVX_CONNECT	Connect to the data source using the given connect string. The command is sent during script reload while executing CUSTOM CONNECT statement. **Request parameters:** **[0]** - a modified connect statement: a) rprovider parameter and value are removed; b) scrambled XUserId and XPassword are substituted with unscrambled UserId and Password parameters. **Request options:** Include connection specific settings. **Reply:** • QVX_OK, if connection to the data source is established successfully.

QVX_EXECUTE	Execute the given statement and return the data via data pipe.
	 The command is sent during script reload and by Select Wizard. SELECT statement is used for fetching actual data from the data source. TABLES, COLUMNS and TYPES (represent SQLTABLES, SQLCOLUMNS and SQLTYPES commands defined in QlikView syntax) are used for fetching the data source metadata.
	 Request parameters:
	 [0] - statement to execute:SELECT, TABLES, COLUMNS and TYPES
	 [1] - data pipe name.
	 [2] - a list of semicolon separated statement specific parameters.
	     E.g. "TABLE_NAME=XYX" asks to return metadata for XYZ table. If no table name is specified for TABLES and COLUMNS commands, then metadata for all tables should be returned.
	     E.g."BLOB=2;" specifies that the second field should be retrieved as BLOB (SELECT statement).
	 Reply:
	 • QVX_OK, if the given statement is executed successfully.
	 Data pipe:
	 • QlikView starts reading data pipe only if QVX_OK is received.
	 • Data is formatted according to QVX file format (Chapter 2).
	 • TABLES result data structure:
	     Field [0] - TABLE_NAME.
	     Field [1] - TABLE_TYPE. Value: "TABLE".
	     Field [2] - CATALOG_NAME (optional).
	     Field [3] - SCHEMA_NAME (optional).
	     Field [4] - REMARKS (optional).

	• COLUMNS result data structure: Field [0] - TABLE_NAME. Field [1] - COLUMN_NAME. Field [2] - DATA_TYPE (optional). Field [3] - IS_NULLABLE (optional). Value the way it will be represented to the user. Field [4] - REMARKS (optional). Field [5] - IS_BLOB (optional). Values"true" or "false" (default).. • TYPES result data structure is not predefined.
QVX_EDIT_CONNECT	Create a connect statement. In future releases this request may also be used for requesting modification of existing connect statements. The request is sent on Connect button click. Also it is sent on Select and Custom buttons click, if QlikView does not have connect information for the data source. **Request parameters:** **[0]** - empty parameter for creating a new connect statement and connect statement for editing the existing connect statement. **Request options:** Include connection specific settings. **Reply:** • QVX_OK, if the connect statement was created or modified successfully. Connection to the data source does not have to be established. • OutputValues[0] - created or modified connect statement.
QVX_EDIT_SELECT	Create a select statement. In future releases this request may also be used for requesting modification of existing select statements. The request is sent on Custom button click. **Request parameters:** **[0]** - empty parameter for creating a new select statement and connect statement for editing the existing select statement. **[1]** - connect statement. Describes the last datasource selected by the user for connection. **Reply:** • QVX_OK, if the select statement was created or modified successfully. • OutputValues[0] - created or modified select statement.
QVX_GENERIC_COMMAND	Execute commands specified by the 1st parameter and return the result via command pipe. Parameters are command specific. The commands are sent on opening Script Editor and on Select button click. **GetCustomCaption** - if the Connector supports the custom button, then return the caption for this button. **Request parameters:** **[0]** - "GetCustomCaption". **Reply:** • QVX_OK, if command executed successfully. • OutputValues[0] - custom button caption.

	IsConnected - true, if the Connector is connected to the currently set data source. **Request parameters:** **[0]** - "IsConnected". **Reply:** • QVX_OK, if command executed successfully. • OutputValues[0] - "true"/"false". **DisableQlikViewSelectButton** - true, if QlikView Select Wizard, i.e. Select button, should be disabled. **Request parameters:** **[0]** - "DisableQlikViewSelectButton". **Reply:** • QVX_OK, if command executed successfully. • OutputValues[0] - "true"/"false". **HaveStarField** - true, if the datasource understands the "*" syntax in select statements, i.e. "SELECT * FROM..." is valid. **Request parameters:** **[0]** - "HaveStarField". **Reply:** • QVX_OK, if command executed successfully. • OutputValues[0] - "true"/"false".
QVX_DISCONNECT	Disconnect from the currently connected data source. **Reply:** • QVX_OK, if successfully disconnected from the data source.
QVX_TERMINATE	Clean-up as the process will be terminated. If Qlikview receives a reply, then it closes the command pipe and terminates the Connector process. Otherwise make sure to terminate. **Reply:** • QVX_OK, if clean-up is finished and QlikView can terminate the Connector process.
QVX_PROGRESS	Currently not used. **Reply:** • QVX_UNSUPPORTED_COMMAND, to enable forward compatibility.
QVX_ABORT	Currently not used. **Reply:** • QVX_UNSUPPORTED_COMMAND, to enable forward compatibility.

19

QvxConnectOptions Type

Child Element	Description
Provider	Connector namethat is equivalent to the Connector file name.
LoginTimeoutSec	The number of seconds to wait for a login request to complete before returning to the application.
ConnectTimeoutSec	The number of seconds to wait for any request on the connection to complete before returning to the application.
AutoCommit	If true, automatically commit SQL statement immediately after it is executed.
ReadOnly	If true, open data source as read-only.
AllowPrompt	If true, then user prompts are allowed.

3.4.1.2   Connect Request Example

```
<QvxRequest>
        <Command>QVX_CONNECT</Command>
        <Parameters>
                <String>ASHOST=XX.XX.X.XX;SYSNR=X;CLIENT=X;UserId=X;Password=X
                 </String>
        </Parameters>
        <Options>
                <Provider> MyCustomConnect.exe </Provider>
                <LoginTimeoutSec>-1</LoginTimeoutSec>
                <ConnectTimeoutSec>-1</ConnectTimeoutSec>
                <AutoCommit>true</AutoCommit>
                <ReadOnly>true</ReadOnly>
                <AllowPrompt>true</AllowPrompt>
        </Options>
</QvxRequest>
```

3.4.1.3   Command Request Example

```
<QvxRequest>
        <Command>QVX_EXECUTE</Command>
         <Parameters>
                <String>SELECT ProductID, Name, ListPrice
                FROM AdventureWorks.Production.Product</String>
                <String>\\.\pipe\765C2F31.pip</String>
        </Parameters>
</QvxRequest>
```

3.4.2   Connector Reply

As a response to QlikView's request the Connector sends a message formatted according to QvxReply XML schema. The data stream if formatted as follow:

- 4-byte that specify the length of the following string.
- Zero terminated string that is formatted according to QvxReply XML schema.

3.4.2.1   QvxReply XML Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
 <xs:simpleType name="QvxResult">
  <xs:restriction base="xs:string">
   <xs:enumeration value="QVX_OK"/>
   <xs:enumeration value="QVX_UNKNOWN_COMMAND"/>
   <xs:enumeration value="QVX_UNSUPPORTED_COMMAND"/>
   <xs:enumeration value="QVX_UNEXPECTED_COMMAND"/>
   <xs:enumeration value="QVX_SYNTAX_ERROR"/>
   <xs:enumeration value="QVX_CONNECT_ERROR"/>
   <xs:enumeration value="QVX_TABLE_NOT_FOUND"/>
   <xs:enumeration value="QVX_FIELD_NOT_FOUND"/>
   <xs:enumeration value="QVX_PIPE_ERROR"/>
   <xs:enumeration value="QVX_UNEXPECTED_END_OF_DATA"/>
   <xs:enumeration value="QVX_UNKNOWN_ERROR"/>
   <xs:enumeration value="QVX_CANCEL "/>
  </xs:restriction>
 </xs:simpleType>

 <xs:element name="QvxReply">
  <xs:complexType>
   <xs:all>
    <xs:element name="Result" type="QvxResult" />
    <xs:element name="OutputValues">
     <xs:complexType>
      <xs:sequence>
       <xs:element name="String" type="xs:string" minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
     </xs:complexType>
    </xs:element>
    <xs:element name="ErrorMessage" type="xs:string" />
   </xs:all>
  </xs:complexType>
 </xs:element>
</xs:schema>
```

QvxReply Element

Child Element	Description
Result	Error message type. See **QvxResult**.
OutputValues	String array of result values. See **QVXCommandType** table for how it is used.
ErrorMessage	Error explanation when Result <> QVX_OK.

QvxResult Type

Type Value	Description
QVX_OK	Command executed successfuly.
QVX_UNKNOWN_COMMAND	Command is not recognized.
QVX_UNSUPPORTED_COMMAND	Command is recognized but not implemented.
QVX_UNEXPECTED_COMMAND	Execution of the command requested not in the right circumstances.
QVX_CONNECT_ERROR	Data source connection is not established or is lost.
QVX_SYNTAX_ERROR	SQL statement contains a syntax error.
QVX_TABLE_NOT_FOUND	Table specified in SQL statement is not found.
QVX_FIELD_NOT_FOUND	Field specified in SQL statement is not found.
QVX_PIPE_ERROR	Failed to write data to data pipe.
QVX_UNEXPECTED_END_OF_DATA	Failed to fetch data from data source.
QVX_UNKNOWN_ERROR	Other error occured.
QVX_CANCEL	Command execution cancelled by the user.

3.4.2.2 QvxReply Example

```
<?xml version="1.0"?>
<QvxReply>
          <Result>QVX_OK</Result>
          <ErrorMessage/>
</QvxReply>
```

3.5 Data Pipe

Data pipe is used by the Connector to stream result data fetched after executing QVX_EXECUTE request command. The result data is formatted according to the QVX file format (Chapter 2).

For every QVX_EXECUTE request QlikView creates a new data pipe and uses parameter [1] to send its name to the Connector (see QVX_EXECUTE specification).

QlikView starts reading a data pipe after it receives a reply with confirmation about successful statement execution, i.e. QVX_OK. It waits for the incoming data as long as the Connector keeps the data pipe open or file separator byte is not found (if data is formatted as

QvxTableHeader.UsesSeparatorByte = true). If incorrectly formatted data is encountered, the remaining data is ignored.

3.6 Using Connectors

3.6.1 Connector Location
QlikView looks for Connectors:

1. At the same location as the current QV.exe
2. "Program Files\Common Files\QlikTech\Custom Data" folder and subfolders
3. "Program Files (x86)\Common Files\QlikTech\Custom Data" folder and subfolders, if it is 64-bit OS

On 64-bit OS 64 and 32-bit QlikView loads both 64 and 32-bit Connectors. First QlikView looks for 64-bit Connectors in the current QV.exe location and then in "Program Files\Common Files\QlikTech\Custom Data". Further QlikView looks for 32-bit Connectors in the current QV.exe location and then in "Program Files (x86)\Common Files\QlikTech\Custom Data".

If QlikView finds both exe and dll Connector files that have the same name then the exe files are prioritized and included in the connector list and will be used to contact the custom data source. Similarly, if a few Connectors have the same file name and extension, then only the first Connector found is included in the connector list.

3.6.2 Connect Statement Syntax

Below connect statement syntax for Connectors is stated:

CUSTOM CONNECT TO "Provider=_custom_connect_app_name_**;**[**UserId | XUserId=**_userid_**;**] [**Password | XPassword=**_password_**;**][_custom_connect_app_specific_param_]**" ;**

Connect Parameters	Description
custom_connect_app_name	Connector file name.
userid	Unscrambled or scrambled user id
password	Unscrambled or scrambled password
custom_connect_app_specific_param	Connector specific parameters

For instance:

CUSTOM CONNECT TO
"Provider=MyCustomConnect.exe;ASHOST=XX.XX.X.XX;SYSNR=X;CLIENT=X;UserId=X;Password=X ";

3.6.3 Parameters Handled by QlikView

- String casing can be handled by QlikView. For instance, the following statements will convert data to upper or lower case respectively:

    Force Case Upper;
    Force Case Lower;

- Null symbol represetation can be handled by QlikView. For instance, the following statement will replace all null values with '<NULL>'.

    Set NullDisplay = '<NULL>';

- QlikView will rename fields received from the Connector if they begin with '@'. There will only be one additional @ even if the name starts with more of them. This convention has been used for years with other data sources inside QlikView. It enables an easy way to iterate Tables and Fields, making virtual names like: @1 - @n. This is handled entirely inside QlikView. For instance, an original table or field named @1 will be renamed as @@1 inside QlikView.