



Product: Database Accelerator (DBXL) v2.5

Title: QueryDB User Manual

This is the User Manual for the QueryDB web service included in DBXL v2.5. The document explains the operations available, as well as the possible filtering and sorting operations.

Below you will find an outline of this document's contents.

Sample data	2
Configuration	3
Accessing a remote database	3
Operations	4
GetColumnsXMLStrQuery	4
GetDropdownEntries	4
GetColumnsXMLQuery	5
DescribeDatabaseSchema	5
XML Query	6
maxrows	7
Filtering	7
Sorting	14
Hints and Tips	15
Installing QueryBuilder	15
Invalid XML	15
Using Current() in Repeating tables	16
Using CDATA for special characters	17



Using countOnly	18
Support	18

SAMPLE DATA

Note that all of the examples in this document will derive their data from the SQL table in the screenshot below.

ID	ProjectName	Client	Estimated_length_months	Actual_length_months
1	Project1	Client1	12	12
2	Project2	Client1	6	18
3	Project3	Client2	18	24
4	Project4	Client3	12	12
5	Project5	Client2	6	6
6	Project6	Client4	12	18
7	Project7	Client5	18	12
8	Project8	Client6	6	6
9	Project9	Client6	18	18
10	Project10	Client7	12	18
11	Project11	NULL	NULL	NULL

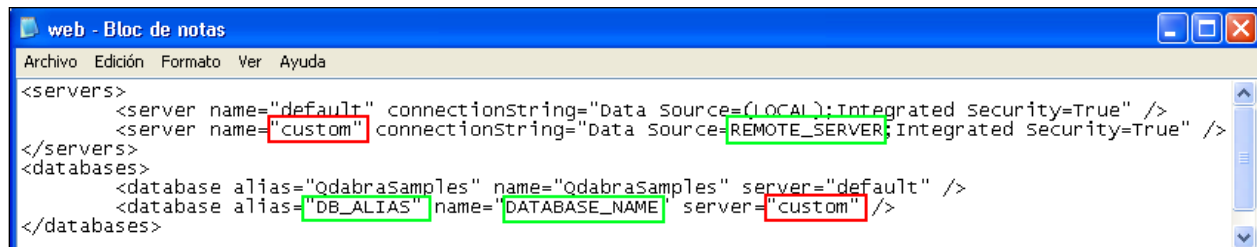


CONFIGURATION

Before you can begin using QueryDB to query a SQL database, the database being queried needs to be manually added to the *configuration/qdabra.dbxl/enumdb/databases* node in web.config. The SQL server indicated during installation (where QdabraDBXL and QdabraSamples are located) is already configured to use QueryDB, but any additional databases must be added individually, like this: `<database alias="DB_Alias" name="SampleDB" server="default" />`.

ACCESSING A REMOTE DATABASE

In order to allow QueryDB to access a database that lies on a different server, you must add the server and database aliases to the web.config file. Look for instructions within the *configuration/qdabra.dbxl/enumdb/servers* and *configuration/qdabra.dbxl/enumdb/databases* nodes. Note that the connection string in the `<server>` element in web.config should not specify a specific database, only the database server. The database alias should be separately added in the database aliases node as stated above.



```

web - Bloc de notas
Archivo Edición Formato Ver Ayuda
<servers>
  <server name="default" connectionString="Data Source=(LOCAL);Integrated Security=True" />
  <server name="custom" connectionString="Data Source=REMOTE_SERVER;Integrated Security=True" />
</servers>
<databases>
  <database alias="QdabraSamples" name="QdabraSamples" server="default" />
  <database alias="DB_ALIAS" name="DATABASE_NAME" server="custom" />
</databases>

```

The database alias must be placed within pound signs (#, also called number sign or hash mark) when using QueryBuilder to build the queryxml and when adding the query to the InfoPath data connection wizard. Note that you must also provide the requester machine read access in the remote database.



OPERATIONS

There are four methods within QueryDB, each of them allowing the user to query the database in different ways.

GETCOLUMNSXMLSTRQUERY

GetColumnsXMLStrQuery returns an array of all columns of the specified rows in a table. The user must provide an xml query, generated by Qdabra's Query Builder, which details database, table, retrieved columns, sorting and filtering.

Depending on the query generated by Query Builder, the user can enable rules within the form in order to use runtime data to affect the query. A simple scenario is also possible, where Query Builder generates a static xml query.

GETDROPDOWNENTRIES

GetDropdownEntries returns a list of values and optionally display values to easily bind with a dropdown control. The user must provide values for database, table and valueColumn. By default, the values will be sorted according to valueColumn in ascending order, or by displayColumn if provided. Otherwise, the user can specify a column to sort in the sortColumn parameter and also indicate the sortOrder (ascending or descending) as well as provide a filterXml. Here is the full information regarding the parameters:

database	required
table	required
valueColumn	column name
displayColumn	optional - column to populate display field in results
sortColumn	optional - can be any column in the table
sortOrder	optional – ["A" or "D"], ["ASC" or "DESC"] for ascending or descending
filterXml	optional – a <filter> xml, same as <filter> node as used in other methods

To illustrate the use of this method, we will query our sample data to list the ProjectName column, sorted by Estimated_length_months in descending order. The parameters required for this operation are seen in Figure 2 and result of that query is seen below, in Figure 3. Note that if no sortColumn and no sortOrder are provided, the method will automatically sort it by valueColumn in ascending order.



Parameter	Type	Value
tns:database	string	SampleDB
tns:table	string	SampleTable
tns:valueColumn	string	Estimated_length_months
tns:displayColumn	string	ProjectName
tns:sortColumn	string	Estimated_length_months
tns:sortOrder	string	A
tns:filterXml	string	

Figure 2

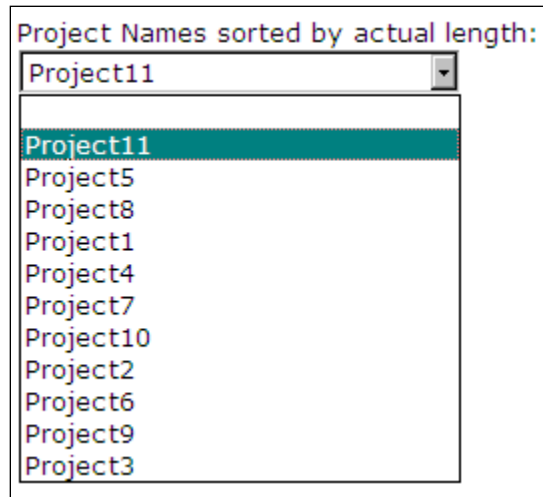


Figure 3

The filtering parameter, filterXML, will be discussed later in this document. There is no need to use a sorting operation within the filterXml since this method already has built in parameters for sorting in the Data Connection Wizard.

GETCOLUMNSXMLQUERY

GetColumnsXMLQuery is meant for use in code exclusively and behaves in the same way as GetColumnsXMLStrQuery. This method accepts unescaped XML for the queryXml parameter.

DESCRIBEDATABASESCHEMA

DescribeDatabaseSchema scans the schema in the specified connection string and returns an XML document describing it. The dataSource parameter should be a connection string as passed to the SqlConnection constructor in the .NET framework.



XML QUERY

When using `GetColumnsXMLStrQuery`, the user can perform fine-grained filtering by using a XML Query, which will have the following format:

```
<query database='' table='' maxrows =''>

  <columns>

    <column name=''>

      ...

    </column>

  </columns>

  <filter>

    ...

  </filter>

  <sort>

    <column name='' order=''/>

  </sort>

</query>
```

The simplest query that can be used is one that will pull down an entire table from a given database. The format for this query is `<query database='<database name>' table='<table name>'></query>` and the result is shown in Figure 4.



ID	Project Name	Client	Estimated Length Months	Actual Length Months
1	Project1	Client1	12	12
2	Project2	Client1	6	18
3	Project3	Client2	18	24
4	Project4	Client3	12	12
5	Project5	Client2	6	6
6	Project6	Client4	12	18
7	Project7	Client5	18	12
8	Project8	Client6	6	6
9	Project9	Client6	18	18
10	Project10	Client7	12	18
11	Project11			

Figure 4

While more fine-grained xml queries can be generated manually, Qdabra provides the Query Builder tool, which allows the user to easily generate a desired xml query. The Query Builder tool will be discussed in an upcoming section.

MAXROWS

The maxrows parameter allows the user to limit the number of rows that are returned by the query. This is especially useful if a user desires to query a large amount of data. The format used for this is `<query maxrows='x'>`, where x is the number of rows to be returned.

FILTERING

There are various types of comparison operations available for the `<filter>` node of queryXml.

a. Comparison operators: lt, le, eq, ne, ge, gt

The first set of comparison operators- namely `<lt>`, `<le>`, `<eq>`, `<ne>`, `<ge>` and `<gt>` - all use similar formats. The comparisons can take place between a column and a value, or between a column and another column. The examples below will illustrate some of the uses of these operators. Each sample will be accompanied by a table showing the result.

```
<lt>
<!-- lt stands for "less than". In the example
below we compare a column and a value. -->
<column name="Actual_length_months"/>
```



```
<value type="int">12</value>
</lt>
```

ID	Project Name	Client	Estimated Length Months	Actual Length Months
5	Project5	Client2	6	6
8	Project8	Client6	6	6

```
<le>
  <!-- le stands for "less than or equal". In the
  example below we compare a column and a value. -->
  <column name="Actual_length_months"/>
  <value type="int">18</value>
</le>
```

ID	Project Name	Client	Estimated Length Months	Actual Length Months
1	Project1	Client1	12	12
2	Project2	Client1	6	18
4	Project4	Client3	12	12
5	Project5	Client2	6	6
6	Project6	Client4	12	18
7	Project7	Client5	18	12
8	Project8	Client6	6	6
9	Project9	Client6	18	18
10	Project10	Client7	12	18

```
<eq>
  <!-- eq stands for "equal". In the example below
  we compare two columns. -->
  <column name="Actual_length_months"/>
  <column name="Estimated_length_months"/>
</eq>
```



ID	Project Name	Client	Estimated Length Months	Actual Length Months
1	Project1	Client1	12	12
4	Project4	Client3	12	12
5	Project5	Client2	6	6
8	Project8	Client6	6	6
9	Project9	Client6	18	18

```

<ne>
  <!-- ne stands for "not equal". In the example
  below we compare a column and an alphanumeric value. -->
  <column name="Client"/>
  <value type="varchar">Client1</value>
</ne>

```

ID	Project Name	Client	Estimated Length Months	Actual Length Months
3	Project3	Client2	18	24
4	Project4	Client3	12	12
5	Project5	Client2	6	6
6	Project6	Client4	12	18
7	Project7	Client5	18	12
8	Project8	Client6	6	6
9	Project9	Client6	18	18
10	Project10	Client7	12	18

```

<ge>
  <!-- ge stands for "greater than or equal". In
  the example below we compare a column and a value. -->
  <column name="Actual_length_months"/>
  <value type="int">12</value>
</ge>

```



ID	Project Name	Client	Estimated Length Months	Actual Length Months
1	Project1	Client1	12	12
2	Project2	Client1	6	18
3	Project3	Client2	18	24
4	Project4	Client3	12	12
6	Project6	Client4	12	18
7	Project7	Client5	18	12
9	Project9	Client6	18	18
10	Project10	Client7	12	18

```

<gt>
  <!-- gt stands for "greater than". In the example
  below we compare a column and a value. -->
  <column name="Actual_length_months"/>
  <value type="int">18</value>
</gt>

```

ID	Project Name	Client	Estimated Length Months	Actual Length Months
3	Project3	Client2	18	24

b. Comparison operator: in

The use of `<in>` is slightly different from the above set of comparison operators. This operator requires the user to specify a column and at least one value. It then selects rows that contain the value(s) listed in the specified column.

The following example illustrates the XML query when using `<in>`:

```

<in>
  <column name="Client"/>
  <values>
    <value>Client1</value>
    <value>Client2</value>
    <value>Client3</value>
  </values>
</in>

```



ID	Project Name	Client	Estimated Length Months	Actual Length Months
1	Project1	Client1	12	12
2	Project2	Client1	6	18
3	Project3	Client2	18	24
4	Project4	Client3	12	12
5	Project5	Client2	6	6

c. NULL values

One useful and common filtering operation involves the search for NULL values in your database. In order to do this, we will need to use `<ne>` and `<eq>` operators.

NULL values can be excluded, as shown below:

```
<ne>
  <column name="Client"/>
  <value nil="true"/>
</ne>
```

ID	Project Name	Client	Estimated Length Months	Actual Length Months
1	Project1	Client1	12	12
2	Project2	Client1	6	18
3	Project3	Client2	18	24
4	Project4	Client3	12	12
5	Project5	Client2	6	6
6	Project6	Client4	12	18
7	Project7	Client5	18	12
8	Project8	Client6	6	6
9	Project9	Client6	18	18
10	Project10	Client7	12	18

NULL values can also be included:

```
<eq>
  <column name="Client"/>
  <value nil="true"/>
```



</eq>

ID	Project Name	Client	Estimated Length Months	Actual Length Months
11	Project11			

d. Using LIKE and wildcards

The <like> filter can search for patterns in the data columns by using wildcard characters:

```
<like>
  <column name="ColumnName"/>
  <value>Pattern</value>
</like>
```

'Pattern' is the specific string of characters to search for in the column "ColumnName" and can include the following four wildcard characters:

- % will search for any string of characters. For example, the pattern %1 will yield the following result:

ID	Project Name	Client	Estimated Length Months	Actual Length Months
1	Project1	Client1	12	12
11	Project11			

- A pattern consisting only of % will yield all results in the queried table.
- An underscore (_) represents any single character. The example below shows the result obtained if we filter using 'Project_'.



ID	Project Name	Client	Estimated Length Months	Actual Length Months
1	Project1	Client1	12	12
2	Project2	Client1	6	18
3	Project3	Client2	18	24
4	Project4	Client3	12	12
5	Project5	Client2	6	6
6	Project6	Client4	12	18
7	Project7	Client5	18	12
8	Project8	Client6	6	6
9	Project9	Client6	18	18

- The square brackets can denote any single character within the specified range. For example, we can search based on a set; [1-3] would be equivalent to [123]. When the pattern is %[1-2] we will obtain all *Client* entries that start with the numbers 1 or 2.

ID	Project Name	Client	Estimated Length Months	Actual Length Months
1	Project1	Client1	12	12
2	Project2	Client1	6	18
3	Project3	Client2	18	24
5	Project5	Client2	6	6

- The square brackets in combination with the carrot (^) will allow the user to search for any single character not within the specified range. To illustrate this, we must implement the opposite filter (<value>%[^1-2]</value>) to the previous example above. The result of the filter is shown below.



ID	Project Name	Client	Estimated Length Months	Actual Length Months
4	Project4	Client3	12	12
6	Project6	Client4	12	18
7	Project7	Client5	18	12
8	Project8	Client6	6	6
9	Project9	Client6	18	18
10	Project10	Client7	12	18

e. Using CONTAINS

The `<contains>` filter allows you to perform keyword searches in your database, in the same way you should use it in a SQL statement (see <http://msdn2.microsoft.com/en-us/library/ms187787.aspx>). You can only use `<contains>` on columns that are full-text indexed. Full text searches allow you to also use the NEAR operation, as well as searching for phrases enclosed between double quotes, and wildcards (*) for prefix searches.

Finally, note that performing a `<contains>` filter within OR/AND operations (`<or>`/`<and>`) produces better performance than performing the same operation across two filters. For example, we will obtain the same result with either of the filtering operations below, but the first will give you better performance.

Option 1:

```
<or>
  <contains>
    <column name="Client"/>
    <value>Client1</value>
  </contains>
  <contains>
    <column name="Client"/>
    <value>Client2</value>
  </contains>
</or>
```

Option 2:

```
<contains>
  <column name="Client"/>
  <value>Client1 OR Client2</value>
</contains>
```

SORTING

GetDropdownEntries allows you to sort, as was explained earlier in this document.



<http://www.qdabra.com>

Last updated on 4/1/2011 1:35 PM

Copyright © 2006-2011 Autonomy Systems, LLC. All rights reserved.

However, GetColumnsXMLQuery and GetColumnsXMLStrQuery also allow users to sort the results. When using multiple entries, the results will be sorted by each of the columns listed, in the order listed. Sorting by:

```
<sort>
  <column name="Actual_length_months" order="ASC"/>
  <column name="Estimated_length_months" order="ASC"/>
</sort>
```

Yields:

ID	Project Name	Client	Estimated Length Months	Actual Length Months
11	Project11			
5	Project5	Client2	6	6
8	Project8	Client6	6	6
4	Project4	Client3	12	12
1	Project1	Client1	12	12
7	Project7	Client5	18	12
2	Project2	Client1	6	18
6	Project6	Client4	12	18
10	Project10	Client7	12	18
9	Project9	Client6	18	18
3	Project3	Client2	18	24

HINTS AND TIPS

INSTALLING QUERYBUILDER

Sometimes installing QueryBuilder will generate the following error: ***"Cannot create InfoPath.ExternalApplication object. Error code = 0x80080005."*** This error is usually avoided if you select the "Just Me" option when installing QueryBuilder (instead of "Everyone").

INVALID XML

When working with QuerySharePoint or QueryDB, you might encounter an error that says: "Data at the root level is invalid. Line 1, position 1". This error indicates a problem with the <query> node. You could try using QueryBuilder to re-generate the value for <query>.



For example, in a dynamic query, you should check that the <query> that begins with concat() is used in rules and not in the data connection wizard.

USING CURRENT() IN REPEATING TABLES

You might encounter a problem when using QuerySharePoint or QueryDB to dynamically retrieve values into a repeating table. To illustrate this scenario, imagine that a user selects a **Product Name** from a dropdown, and the goal is to set the **Code** and the rest of the repeating table fields with the corresponding values for the selected product. A second QuerySharePoint or QueryDB connection is created to dynamically query. The problem? The query (as generated by QueryBuilder) will not get updated when inserting new rows. The query always used the **Product Name** from the first row and returns the same value, as seen below.

Product Name	Code	Category	UM	Price
Chocolate Milk	104	Dairy	oz	18.00
Nescafe	104	Dairy	oz	18.00
Total				\$36.00

To identify the issue, we first need to look at the <query> that was created using Query Builder:

```
concat("<query><columns><column name='Product_x0020_Code'/><column name='Category'/><column name='Unit_x0020_Of_x0020_Measure'/><column name='Price'/></columns><filter><eq><column name='Product_x0020_Name'/><value>", ProductName, "</value></eq></filter></query>")"
```

To fix this, simply replace **ProductName** with current():

```
concat("<query><columns><column name='Product_x0020_Code'/><column name='Category'/><column name='Unit_x0020_Of_x0020_Measure'/><column name='Price'/></columns><filter><eq><column name='Product_x0020_Name'/><value>", current(), "</value></eq></filter></query>")"
```

Now, when the user inserts a new row and selects a different **Product Name**, we get the correct data:



Inventory Report

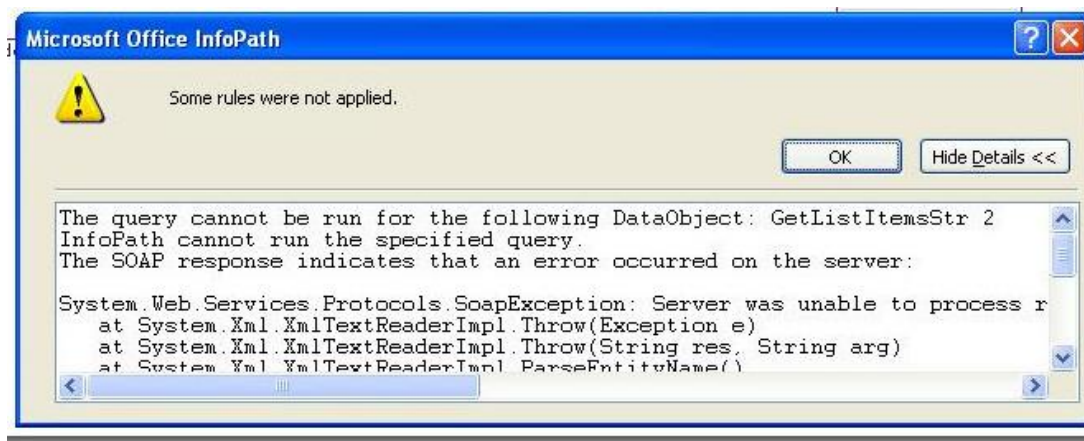
Branch Name: Date:

Branch Code:

Product Name	Code	Category	UM	Price
Chocolate Milk v	104	Dairy	oz	18.00
Nescafe v	101	Breakfast Foods	oz	12.00
Total				\$30.00

USING CDATA FOR SPECIAL CHARACTERS

When SharePoint or database queries contain special characters, you might encounter an error such as the one seen in the screenshot below:



This means that our database or SharePoint data contains special characters, and we must edit the value of <query> to use CDATA.

```
concat("<query><columns><column name='Product_x0020_Code'/><column name='Category'/><column name='Unit_x0020_Of_x0020_Measure'/><column name='Price'/></columns><filter><eq><column name='Product_x0020_Name'/><value>", current(), "</value></eq></filter></query>")
```

What we need to do is just add a CDATA inside the query <value> </value>:

```
concat("<query><columns><column name='Product_x0020_Code'/><column name='Category'/><column name='Unit_x0020_Of_x0020_Measure'/><column name='Price'/></columns><filter><eq><column name='Product_x0020_Name'/><value><![CDATA["", current(), "]]></value></eq></filter></query>")
```



Note that CDATA should always start with the sequence: `<![CDATA[` and ends with `]]>`.

USING COUNTONLY

The countOnly parameter allows you to return only the number of items, instead of returning the entire data set. To set the parameter to true, use a query like this:

```
<query countOnly='true' database="MyDB" table="MyTable"></query>
```

If the countOnly parameter is equal to anything other than 'true' (or if it is not present), then it is considered to be equal to false, and all records are returned.

SUPPORT

If you have questions about the information in this document, please contact us for assistance.

Licensed customers contact us via Support@Qdabra.com. Alternatively, please use the InfoPathDev.com [Qdabra](http://Qdabra.com) [Product support forums](http://ProductSupportForums.com) to request help.

