FileMaker Powershell class ‘FMQ’ Documentation

The Basics

Include the Class:

```
. “\\path\to\fmq.ps1”
```

Instantiate the Class and Choose DB, and Layout:

One way:

```
$fm = New-Object fmq([YourDB],[YourLayout])
```

Another way:

```
$fm = [fmq]::New([YourDB],[YourLayout])
```

Add Query Parameters:

```
$fm.AddParam($FMField,$value)
```

Repeat this line as many times as you have field/value pairs to search for, to edit, or to add (new).
(edit will require the field ‘-recid’ with its value, ‘delete’ will require ONLY this parameter)
Note: The ‘-recid’ field will be included in every result array (except ‘delete’) regardless of having placed it on the layout. The field name will be “RecID”

Send Request while choosing Query Type:

```
$fm.sendRequest($type)
```

Available Types:
find – regular search
findall – Find all records (no Query Parameters passed with this type)
findany – Find a single random record (no Query Parameters passed with this type)
findquery – Compound Find (see examples below)
new – Create a new Record
edit – Edit an existing record (-recid parameter required)
delete – Delete an existing record (-recid parameter required)
view - get value lists **or** get list of fields for a given layout (no Query Parameters passed with this type)

Examples

Basic Find Example: Find all Records with First name beginning with Jim and with age over 30

Note: the default search type in FileMaker is ‘begins with’, to be more explicit, use the standard FileMaker Pro operators. For example, if you used ‘=jim’ below, you would get only records where the name is jim and jim only. Additionally, this class has the added functionality of using “<>” to mean “not equal to”. So ‘<>jim’ would be “not equal to jim”

```
. \\path\to\fmq.ps1
$fm = New-Object fmq (’MyDatabase’,’MyLayout’)
$fm.AddParam(‘FirstName’,’jim’)
$fm.AddParam(‘Age’,’>30’)
$result = $fm.sendRequest(‘find’)
```

$result will be an array of Hashtables for each record found:

```
$result =>

        [0] => [fieldname1] => fieldValue1

               [fieldname2] => fieldValue2

               [fieldname3] =>    …


        [1] => [fieldname1] => fieldValue1

               [fieldname2] => fieldValue2

               [fieldname3] =>    …
```

…repeated for as many results found and as many fields that were on the given layout

Note: the Record ID will also be among these results for each record as:

```
                ['RecID'] => 1234
```

Regardless of whether or not this layout contains such a field

When using the actions: find, findall, edit, new, findany, and findquery, the resulting array will always be in this format.

This data can be displayed a number of ways, but here is one example:

```
If($result.count) { #if the $result array contains records

    foreach($data in $result) {

        Write-Host $data[‘FieldNameOne’] $data[‘FieldNameTwo’]

    }

} else {

    Write-Host “Nothing Found”

}
```

Basic Edit Example: Change an Age and First Name in a record given the RecID (required)

```
. \\path\to\fmq.ps1
$fm = New-Object fmq (’MyDatabase’,’MyLayout’)
$fm.AddParam(‘-recid’,1234)
$fm.AddParam(‘Age’,29)
$fm.AddParam(‘FirstName’,’Jimmy’)
$result = $fm.sendRequest(‘edit’); #(note the action change to ‘edit’)
```

This will change the ‘Age’ field to 29 and the ‘FirstName’ field to ‘Jimmy’ in the record with the ‘–recid’ of 1234

As previously mentioned, the $result array will be in the above format and will contain the changes you made to this record.

Note: A ‘delete’ action would look identical to the above code, with the exception that you would not include any other query parameters except for ‘-recid’ (required), and the query action would be changed from ‘edit’ to ‘delete’. If we removed the other query parameters above and changed the Query action to ‘delete’, it would have deleted the above record from the database as long as the ‘-recid’ field was provided and was correct.

Basic New Record Example: Add a Record and set the fields FirstName and Age

```
. \\path\to\fmq.ps1
$fm = New-Object fmq (’MyDatabase’,’MyLayout’)
$fm.fmCred(‘myUsername’,’myPassword’)
```

```
$fm.AddParam('Age',31)
$fm.AddParam('FirstName,'Bob')
$result = $fm.sendRequest('new'); #(note the action change to 'new')
```

This will add a record to the database and set the fields 'Age' to 31 and the FirstName field to 'Bob'

As previously mentioned, the $result array will be in the standard format and will contain the new record's fields and data

Other Find Types:

There are 3 other find types, 2 of them, findall and findany, are in the same format as above, but they do not take any parameters. (no AddParam()'s)

The action 'findall' finds all records in the database (see maxRecords() for control of how many of those records will actually be returned).

The action 'findany' will find any single random record.

Performing a compound find using the 'findquery' action:

Query: find all records with the name jimmy **or** the name bob, but **not** if the age in the record is less than 25:

First step is to determine how many find requests will be needed to perform the compound find, and define them using the '-query' keyword like so:

```
$fm.AddParam('-query','(q1);(q2);!(q3)')
```

In this case we need 3 separate find requests, the first 2 are going to be performed as logical 'or' searches, and since we want the last request to be an 'omit' type search, we use the '!' before the query. Notice queries are enclosed in parenthesis and delimited by semicolons.

There will then be 2 separate parameters needed for each request in this form (x representing the numbers after the q):

```
$fm.AddParam('-qx','field_to_query')
$fm.AddParam('-qx.value','corresponding_qx_value')
```

Given this, the full demo query above would look like this:

```
. \\path\to\fmq.ps1
$fm = New-Object fmq ('MyDatabase','MyLayout')
$fm.AddParam('-query','(q1);(q2);!(q3)')
$fm.AddParam('-q1,'FirstName')
$fm.AddParam('-q1.value','Jimmy')
$fm.AddParam('-q2','Firstname')
$fm.AddParam('-q2.value','Bob')
```

```
$fm.AddParam('-q3','Age')
$fm.AddParam('-q3.value','<25')
$result = $fm.sendRequest('findquery')
```

As previously mentioned, the $result array will be in the standard format and will contain the results of this compound find.

NOTE: To perform a logical **and** search along with other find requests, you would use (q1,q2);(q3).

Performing Scripts

If the database you are querying contains a FileMaker script that is “Custom Web Publishing” capable, you can run this script (and pass parameters) with any of the query actions except for ‘view’.

Example: run a script called ‘MyScript’ with a basic find:

```
. \\path\to\fmq.ps1
$fm = New-Object fmq ('MyDatabase','MyLayout')
$fm.fmCred('myUsername',myPassword')
$fm.AddParam('FirstName','jim')
$fm.AddScript('MyScript')
$result = $fm.sendRequest('find')
```

The above code will run ‘MyScript’ without passing any parameters.

To pass parameters use this method:

```
$fm.AddScript('MyScript','myParam1|myparam2|myparam3')
```

This will pass one parameter to your FileMaker script that you can then split out into the 3 separate parameters using the delimiter of your choice, a “|” in this case.

Note: using a [char]13 as delimiter will play right into the hands of the FileMaker GetValue script step and will make your separate parameters automatically available using that script step, see documentation on that step for further info.

Sorting Returned Results

You can sort your results by multiple fields in ascending or descending order. FileMaker XML Web publishing supports up to 9 sorts.

Example: find a set of records and sort first by FirstName asending, and then by DateCompleted descending:

Sorting is accomplished using the method:

$fm->AddSort('fieldname','sortorder') The second parameter can be left off for ascending sorts

The recommended values for the second parameter are either ‘ascend’ (optional) or ‘descend’ for the sake of code readability. However, all the class is looking for is that the first character of whatever is passed is either an ‘a’ or a ‘d’ (or nothing for ascending)

Sorts are processed in the order in which they logically are listed on the page, so the above example would be:

```
. \\path\to\fmq.ps1
$fm = New-Object fmq (’MyDatabase’,’MyLayout’)
$fm.fmCred(‘myUsername’,myPassword’);
$fm.AddSort(‘FirstName’) # notice there is no second parameter needed for ascending sorts
$fm.AddSort(‘DateCompleted’,’descend’)
$result = $fm.sendRequest(‘findall’)
```

Controlling Number of returned results using maxRecords()

To prevent an accidentally large search from happening and hanging up the script or taking forever, the default maximum number of results returned by the FMQ class is 50. This means if you perform a ‘findall’ on a database that you know has 2000 results, by default you will only get 50 of them. You can change this number to a number of your choosing, or choose to return all results using the maxRecords() method like so:

$fm.maxRecords(1000) (where 1000 could be any number you like)

Or to display all results regardless of record count:

$fm.maxRecords(‘all’)

Troubleshooting:

If you run into trouble getting results when you think there should be, the class includes an infoArr hashtable in every result:

$fm.infoArr

This Hashtable contains 3 things:

foundCount: the number for found records

displayCount: the number of records being displayed

And Error: this could contain a number of things like a FileMaker Pro error code such as 102 (field is missing) or maybe some text like ‘Could not connect to the server, check username and password’