fmxData

PHP Class for the FileMaker Server Data API

Documentation

Version 1.0

Author: Duane Weller (eXcelisys Inc.)

INDEX

WHAT IS FMXDATA?

FMXData is a PHP class that provides a library of simple functions for working with the FileMaker Server Data API. The PHP class takes care of creating the appropriate cURL calls to connect and run queries through the Data API.

WHY FMXDATA?

Intended to be a different flavor of PHP class, FMXData is structured a little bit differently then some of the other PHP classes for the Data API. As a single file it offers a simple way to add a library of PHP functions to easily work with the Data API. Functions such as connecting, disconnecting, and uploading container data are static functions and do not require you to initialize a class instance to use. Yet, you can initialize separate instances for each query. This makes it much more flexible to use in PHP framework, CMS, or even just on a simple PHP web page.

SUPPORTED DATA API FEATURES

fmxData supports all of the following Data API features.

1. Log in to a database session.

2. Log out of a database session.

3. Get Product Information

4. Get Database Names

5. Get Script Names

6. Get Layout Names

7. Get Layout Metadata

8. Create a Record

9. Edit a Record

10. Delete a Record

11. Get a Record

12. Get a Range of Records

13. Upload a File to a Container Field

14. Find FileMaker Records

15. Set FileMaker Globals

16. Set a Response Layout

17. Perform a Script

18. Perform Pre-request Script

19. Perform Presort Script

20. Limit which Portals to Return

21. Set Portal Limits and Offsets

22. Set the Record Limit and Offset

23. Add a Sort Field, Sort Direction, and Sort Order

THINGS NOT SUPPORTED

fmxData does not yet support the following FileMaker Data API features.

1) Log in to an external data source.

2) Log in to a database session using an OAuth identity provider.

3) Log in to a database session using a FileMaker ID account.

4) Duplicate a record

5) Logging or Debugging

6) Checks or traps for FileMaker erros

The log in features were not necessarily needed for what I plan to use fmxData for but may be added in some future version. Record duplication does not appear to be working as documented in the FileMaker Data API documentation. If at some point Claris updates the documentation, perhaps additional features and functions may be added to fmxData. To keep this class lightweight I have not implemented any kind of logging or debugging. Some of the other PHP classes available offer those features. FMXData does not check or test for errors returned by the FileMaker Server Data API. The entire response is retuned to the calling script so that you can manage the FileMaker errors in your own PHP.

PREPARING FILEMAKER SERVER

Go into the FileMaker Server Admin Console, under the connections area select the "FileMaker Data API" and enable it.

INSTALLING THE EXAMPLE FILES

1) Host the FMXData.fmp12 with FileMaker Server. Use the "Upload to Host" feature in FileMaker Pro Advanced (under "File -> Sharing -> Upload to Host") to upload the FMXData.fmp12 to the server.

Open the FMXData.fmp12 with the following credentials:

Username: admin
Password: admin

2) Enable the FileMaker Data API in the FileMaker Server Admin Console. Under the connections area select the "FileMaker Data API" and enable it.

3) Add the example.php, FMXData.php, and photo.jpg files to the FileMaker Server's Web Server root folder.

On Mac the web server's root folder will be in the FileMaker Server

Windows: [drive]:\Program Files\FileMaker\FileMaker Server\HTTPServer\conf

macOS for HTTP URLs: /Library/FileMaker Server/HTTPServer/htdocs

macOS for HTTPS URLs: /Library/FileMaker Server/HTTPServer/htdocs/httpsRoot

Accessing the Example Files from a Browser

Pay attention to which folder you add the files to. If you added the files to the htdocs folder then you'll get to the example file here:

`http://localhost/example.php`

If you added the files to the secure folder (httpsRoot), you'll need to go to this URL:

`https://localhost/example.php`

If you're seeing a permission error, you may have to start PHP. Open the terminal and type in.

`fmsadmin set cwpconfig enablephp=true`
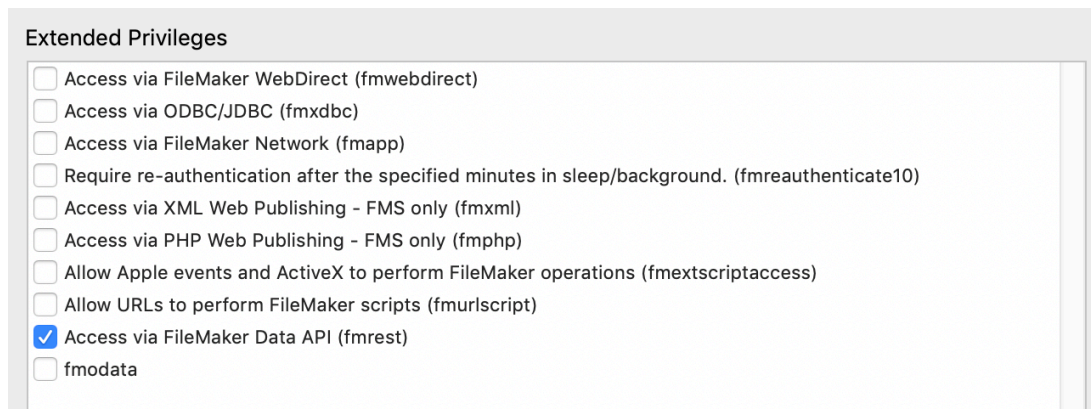
You'll need to enter your FileMaker Server admin console username and password, then restart the FileMaker Server for the changes to take effect.

PREPARING AND HOSTING YOUR FILE

NAMING - Try and keep your database, layout, script, and to some extent field names URL friendly. This means sticking to the standard alphanumeric characters. Avoid names with spaces and most special characters. Periods "." and dashes "-" are URL friendly but can't be used in FileMaker field names. The underscore "_" character is about the only special character that's universally acceptable.

SECURITY - I recommend you add a specific FileMaker Account for the Data API, however you can add the functionality to any FileMaker account by enabling the "Access via FileMaker Data API (fmrest)" extended privilege. This must be enabled before the account can connect using the FileMaker Data API.

ADDING THE LIBRARY TO YOUR PROJECT

To add the library to your project just require the FMXData file at the top of your PHP file.

Example Code:
```
require('FMXData.php');
```

CONNECTING TO FILEMAKER SERVER

Before you can start running finds and working with records you need to connect to the FileMaker database. Use the fmxConnect( ) function to create a session with FileMaker Server. Only one session may be open at a time. If you need to access more then one database or host, you can do it one at a time. Run the fmxDisconnect( ) function to close your fist session then run fmxConnect( ) to open a new session with the new database or host.

fmxConnect ($host, $database, $username, $password {, $sslVarify })

Parameters:
$host - The host server's domain or IP - can be localhost or 127.0.0.1 as well.
$database - The filename of the FileMaker file without the extension (.fmp12).
$username - The username of the FileMaker account.
$password - The password of the FileMaker account.
$sslVarify - (Optional) Can be set to "enable" or "disable". The default is enable.

A Note About SSL Verification
FileMaker Server only accepts Data API connections via https:// which means there's a SSL certificate providing encryption for the connection. cURL will verify that the certificate is valid when making the connection. Your connection will fail if you have the default server certificate or a self-signed certificate installed. To bypass the SSL verification set the value of $sslVarify to "disable".

Example Code:
```
$result =
FMXData::fmxConnect('localhost','FMXData','fmxdata','fmxdata','disable');
```

On success, returns a response array with the token and a message array with the status.

```
Array (
    [response] => Array ( [token] => 2a9f528858e9905cf5ea9fdf9281... )
    [messages] => Array ( [0] => Array ( [code] => 0 [message] => OK ) )
 )
```

The Token value is also added to the $GLOBALS variable. You can access it's value by including the global in your code:

Example Code:
```
global $fmx_token;
```

DISCONNECTING FROM FILEMAKER SERVER

I recommend that you disconnect from the session when you have completed all queries. You can choose to leave the FileMaker session open and save the token to the server's session or a cookie. However you'll need to manage the token and verify it hasn't expired with each page load. If necessary, you'll need to refresh the session and keep the token updated. Keep in mind that each open session will be listed in the FileMaker Server admin console on the Clients list until they expire. The maximum number of open sessions is 1000.

fmxDisconnect ( )

Example Code:
```
$result = FMXData::fmxDisconnect( );
```

On success, returns an empty response array and a message array with the status.

```
Array (
    [response] => Array ( )
    [messages] => Array ( [0] => Array ( [code] => 0 [message] => OK ) )
)
```

FILEMAKER SERVER FUNCTIONS

There are two functions that work directly with FileMaker Server. The fmxProductInfo( ) returns information about FileMaker Server and the fmxListDatabases( ) function returns a list of databases available based on your connection.

fmxProductInfo ( )

Example Code:
```
$result = FMXData::fmxProductInfo( );
```

On success, returns a response array with the product information and a message array with the status.

```
Array (
    [response] => Array ( [productInfo] => Array ( ... ) )
    [messages] => Array ( [0] => Array ( [code] => 0 [message] => OK ) )
 )
```

fmxListDatabases ( )

Example Code:
```
$result = FMXData::fmxListDatabases();
```

On success, returns a response array with the database list and a message array with the status.

```
Array (
    [response] => Array ( [databases] => Array ( ... ) )
    [messages] => Array ( [0] => Array ( [code] => 0 [message] => OK ) )
 )
```

DATABASE FUNCTIONS

There are three database functions to provide access to your database metadata like layout names, script names, and layout information. The fmxListScripts( ) function gets an array of the database scripts. The fmxListLayouts( ) function gets a list of the database layouts. The fmxLayoutMeta( ) function returns layout data including value lists. With the fmxLayoutMeta( ) function you can even pass the record ID to return related value lists.

fmxListScripts ( )

Example Code:
```
$result = FMXData::fmxListScripts();
```

On success, returns a response array with the script list and a message array with the status.

```
Array (
    [response] => Array ( [scripts] => Array ( ... ) )
    [messages] => Array ( [0] => Array ( [code] => 0 [message] => OK ) )
 )
```

fmxListLayouts ( )

Example Code:
```
$result = FMXData::fmxListLayouts();
```

On success, returns a response array with the layouts list and a message array with the status.

```
Array (
    [response] => Array ( [layouts] => Array ( ... ) )
    [messages] => Array ( [0] => Array ( [code] => 0 [message] => OK ) )
 )
```

fmxLayoutMeta ($layout {, $recordId })

Parameters:
$layout - The FileMaker Layout to query.
$recordId - (Optional) record id for related value lists.

Example Code:
```
$result = FMXData::fmxLayoutMeta( "CONTACTS", 15297 );
```

On success, returns a response array with multiple arrays for field MetaData, portal MetaData, value lists, etc., and a message array with the status.

```
Array (
    [response] => Array (
        [fieldMetaData] => Array ( ... )
        [portalMetaData] => Array (
            [email_portal] => Array ( ... )
            [phone_portal] => Array ( ... )
        )
    [messages] => Array ( [0] => Array ( [code] => 0 [message] => OK ) )
)
```

RECORD FUNCTIONS & OPTIONS

There are six FMXData functions for working with FileMaker records fmxCreateRecord( ), fmxEditRecord( ), fmxDeleteRecord( ), fmxGetRecord( ), fmxGetRange( ), and fmxFindRecords( ). All of the previous functions you could just call the function, there were no additional options you could add. However, the record functions you need to initialize a new instance of the class then you can add additional configuration options and execute the function. Your code might look like something like this:

```
$fmx = new FMXData( );
$fmx->addPostFieldArray($data);
$result = $fmx->fmxCreateRecord('CONTACT');
```

If you need additional options you just add them to the $fmx class object. So for example if I want to add a FileMaker script, I can add the "addScript( )" option.

```
$fmx = new FMXData( );
$fmx->addPostFieldArray($data);
$fmx->addScript('FMX_Script','param1');
$result = $fmx->fmxCreateRecord('CONTACT');
```

All of the compatible options will be listed with the function information. For the most part it does not matter the order you add your options. However, there is an exception to this when adding sort fields. If you don't specify the $sortOrder parameter the fields will be sorted in the order you add them.

fmxCreateRecord ($layout)

<u>Parameters:</u>
$layout - The FileMaker Layout to query.

<u>Example Code:</u>
```
$fieldArray['First Name'] = 'Duane';
$fieldArray['Last Name'] = 'Weller';
$fieldArray['Job Title'] = 'Lead Developer';
$fmx = new FMXData();
$fmx->addPostFieldArray($fieldArray);
$fmx->addPostField('Company','Excelisys Inc');
$fmx->addScript('FMX_Script','param1');
$result = $fmx->fmxCreateRecord('CONTACT');
```

On success, returns a response array with the new recordId and modId. If you included a script option the script results will also be included.

```
Array (
    [response] => Array (
        [scriptResult] => Script Run - Param: param1
        [scriptError] => 0
        [recordId] => 20303
        [modId] => 0
    )
    [messages] => Array ( [0] => Array ( [code] => 0 [message] => OK ) )
)
```

<u>COMPATIABLE OPTIONS</u>
<u>addPostFieldArray ($fieldArray)</u>
<u>addPostField ($fieldName, $fieldValue)</u>
<u>addScript ($scriptName {, $scriptParameter})</u>
<u>addPresortScript ($scriptName {, $scriptParameter})</u>
<u>addPrerequestScript ($scriptName {, $scriptParameter})</u>

fmxEditRecord ($layout, $recordId)

Parameters:
$layout - The FileMaker Layout to query.
$recordId - The FileMaker record ID of the record you wish to edit.

Example Code:
```
$fieldArray['Job Title'] = 'FileMaker Developer';
$fieldArray['Website'] = 'https://www.excelisys.com';
$fmx = new FMXData();
$fmx->addPostFieldArray($fieldArray);
$fmx->addPostField('Company','Excelisys Inc');
$fmx->addScript('FMX_Script','param1');
$result = $fmx->fmxEditRecord('CONTACT',$recId);
```

On success, returns a response array with the modId. If you included a script option the script results will also be included.

```
Array (
    [response] => Array (
        [scriptResult] => Script Run - Param: param1
        [scriptError] => 0
        [modId] => 0
    )
    [messages] => Array ( [0] => Array ( [code] => 0 [message] => OK ) )
)
```

COMPATIABLE OPTIONS
addPostFieldArray ($fieldArray)
addPostField ($fieldName, $fieldValue)
addScript ($scriptName {, $scriptParameter})
addPresortScript ($scriptName {, $scriptParameter})
addPrerequestScript ($scriptName {, $scriptParameter})

fmxDeleteRecord ($layout, $recordId)

<u>Parameters:</u>
$layout - The FileMaker Layout to query.
$recordId - The FileMaker record ID of the record you wish to delete.

<u>Example Code:</u>
```
$fmx = new FMXData();
$fmx->addScript('FMX_Script','param1');
$fmx->addPresortScript('FMX_Presort_Script','param2');
$fmx->addPrerequestScript('FMX_Prerequest_Script','param3');
$result = $fmx->fmxDeleteRecord('CONTACT',$recId);
```

On success, returns an empty response array. If you included a script option the script results will be included.

```
Array (
    [response] => Array (
        [scriptResult] => Script Run - Param: param1
        [scriptError] => 0
    )
    [messages] => Array ( [0] => Array ( [code] => 0 [message] => OK ) )
)
```

<u>COMPATIABLE OPTIONS</u>
<u>addScript ($scriptName {, $scriptParameter})</u>
<u>addPresortScript ($scriptName {, $scriptParameter})</u>
<u>addPrerequestScript ($scriptName {, $scriptParameter})</u>

fmxGetRecord ($layout, $recordId)

<u>Parameters:</u>
$layout - The FileMaker Layout to query.
$recordId - The FileMaker record ID of the record you wish to return.

<u>Example Code:</u>
```
$fmx = new FMXData();
$fmx->setResponseLayout('CONTACT');
$fmx->addScript('FMX_Script','param1');
$fmx->limitPortals('phone_portal,email_portal');
$fmx->limitPortalRows('phone_portal',1,2);
$result = $fmx->fmxGetRecord('CONTACT_LIST',15297);
```

On success, returns a response with the script results, dataInfo (which includes the base table name and record counts). The "data" array will have the record information in the first element. The record information will contain arrays of the "fieldData", "portalData", the "portalDataInfo" as well as the recordId and modId.

```
Array (
    [response] => Array (
        [scriptResult] => Script Run - Param: param1
        [scriptError] => 0
        [dataInfo] => Array ( ... )
        [data] => Array (
            [0] => Array (
                [fieldData] => Array ( ... )
                [portalData] => Array (
                    [email_portal] => Array ( ... )
                    [phone_portal] => Array ( ... )
                )
                [recordId] => 15297
                [modId] => 4
                [portalDataInfo] => Array ( ... )
            )
        )
    )
    [messages] => Array ( [0] => Array ( [code] => 0 [message] => OK ) )
)
```

<u>COMPATIABLE OPTIONS</u>
<u>setResponseLayout($layout)</u>
<u>limitPortals($portalNames)</u>
<u>limitPortalRows ($portalName, $limit, $offset)</u>
<u>addScript ($scriptName {, $scriptParameter})</u>
<u>addPresortScript ($scriptName {, $scriptParameter})</u>
<u>addPrerequestScript ($scriptName {, $scriptParameter})</u>

fmxGetRange ($layout)

Parameters:
$layout - The FileMaker Layout to query.

Example Code:
```
$fmx = new FMXData();
$fmx->setResponseLayout('CONTACT_LIST');
$fmx->addScript('FMX_Script','param1');
$fmx->addSortParam('First Name','ascend',2);
$fmx->addSortParam('Last Name','ascend',1);
$fmx->setLimitOffset(10,50);
$result = $fmx->fmxGetRange('CONTACT');
```

On success, returns a response with the script results, dataInfo (which includes the base table name and record counts). The "data" array will contain the found records. The record information will contain arrays of the "fieldData", "portalData", the "portalDataInfo" as well as the recordId and modId.

```
Array (
    [response] => Array (
        [scriptResult] => Script Run - Param: param1
        [scriptError] => 0
        [dataInfo] => Array ( ... )
        [data] => Array (
            [0] => Array (
                [fieldData] => Array ( ... )
                [portalData] => Array ( ... )
                [recordId] => 16035
                [modId] => 0 )
            [1] => Array ( ... )
        )
    [messages] => Array ( [0] => Array ( [code] => 0 [message] => OK ) )
)
```

COMPATIABLE OPTIONS
setResponseLayout($layout)
limitPortals($portalNames)
limitPortalRows ($portalName, $limit, $offset)
addScript ($scriptName {, $scriptParameter})
addPresortScript ($scriptName {, $scriptParameter})
addPrerequestScript ($scriptName {, $scriptParameter})
addSortParam($fieldName {, $direction, $sortOrder})
setLimitOffset($limit {, $offset})

fmxFindRecords($layout)

Parameters:
$layout - The FileMaker Layout to query.

Example Code:
```
$fmx = new FMXData();
$fmx->setResponseLayout('CONTACT_LIST');
$fmx->addRequestArray(array('Company'=>'Bayer Ltd'));
$fmx->addRequestArray(array('Company'=>'Bayer Group'));
$fmx->addRequestArray(array('Job Title'=>'Teacher'),TRUE);
$fmx->addSortParam('First Name','ascend',2);
$fmx->addSortParam('Last Name','ascend',1);
$fmx->setLimitOffset(10,50);
$result = $fmx->fmxFindRecords('CONTACT');
```

On success, returns a response with the script results, dataInfo (which includes the base table name and record counts). The "data" array will contain the found records. The record information will contain arrays of the "fieldData", "portalData", the "portalDataInfo" as well as the recordId and modId.

```
Array (
    [response] => Array (
        [dataInfo] => Array ( ... )
        [data] => Array (
            [0] => Array ( ... )
            [1] => Array ( ... )
        )
    [messages] => Array ( [0] => Array ( [code] => 0 [message] => OK ) )
)
```

COMPATIABLE OPTIONS
setResponseLayout($layout)
addRequestArray($requestArray {, $omit })
limitPortals($portalNames)
limitPortalRows ($portalName, $limit, $offset)
addScript ($scriptName {, $scriptParameter})
addPresortScript ($scriptName {, $scriptParameter})
addPrerequestScript ($scriptName {, $scriptParameter})
addSortParam($fieldName {, $direction, $sortOrder})
setLimitOffset($limit {, $offset})

SETTING FIELDS IN FILEMAKER

Use the addPostFieldArray( ) and addPostField( ) options to set the fields and field data you want to send to FileMaker. AddPostFieldArray( ) and addPostField( ) functions are compatible with both the fmxCreateRecord( ) and fmxEditRecord( ) functions.

addPostFieldArray($fieldArray)

Parameters:
$fieldArray - An array of field names and the associated values.

The addPostFieldArray requires an array of $key=>$value pairs where the $key will be the field name and the $value will be the data you want in the field. Usually this data comes from a form on a web page. You can assemble the array using the array( ) function or you can set each element of the array one at a time.

Example Code:
```
$fieldArray = array(
     'Job Title' => 'FileMaker Developer',
     'Website' => 'https://www.excelisys.com'
);
$fmx->addPostFieldArray($fieldArray);
```

OR

```
$fieldArray['Job Title'] = 'FileMaker Developer';
$fieldArray['Website'] = 'https://www.excelisys.com';
$fmx->addPostFieldArray($fieldArray);
```

addPostField($fieldName, $fieldValue)

Parameters:
$fieldName - The name of the field you want to set in FileMaker.
$fieldValue - The value you want to set the associated field to.

With addPostField( ) you can just add the field and value directly. You don't need to create an array.

Example Code:
```
$fmx->addPostField('Company','Excelisys Inc');
```

NOTE: You can use both addPostFieldArray( ) and addPostField( ) at the same time. All of the fields will be sent to FileMaker. You can add either of these options as many times as you need.

COMPATIABLE FUNCTIONS
```
$fmxCreateRecord ( $layout )
$fmxEditRecord ( $layout, $recordId )
```

PERFORMING FILEMAKER FINDS

The addRequestArray( ) option is used exclusively to add FileMaker find requests to the fmxFindRecords( ) functions.

addRequestArray($requestArray {, $omit })

Parameters:
$requestArray - An array of field names and the associated search criteria.
$omit - (Optional) Can be either TRUE or FALSE. Set this to TRUE to mark the request to omit the criteria.

The $requestArray parameter requires an array of $key=>$value pairs where the $key will be the field name and the $value will be the search criteria. You can use FileMaker search operators such as "=", "<", "<=", ">", '>=", "//", "*" for your find criteria.

Example Code:
```
$fmx->addRequestArray(array('Company'=>'Bayer Ltd'));
$fmx->addRequestArray(array('Company'=>'Bayer Group'));
$fmx->addRequestArray(array('Job Title'=>'Teacher'),TRUE);
```

COMPATIABLE FUNCTIONS
fmxFindRecords($layout)

SORTING THE RESULTS

Use the addSortParam( ) option to indicate how you want the results sorted. The direction parameter indicates if you want the field sorted ascending or descending. If you don't specify a direction "ascending" is the default. Use the sortOrder parameter to indicate which order to sort the field, 1, 2, 3, etc. If you do not indicate a sort order then the default is to use the order in which they are added. You can use as many addSortParam( ) options as you need.

addSortParam($fieldName {, $direction, $sortOrder})

Parameters:
$fieldName - The name of the field to sort.
$direction - (Optional) Can be either "ascend" or "decend".
$sortOrder - (Optional) Integer beginning with 1 for the first addSortParam( ) used, 2 for the second, etc.

Example Code:
```
$fmx->addSortParam('First Name','ascend',2);
$fmx->addSortParam('Last Name','ascend',1);
```

COMPATIABLE FUNCTIONS
fmxGetRange ($layout)
fmxFindRecords($layout)

LIMITING THE FIND RESULTS

Use the setLimitOffset( ) option to limit the result set. The default limit for records is 100. The default offset is 1.

setLimitOffset($limit {, $offset})

Parameters:
$limit - The maximum number of records to return.
$offset - (Optional) The number of the first record to return (records before it are skipped).

Example Code:
```
$fmx->setLimitOffset(10,50);
```

COMPATIABLE FUNCTIONS
fmxGetRange ($layout)
fmxFindRecords($layout)

WORKING WITH FILEMAKER SCRIPTS

You can execute FileMaker Scripts in just about all the fmxData record functions. There are three options to trigger FileMaker Scripts. The three options are addScriptPrerequest( ), addScriptPresort( ), and addScript( ). They work in this order...

addScriptPrerequest( ) - Executes the script before the find is run.

[The find is run]

addScriptPresort( ) - Executes after the find but before and sorts are run.

[The sorts are run]

addScript( ) - Executes at the end of the request.

addScriptPrerequest($scriptName {, $scriptParameter)
addScriptPresort($scriptName {, $scriptParameter)
addScript($scriptName {, $scriptParameter)

Parameters:
$scriptName - The name of the script you want to trigger.
$scriptParameter - (Optional) The script parameter you want to pass to the script.

Example Code:
```
$fmx->addPrerequestScript('FMX_Prerequest_Script','param3');
$fmx->addPresortScript('FMX_Presort_Script','param2');
$fmx->addScript('FMX_Script','param1');
```

NOTE: You can use all three at the same time, but you can only use them once each.

COMPATIABLE FUNCTIONS
fmxCreateRecord ($layout)
fmxEditRecord ($layout, $recordId)
fmxDeleteRecord ($layout, $recordId)
fmxGetRecord ($layout, $recordId)
fmxGetRange ($layout)
fmxFindRecords($layout)

SETTING THE RESPONSE LAYOUT

For many functions you can use the setResponseLayout( ) option to determine which layout your query is returned from. For example, you can use the fields on a detail layout to perform a complex find, but set the response layout to a list view that may have a limited number of fields on it.

setResponseLayout($layout)

Parameters:
$layout - The FileMaker Layout to query.

Example Code:
```
$fmx->setResponseLayout('CONTACT_LIST');
```

COMPATIABLE FUNCTIONS
fmxGetRecord ($layout, $recordId)
fmxGetRange ($layout)
fmxFindRecords($layout)

WORKING WITH PORTALS

There are two options to help you work with portals, limitPortals( ) and limitPortalRows( ). Using limitPortals( ) you can specify which portals to return. The limitPortalRows( ) option allows you to set a limit and offset for each portal.

limitPortals($portalNames)

Parameters:
$portalNames - A comma delimited list of the portal object names from the layout in FileMaker.

Example Code:
```
$fmx->limitPortals('phone_portal,email_portal');
```

Use limitPortalRows( ) to limit the number of portal records returned. The default limit for portals is 50. The default offset is 1.

limitPortalRows ($portalName, $limit {, $offset })

Parameters:
$portalName - The portal object names from the layout in FileMaker.
$limit - The number of records to return from the portal.
$offset - (Optional) The number of the first record to return (records before it are skipped).

Example Code:
```
$fmx->limitPortalRows('phone_portal',1,2);
```

COMPATIABLE FUNCTIONS
fmxGetRecord ($layout, $recordId)
fmxGetRange ($layout)
fmxFindRecords ($layout)

UPLOADING CONTAINER FIELD FILES

You can upload container field files with the fmxUploadFile( ) function. Unlike the other record functions you do not need create a new class instance, you can just call this function directly.

It's important to note that the $pathToFile parameter must be a full file path. Don't get this confused with a URL, this is the path to the file relative to the root of the disk drive. You can use the magic constant __DIR__ to get the full path to the directory of the current PHP file, then you can just append the path to your upload file from there.

fmxUploadFile($layout, $recordId, $fieldName, $repetition, $pathToFile)

Parameters:
$layout - The FileMaker Layout to query.
$recordId - The FileMaker record ID of the record you wish to edit.
$fieldName - The name of the container field you wish to upload to.
$repetition - The number of the field repetition you want to upload into.
$pathToFile - The full path to the file that needs to be uploaded.

Example Code:
```
$path = __DIR__.'/photo.jpg';
$upload = FMXData::fmxUploadFile('CONTACT',15297,'Photo',1,$path);
```

On success, returns a response array with the modId of the record and a message array with the status.

```
Array (
    [response] => Array ( [modId] => 3 )
    [messages] => Array ( [0] => Array ( [code] => 0 [message] => OK ) )
)
```

SETTING GLOBAL FIELDS

You can use fmxSetGlobalFields( ) and addGlobalField( ) to set global fields in the database. Use this to set globals before running a script or performing a find. The global field values will persist until you run fmxDisconnect( ) and terminate the session. Use the addGlobalField( ) option to specify the fields and values you want to set. You can use as many addGlobalField( ) options as you need.

fmxSetGlobalFields( )

Example Code:
```
$fmx = new FMXData();
$fmx->addGlobalField('CONTACT','global1','FileMaker');
$fmx->addGlobalField('CONTACT','global2','Rocks');
$result = $fmx->fmxSetGlobalFields();
```

On success, returns an empty response array and a message array with the status.

```
Array (
    [response] => Array ( )
    [messages] => Array ( [0] => Array ( [code] => 0 [message] => OK ) )
)
```

addGlobalField($baseTable, $fieldName, $value)

Parameters:
$baseTable - The base table name in FileMaker.
$fieldName - The name of the field you wish to set.
$value - The value you want to set the field to.