A PHP extension for InterSystems M/Caché/IRIS and YottaDB

mg_php

M/Gateway Developments Ltd.
Chris Munt

Revision History:

13 June 2019

Contents

1 Introduction

mg_php is an Open Source PHP extension providing direct access to InterSystems **Caché**, **IRIS** and the **YottaDB** database. It will also work with other M-like databases.

Although this document refers to ***InterSystems Caché*** throughout, it can be assumed that the same applies to ***InterSystems IRIS***.

2 Pre-requisites

It is assumed that you have the following three components already installed:

- A Web Server
- PHP http://www.php.net

Either:

- InterSystems Caché or IRIS http://www.intersystems.com

Or:

- YottaDB https://www.yottadb.com

3 Installing mg_php

There are three parts to **mg_php** installation and configuration.

- The PHP extension (**mg_php.so** or **mg_php.dll**).
- The database (or server) side code: **zmgsi.**
- A network configuration to bind the former two elements together.

3.1 InterSystems Caché or IRIS

Log in to the Manager UCI and, using the %RI utility (or similar) load the **zmgsi** routines held in **/m/zmgsi.ro**. Change to your development UCI and check the installation:

```
do ^%zmgsi

M/Gateway Developments Ltd - Service Integration Gateway
Version: 3.0; Revision 1 (13 June 2019)
```

3.2 YottaDB

The instructions given here assume a standard 'out of the box' installation of **YottaDB** deployed in the following location:

```
/usr/local/lib/yottadb/r122
```

The primary default location for routines:

```
/root/.yottadb/r1.22_x86_64/r
```

Copy all the routines (i.e. all files with an 'm' extension) held in the GitHub **/yottadb** directory to:

```
/root/.yottadb/r1.22_x86_64/r
```

Change directory to the following location and start a **YottaDB** command shell:

```
cd /usr/local/lib/yottadb/r122
./ydb
```

Check the installation:

```
do ^%zmgsi

M/Gateway Developments Ltd - Service Integration Gateway
Version: 3.0; Revision 1 (13 June 2019)
```

Note that the version of **zmgsi** is successfully displayed.

3.3 Setting up the network service

The default TCP server port for **zmgsi** is **7041**. If you wish to use an alternative port then modify the following instructions accordingly. PHP code using the **mg_php** functions will, by default, expect the database server to be listening on port **7041** of the local server (localhost). However, **mg_php** provides the functionality to modify these default settings at run-time. It is not necessary for the web server/PHP installation to reside on the same host as the database server.

3.3.1 InterSystems Caché or IRIS

Start the M-hosted concurrent TCP service in the Manager UCI:

```
do start^%zmgsi(0)
```

To use a server TCP port other than 7041, specify it in the start-up command (as opposed to using zero to indicate the default port of 7041).

3.3.2 YottaDB

Network connectivity to **YottaDB** is managed via the **xinetd** service. First create the following launch script (called **zmgsi_ydb** here):

```
/usr/local/lib/yottadb/r122/zmgsi_ydb
```

Content:

```
#!/bin/bash
cd /usr/local/lib/yottadb/r122
export ydb_dir=/root/.yottadb
export ydb_dist=/usr/local/lib/yottadb/r122
export
ydb_routines="/root/.yottadb/r1.22_x86_64/o*(/root/.yottadb/r1.22_x86_6
4/r /root/.yottadb/r) /usr/local/lib/yottadb/r122/libyottadbutil.so"
export ydb_gbldir="/root/.yottadb/r1.22_x86_64/g/yottadb.gld"
$ydb_dist/ydb -r xinetd^%zmgsis
```

Create the **xinetd** script (called **zmgsi_xinetd** here):

```
/etc/xinetd.d/zmgsi_xinetd
```

Content:

```
service zmgsi_xinetd
{
     disable         = no
     type            = UNLISTED
     port            = 7041
     socket_type     = stream
     wait            = no
     user            = root
     server          = /usr/local/lib/yottadb/r122/zmgsi_ydb
}
```

- Note: sample copies of **zmgsi_xinetd** and **zmgsi_ydb** are included in the /unix directory.

Edit the services file:

```
/etc/services
```

Add the following line to this file:

```
zmgsi_xinetd          7041/tcp                        # ZMGSI
```

Finally restart the **xinetd** service:

```
/etc/init.d/xinetd restart
```

3.4 PHP configuration

PHP should be configured to recognise the **mg_php** extension. The PHP configuration file (**php.ini**) is usually found in the following locations:

3.4.1 UNIX:

```
/usr/local/lib/php.ini
```

Add the following line to the extensions section:

```
extension=mg_php.so
```

Finally, install the **mg_php.so** file in the PHP modules directory, which is usually:

```
/usr/local/lib/php/extensions/[version_information]/
```

3.4.2 Windows:

```
C:\Windows\php.ini
```

Add the following line to the extensions section:

```
extension=mg_php.dll
```

Finally, install the **mg_php.dll** file in the PHP modules directory, which is usually:

```
C:\Windows\System32\
```

4 PHP Arrays and M/Caché Globals

mg_php exploits the close conceptual relationship between PHP arrays and M/Caché globals. PHP arrays can be merged into M/Caché globals and M/Caché globals can be merged into PHP arrays.

A single-dimensional PHP array can be made to correspond to its equivalent M/Caché global, or a specific section of another M/Caché global. For example:

PHP Array:

```
$Customer[1234]=”Chris Munt”
$Customer[1235]=”Rob Tweed”
```

Corresponding M/Caché Global:

```
^Customer(1234)=”Chris Munt”
^Customer(1235)=”Rob Tweed”
```

The same is true for multi-dimensional arrays. For example:

PHP Array:

```
$CustomerInvoice[1234][1]="1.2.2001"
$CustomerInvoice[1234][2]="5.3.2001”
```

Corresponding M/Caché global:

```
^CustomerInvoice(1234,1)="1.2.2001"
^CustomerInvoice(1234,2)="5.3.2001”
```

However, with multi-dimensional arrays there is one important difference between PHP and M/Caché.

A M/Caché global is underpinned by a B-Tree storage system. The unique key to a particular node within the tree can be divided up into individual components known as ‘subscripts’. Subscripts give M/Caché globals their multi-dimensional characteristics.

A PHP array, on the other hand, is inherently a single-dimensional storage construct. You can, however, set a data node within a PHP array to point to another array. Multi-dimensional arrays in PHP are therefore arrays within arrays.

What this means in practice is that, unlike M/Caché globals, data nodes within PHP arrays are unable to simultaneously point to data and further subscripts. For example, in M/Caché it is possible to generate data structures such as:

```
^Customer(1234)="Chris Munt"
^Customer(1234,"city")="London"
```

In this Global, node '1234' points to both data ("Chris Munt") and a second dimension as indicated by the second subscript ("city").

The equivalent PHP array, if it were possible to express such a data structure, would be:

```
$Customer[1234]="Chris Munt"
$Customer[1234]["city"]="London"
```

However, this is not possible because data node '$Customer[1234]' cannot simultaneously take the value of a string ("Chris Munt") and an array (["city"]="London").

In **mg_php** we overcome this difficulty by adopting a convention whereby the data value in such cases is represented within the adjacent array, keyed by a white space. For example:

```
$Customer[1234][" "]="Chris Munt"
$Customer[1234]["city"]="London"
```

Of course, this array, when merged to M/Caché, will generate the global structure shown below:

```
^Customer(1234)="Chris Munt"
^Customer(1234,"city")="London"
```

5 Function Reference for mg_php

The PHP functions allow you to directly manipulate the M/Caché database from within the PHP programming environment. Functions are also supplied to allow you to directly call M/Caché extrinsic functions and M/Caché procedures that are capable of generating sections of HTML form data.

5.1 Direct access to the M/Caché database

The functions in this section give you direct access to the M/Caché commands for manipulating global data. A M/Caché global is essentially a multi-dimensional associative array that's held in permanent storage.

For example, two records in a Customer database would look something like:

```
^Customer(1234)="Chris Munt"
^Customer(1235)="Rob Tweed"
```

As previously stated, globals can be multi-dimensional. For example, the Invoices database for a customer may look something like:

```
^CustomerInvoice(1234,1)="1.2.2001"
^CustomerInvoice(1234,2)="5.3.2001"
```

A note on terminology: The global and its subscripts are usually known as the 'global node'. For example:

```
^Customer(1234)
```

A global node points directly to a record. A record (or row) usually consists of a number of fields (or columns), separated by a delimiter.

5.1.1 Set a global node.

```
m_set(<global> {, subscript(s) …}, <data>)
```

By convention, the last argument is always the global node's data record.

Example 1:

Caché command:

```
Set ^Customer(1234)="Chris Munt"
```

Equivalent PHP function:

```
m_set("^Customer", 1234, "Chris Munt");
```

5.1.2 Retrieve the data from a global node.

```
data = m_get(<global> {, subscript(s) …})
```

Example 1:

Caché command:

```
Set data=$Get(^Customer(1234))
```

Equivalent PHP function:

```
data = m_get("^Customer", 1234);
```

5.1.3 Check that a global node exists.

```
defined = m_data(<global> {, subscript(s) …})
```

Example 1:

Caché command:

```
Set defined=$Data(^Customer(1234))
```

Equivalent PHP function:

```
defined = m_data("^Customer", 1234);
```

5.1.4 Delete a global node.

```
m_kill(<global> {, subscript(s) …})
```

Example 1:

Caché command:

```
Kill ^Customer(1234))
```

Equivalent PHP function:

```
m_kill("^Customer", 1234);
```

5.1.5 Get the next record (subscript) from a global node.

```
next = m_order(<global> {, subscript(s) …})
```

Example 1:

Caché command:

```
Set nextID=$Order(^Customer(1234))
```

Equivalent PHP function:

```
nextID = m_order("^Customer", 1234);
```

5.1.6 Get the previous (subscript) from a global node.

```
prev = m_previous(<global> {, subscript(s) …})
```

Example 1:

Caché command:

```
Set prevID=$Order(^Customer(1234),-1)
```

Equivalent PHP function:

```
prevID = m_previous("^Customer", 1234);
```

5.1.7 Merge a PHP array to a global.

```
result = m_merge_to_db(<global> {, subscript(s) …},
                       <php_array>, <options>)
```

The 'options' argument can currently take the following value:

```
ks
```

This means '**K**ill at **S**erver'. If this option is selected, the M/Caché global will be deleted at the level specified within the merge function before the actual merge is performed.

Example 1:

Caché commands:

```
Set custList(1234)=”Chris Munt”
Set custList(1235)=”Rob Tweed”
Merge ^Customer=custList
```

Equivalent PHP function:

```
$custList = array("1234" => "Chris Munt",
                  "1235" => "Rob Tweed”);
$options = “”;
result = m_merge_to_db(“^Customer”, $custList, $options);
```

The following records will be created in M/Caché:

```
^Customer(1234)=”Chris Munt”
^Customer(1235)=”Rob Tweed”
```

Example 2:

Caché commands:

```
Set custInvoice(1)=”1.2.2001”
Set custInvoice(2)=”5.3.2001”
Merge ^CustomerInvoice(1234)=custInvoice
```

Equivalent PHP function:

```
$custInvoice = array("1" => "1.2.2001",
                     "2" => "5.3.2001”);
$options = “”;
result = m_merge_to_db(“^CustomerInvoice”, 1234, $custInvoice, $options);
```

The following records will be created in M/Caché:

```
^CustomerInvoice(1234,1)="1.2.2001"
^CustomerInvoice(1234,2)="5.3.2001”
```

Example 3 (Using the ‘ks’ option):

Existing M/Caché database:

```
^CustomerInvoice(1234,1)="1.2.2001"
^CustomerInvoice(1234,2)="5.3.2001”
^CustomerInvoice(1234,3)="7.9.2001”
```

Caché commands:

```
Set custInvoice(3)=”8.9.2001”
Set custInvoice(4)=”12.11.2001”
Kill ^CustomerInvoice(1234)
```



```
Merge ^CustomerInvoice(1234)=custInvoice
```

Equivalent PHP function:

```
$custInvoice = array("3" => "8.9.2001",
                     "4" => "12.11.2001”);
$options = “ks”;
result = m_merge_to_db(“^CustomerInvoice”, 1234, $custInvoice, $options);
```

After this operation, M/Caché will hold the following records:

```
^CustomerInvoice(1234,3)="8.9.2001"
^CustomerInvoice(1234,4)="12.11.2001”
```

Example 4 (Dealing with multi-dimensional arrays):

Caché commands:

```
Set custInvoice(1234,1)=”1.2.2001”
Set custInvoice(1234,2)=”5.3.2001”
Set custInvoice(1235,7)=”7.6.2002”
Set custInvoice(1235,8)=”1.12.2002”
Merge ^CustomerInvoice=custInvoice
```

Equivalent PHP function:

```
$custInvoice[1234][1]=”1.2.2001”;
$custInvoice[1234][2]=”5.3.2001”;
$custInvoice[1235][7]=”7.6.2002”;
$custInvoice[1235][8]=”1.12.2002”;
$options = “”;
result = m_merge_to_db(“^CustomerInvoice”, $custInvoice, $options);
```

After this operation, M/Caché will hold the following records:

```
^CustomerInvoice(1234,1)=”1.2.2001”
^CustomerInvoice(1234,2)=”5.3.2001”
^CustomerInvoice(1235,7)=”7.6.2002”
^CustomerInvoice(1235,8)=”1.12.2002”
```


5.1.8 Merge a global to a PHP array.

```
result = m_merge_from_db(<global> {, subscript(s) …},
                         <php_array>, <options>)
```

The last ‘options’ argument is reserved for future use.

Example 1:

Caché database:

```
^Customer("1234")="Chris Munt"
^Customer("1235")="Rob Tweed”
```

Caché command:

```
Merge custList=^Customer
```

Equivalent PHP function:

```
$custList = array();
$options = “”;
result = m_merge_from_db(“^Customer”, $custList, $options);
```

The following PHP array will be created:

```
$custList["1234"]="Chris Munt"
$custList["1235"]="Rob Tweed”
```

Example 2:

Caché database:

```
^CustomerInvoice(1234,1)="1.2.2001"
^CustomerInvoice(1234,2)="5.3.2001”
```

Caché commands:

```
Merge custInvoice=^CustomerInvoice(1234)
```

PHP function:

```
$custInvoice = array();
$options = “”;
result = m_merge_from_db(“^CustomerInvoice”, 1234, $custInvoice, $options);
```

The following PHP array will be created:

```
$custInvoice["1"]="1.2.2001"
$custInvoice["2"]="5.3.2001”
```

Example 3 (Dealing with multi-dimensional arrays):

Caché database:

```
^CustomerInvoice(1234,1)=”1.2.2001”
^CustomerInvoice(1234,2)=”5.3.2001”
^CustomerInvoice(1235,7)=”7.6.2002”
^CustomerInvoice(1235,8)=”1.12.2002”
```

Caché commands:

```
Merge custInvoice=^CustomerInvoice
```

Equivalent PHP function:

```
$custInvoice = array();
$options = “”;
result = m_merge_from_db(“^CustomerInvoice”, $custInvoice, $options);
```

The following PHP array will be created:

```
$custInvoice[1234][1]=”1.2.2001”;
$custInvoice[1234][2]=”5.3.2001”;
$custInvoice[1235][7]=”7.6.2002”;
$custInvoice[1235][8]=”1.12.2002”;
```

5.2 Direct access to M/Caché functions and procedures

Caché provides a rich scripting language for developing function and procedures. The following functions are supplied by **mg_php** for the purpose of directly accessing M/Caché functions and procedures within the PHP environment.

A note on terminology: M/Caché procedures and functions are contained within blocks of code known as routines. A function or procedure is referred to using the following syntax:

```
<FunctionName>^<Routine>
```

For example:

```
MyFunction^MyRoutine
```


A routine name of ‘MyRoutine’ will be used throughout the following examples.

5.2.1 Call a M/Caché extrinsic function.

```
result = m_proc(<function> {, argument(s) …})
```

And:

```
result = m_proc_byref(<function> {, argument(s) …})
```

The ‘***byref’*** version of this function should be used when modifying input parameters (e.g. arrays) in M/Caché and passing the modified (or new) values back to PHP.

Example 1 (A simple function call):

Caché procedure:

```
GetTime()   ; Get the current date and time in M/Caché’s internal
format
            Set result=$Horolog
            Quit result
            ;
```

This function returns the date and time in M/Caché’s internal format.

Equivalent PHP function:

```
time = m_proc(“GetTime^MyRoutine”);
```

Example 2 (Passing arguments by reference):

Caché procedure:

```
GetDateDecoded(dateDisp)      ; Get the current date in decoded form
                              Set result=+$Horolog
                              Set dateDisp=$ZD(result,2);
                              Quit result
                              ;
```

This function returns the date in M/Caché’s internal format. In addition, it will return the date in a human-readable format (i.e. decoded).

Equivalent PHP function (default server):

```
date = m_proc_byref(“GetDateDecoded^MyRoutine”, $dateDisp);
```


Example 3 (Another simple function call):

Caché procedure:

```
GetCust(custID)   ; Return the customer name
                  Set cust=$Get(^Customer(custID))
                  Quit cust
                  ;
```

Equivalent PHP function:

```
cust = m_proc("GetCust^MyRoutine", 1234);
```

Example 4 (Passing an array from PHP to M/Caché):

Caché procedure:

```
ProcCustList(custList)     ; Return the number of active customers
                  Set custID="",activeCust=0
                  For  Set CustID=$Order(^CustList(custID)) Do
                  . If $Data(^CustOrderStatus(custID))="Active" Do
                  .. Set activeCust=activeCust+1
                  Quit activeCust
                  ;
```

Equivalent PHP function:

```
$custList = array("1234" => "", "1235" => "", "1236" => "");
activeCust = m_proc("ProcCustList^MyRoutine", $custList);
```

Example 5 (Passing an array from M/Caché to PHP):

Caché procedure:

```
ActCList(custList) ; Return a list of active customers
 Set custID="",activeCust=0
 For  Set CustID=$Order(^CustOrderStatus(custID)) Quit:custID=""  Do
  . If $Data(^CustOrderStatus(custID))="Active" Do
  .. activeCust=activeCust+1
  .. Set custList(custID)=$Get(^Customer(custID))
  Quit activeCust
  ;
```


Equivalent PHP function:

```
$custList = array();
activeCust = m_proc_byref("ActCList^MyRoutine", $custList);
```

Example 6 (Using a M/Caché function to modify a PHP array):

Caché procedure:

```
GetCustNames(custList)     ; Return the names for a list of customers
                   Set custID="",result=""
                   For  Set CustID=$Order(^CustList(custID)) Do
                   . Set custList(custID)=$Get(^Customer(custID))
                   Quit result
                   ;
```

Equivalent PHP function:

```
$custList = array("1234" => "", "1235" => "", "1236" => "");
custID = m_proc_byref("GetCustNames^MyRoutine", $custList);
```

5.2.2 Return a block of HTML from a M/Caché function.

```
m_html(<function> {, argument(s) …})
```

Example 1:

Caché procedure:

```
MyHtml      ; Return some HTML to PHP
            Write "<p>This text was returned from M/Caché"
            Write "<br>You can return as text much as you like …"
            Quit
            ;
```

Equivalent PHP function:

```
m_html("MyHtml^MyRoutine");
```

Example 2:

Caché procedure:

```
GetHTML(custID) ; Return some HTML to PHP
             Write "<p>This text was returned from M/Caché"
             Write "<br>You can return as text much as you like …"
             Write "<br>A single argument '",custID,"' was passed from PHP"
             Quit
             ;
```

Equivalent PHP function:

```
m_html("GetHTML^MyRoutine", 1234);
```

Example 3 (Passing an array from PHP to M/Caché):

Caché procedure:

```
GetCTable(custList) ; Return a table of customers to PHP
             Write "<table>"
             Set custID=""
             For  Set CustID=$Order(^CustList(custID)) Quit:custID=""  Do
             . Set custName=$Get(^Customer(custID))
             . Set custList(custID)=custName ; Return name to PHP
             . Write "<tr><td>",custID,"</td>"
             . Write "<td>",custName,"</td></tr>"
             Write "</table>"
             Quit
             ;
```

Equivalent PHP function :

```
$custList = array("1234" => "", "1235" => "", "1236" => "");
m_html("GetCTable^MyRoutine", $custList);
```

5.3 Direct access to Caché methods

Caché provides an Object Oriented development environment (Caché Objects). The following functions are supplied by mg_php for the purpose of directly accessing Caché class methods from within the PHP environment.

This section will show the same examples used in the previous section (Caché functions and procedures), but implemented as methods of an object class.

The methods described here will belong to a class called 'MyUtilities.MyClass'. This translates to a Caché package name of 'MyUtilities' and a class name of 'MyClass'. Of course, in a real application the methods described would be contained within a class more appropriate to the functionality they implement.

5.3.1 Call a Caché method.

```
result = m_method(<class_name>,
                  <method_name> {, argument(s) …})
```

And:

```
result = m_method_byref(<class_name>,
                  <method_name> {, argument(s) …})
```

The '***byref***' version of this function should be used when modifying input parameters (e.g. arrays) in Caché and passing the modified (or new) values back to PHP.

Example 1 (A simple method):

Caché method:

```
Class MyUtilities.MyClass Extends etc …

ClassMethod GetTime()
{
      ; Get the current date and time in M/Caché's internal format
      Set result=$Horolog
      Quit result
}
```

This method returns the date and time in Caché's internal format.

Equivalent PHP function:

```
time = m_method("MyUtilities.MyClass", "GetTime");
```

Example 2 (Passing arguments by reference):

Caché method:

```
Class MyUtilities.MyClass Extends etc …

ClassMethod GetDateDecoded(dateDisp)
{
      ; Get the current date in decoded form
      Set result=+$Horolog
      Set dateDisp=$ZD(result,2);
      Quit result
}
```

This method returns the date in Caché's internal format. In addition, it will return the date in a human-readable format (i.e. decoded).

Equivalent PHP function:

```
date = m_method_byref("MyUtilities.MyClass", "GetDateDecoded", $dateDisp);
```

Example 3 (Another simple method):

Caché method:

```
Class MyUtilities.MyClass Extends etc …

ClassMethod GetCust(custID)
{
      ; Return the customer name
      Set cust=$Get(^Customer(custID))
      Quit cust
}
```

Equivalent PHP function:

```
cust = m_method("MyUtilities.MyClass", "GetCust", 1234);
```

Equivalent PHP function (named server):

```
cust = m_method("MyCachéServer", "MyUtilities.MyClass",
                 "GetCust", 1234);
```

Example 4 (Passing an array from PHP to Caché):

Caché method:

```
Class MyUtilities.MyClass Extends etc …

ClassMethod ProcCustList(custList)
{
      ; Return the number of active customers
      Set custID="",activeCust=0
      For  Set CustID=$Order(^CustList(custID)) Do
      . If $Data(^CustOrderStatus(custID))="Active" Do
      .. Set activeCust=activeCust+1
      Quit activeCust
}
```


Equivalent PHP function:

```
$custList = array("1234" => "", "1235" => "", "1236" => "");
activeCust = m_method_byref("MyUtilities.MyClass", "ProcCustList",
                            $custList);
```

Example 5 (Passing an array from Caché to PHP):

Caché method:

```
Class MyUtilities.MyClass Extends etc …

ClassMethod ActCList(custList)
{
       ; Return a list of active customers
       Set custID="",activeCust=0
       For  Set CustID=$Order(^CustOrderStatus(custID)) Quit:custID=""  Do
                . If $Data(^CustOrderStatus(custID))="Active" Do
                          .. activeCust=activeCust+1
       .. Set custList(custID)=$Get(^Customer(custID))
       Quit activeCust
}
```

Equivalent PHP function:

```
$custList = array();
activeCust = m_method_byref("MyUtilities.MyClass", "ActCList", $custList);
```

Example 6 (Using a Caché method to modify a PHP array):

Caché method:

```
Class MyUtilities.MyClass Extends etc …

ClassMethod GetCustNames(custList)
{
       ; Return the names for a list of customers
       Set custID="",result=""
       For  Set CustID=$Order(^CustList(custID)) Do
       . Set custList(custID)=$Get(^Customer(custID))
       Quit result
}
```

Equivalent PHP function:

```
$custList = array("1234" => "", "1235" => "", "1236" => "");
custID = m_method_byref("MyUtilities.MyClass", "GetCustNames", $custList);
```


5.3.2 Return a block of HTML from a Caché method.

```
m_html_method(<class_name>, <method_name> {, argument(s) …})
```

Example 1:

Caché method:

```
Class MyUtilities.MyClass Extends etc …

ClassMethod MyHtml()
{
      ; Return some HTML to PHP
      Write "<p>This text was returned from Caché"
      Write "<br>You can return as text much as you like …"
      Quit
}
```

Equivalent PHP function:

```
m_html_method("MyUtilities.MyClass", "MyHtml");
```

Example 2:

Caché method:

```
Class MyUtilities.MyClass Extends etc …

ClassMethod GetHTML(custID)
{
      ; Return some HTML to PHP
      Write "<p>This text was returned from Caché"
      Write "<br>You can return as text much as you like …"
      Write "<br>A single argument '",custID,"' was passed from PHP"
      Quit
}
```

Equivalent PHP function:

```
m_html_method("MyUtilities.MyClass", "GetHTML", 1234);
```

Example 3 (Passing an array from PHP to Caché):

Caché method:

```
Class MyUtilities.MyClass Extends etc …

ClassMethod GetCTable(custList)
{
      ; Return a table of customers to PHP
      Write “<table>”
      Set custID=””
      For  Set CustID=$Order(^CustList(custID)) Quit:custID=””  Do
      . Set custName=$Get(^Customer(custID))
      . Set custList(custID)=custName ; Return name to PHP
      . Write “<tr><td>”,custID,”</td>”
      . Write “<td>”,custName,”</td></tr>”
      Write “</table>”
      Quit
}
```

Equivalent PHP function:

```
$custList = array("1234" => "", "1235" => "", "1236" => "");
m_html_method(“MyUtilities.MyClass”, “GetCTable”, $custList);
```


5.4 Direct access to PHP and M/Caché functions from the browser

There are many situations in web application programming where it is desirable to directly access server-side functionality from the context of the browser environment. For example, such a facility could be used for performing individual field validation and simple table lookups. For trivial operations of this sort it is rather expensive to have to submit the whole form to the server in order to communicate with the database. Browser components are supplied with **mg_php** to provide the capability of accessing PHP and, subsequently, M/Caché functionality directly through browser-based scripting (i.e. JavaScript code).

Using the XMLHTTP script (m_client.js)

The XMLHTTP script file (JavaScript) is included in the **mg_php** distribution:

```
/js/m_client.js
```

The following procedure should be used:

1. Include the JavaScript file in the hosting page instead of the Java Applet:

```
<script language="JavaScript" src="/m_client.js"></script>
```

Example:

```
<HTML>
<HEAD><TITLE>My Form</TITLE>
<script language="JavaScript" src="/m_client.js"></script>
</HEAD>
<BODY>

etc ...
```

2. Having initialized the JavaScript environment for 'm_client.js' as described in the previous step, the internal functions can be used.

For example:

```
result = server_proc(<URL>);
```

```
<?php
   $m_fun = $_POST['m_fun'];
   $m_arg = $_POST['m_arg'];
   if (!is_null($m_fun)) {
      if ($m_fun == "NameLookup")
         m_return_to_client(m_get("^MGWCust", $m_arg));
      die();
   }
?>
<html>
<head>
<script language="JavaScript" src="/m_client.js"></script>
<script LANGUAGE = "JavaScript">
function NameLookup(FormObject, value) {
   FormObject.value = server_proc(
                  "/GetName.php?m_fun=NameLookup&m_arg=" + value);
   return;
}
</script>
<TITLE>PHP to M - applet demo</TITLE>
</head>
<body>
<form>
<h1>PHP to M - applet demo</h1>
Customer No <INPUT TYPE=TEXT NAME=id SIZE=30
ONCHANGE="NameLookup(form.name, this.value)">
Name <INPUT TYPE=TEXT NAME=name SIZE=30>
<p>
Setup database when we load form for the first time ...
<?php
   m_set("^MGWCust", 1, "Chris Munt");
   m_set("^MGWCust", 2, "Rob Tweed");
?>
<hr>
</form>
</body>
</html>
```

5.5 Handling error conditions

```
result = m_set_error_mode(<mode>[, error_code])

error = m_get_last_error()
```

Occasionally it is necessary for an **mg_php** function to return an error condition after failing to complete the prescribed task. For example, a target M/Caché server may be unavailable or there may be a problem with the supporting network.

On failure, the default behavior for all **mg_php** functions is to return a trappable fatal error code to the PHP environment. In the absence of any user-defined error handling procedures, PHP will, by default, display the error text and terminate the script. However, you can trap these errors (PHP error code: E_USER_ERROR) using the standard PHP error handling facilities (e.g. set_error_handler()).

The error handling behavior can be modified using the **m_set_error_mode** function.

Mode 0

```
m_set_error_mode(0);
```

The default behavior as described above.

Mode 1

```
m_set_error_mode(1);
```

On error, return a PHP warning (E_USER_WARNING) instead of a full error code. PHP warnings are not fatal and PHP will complete the execution of the script after writing the warning message to the screen. PHP warnings can be trapped using the standard PHP error handling facilities.

Mode 9

```
m_set_error_mode(9);
```

On error, the **mg_php** functions will return an error code. The value of the error code is minus 1 by default (-1). The corresponding error message can be obtained using the 'm_get_last_error' function. For example:

```
m_set_error_mode(9);

$e = m_kill("^MGWCust");
if ($e == -1) {
   $m = m_get_last_error();
   echo "<br>ERROR: ",$e," ", $m;
   die();
}
```

When using this mode of operation, be careful to avoid infinite looping conditions when using functions such as **m_order().**

Of course, in some cases the error code of '-1' may clash with legitimate return values. If this is anticipated to be the case, you can specify your own error code as follows:

```
m_set_error_mode(9, -7);

$e = m_kill("^MGWCust");
if ($e == -7) {
   $m = m_get_last_error();
   echo "<br>ERROR: ",$e," ", $m;
   die();
}
```

Scope

The **'m_set_error_mode'** function operates on a per-page basis. This function should be called at (or near) the beginning of each page if non-default error handling is to be used.

5.6 Handling PHP strings that exceed the maximum size allowed under M/Caché

Caché, in common with other M-based systems imposes a hard limit on the length of string that can be assigned to global nodes and variables in programs. In Caché the (default) maximum string length is 32767 Bytes.

PHP does not impose any such limit and the following convention can be used to trade 'oversize' strings between M/Caché and PHP. Oversize strings are broken up into individual sections in M/Caché as follows:

```
variable = <First Section>
variable(extra, <Section Number>) = <Subsequent Section>
```

For example, take a string that exceeds the maximum length allowed in M/Caché by a factor of three:


```
variable = <First Section>
variable(extra, 1) = <Second Section>
variable(extra, 2) = <Third Section>
```

The same convention applies to arrays:

```
array("key") = <First Section>
array("key", extra, 1) = <Second Section>
array("key", extra, 2) = <Third Section>
```

The special variable '**extra**' is set by the **mg_php** engine. Its default value is **ASCII 1**. There is a possibility that this special data marker will clash with your data particularly where arrays are concerned. For example, you may have data keyed by the default value of ASCII 1. If this is the case, you can change the value of extra by editing its value in **mg_php** core routine: `vars^%zmgsis`

```
vars  ; Public system variables
      Set extra=$C(1)
```

It should be noted however that this variable can only be reassigned on a per-installation basis in the above procedure. It should not be dynamically changed in other code.

mg_php will handle the transformation of oversize values to and from this form between PHP and M/Caché and vice versa. The PHP software is unaffected by these transformations.

Example 1 (Pass an oversize PHP variable to M/Caché):

M/Caché procedure:

```
MyFun(arg)  ; Accept an oversize variable
            Set s1=$Get(arg)
            Set s1=$Get(arg(extra,1))
            Set s1=$Get(arg(extra,2))
            Quit 1
            ;
```

Equivalent PHP function (default server):

```
$arg = "large value …….";
result = m_proc("MyFun^MyRoutine", $arg);
```

Equivalent PHP function (named server):

```
$arg = "large value …….";
result = m_proc("MyCachéServer", "MyFun^MyRoutine", $arg);
```


Example 2 (Pass an oversize M/Caché variable to PHP – by reference):

Caché procedure:

```
MyFun(arg)  ; Accept an oversize variable
            Set arg=”first section ……”
            Set arg(extra,1)=”second section ……”
            Set arg(extra,2))=”third section ……”
            Quit 1
            ;
```

Equivalent PHP function:

```
result = m_proc(“MyFun^MyRoutine”, $x);
```

Example 3 (Pass an oversize M/Caché variable to PHP – by return value):

Note the use of the ‘**`oversize`**’ flag to instruct the **mg_php** engine to expect additional sections of an oversize return value to be held in global node **`^WORK($Job,0,`** where **`$Job`** is the M/Caché process ID returned by the M/Caché environment.

M/Caché procedure:

```
MyFun()    ; Return an oversize variable
           Set result=”first section ……”
           Set ^WORKJ($Job,0,extra,1)=”second section ……”
           Set ^WORKJ($Job,0,extra,2))=”third section ……”
           Set oversize=1
           Quit result
           ;
```

Equivalent PHP function:

```
result = m_proc(“MyFun^MyRoutine”);
```


Example 4 (Pass an oversize PHP array node to M/Caché):

M/Caché procedure:

```
MyFun(array) ; Accept an oversize array node
             Set s1=$Get(array("key to long string")
             Set s2=$Get(array("key to long string",extra,1)
             Set s3=$Get(array("key to long string",extra,2)
             Quit 1
             ;
```

Equivalent PHP function:

```
$a = array()
$a["key to long string"] = "large value …….";
result = m_proc("MyFun^MyRoutine", $a);
```

Example 4 (Pass an oversize M/Caché array node to PHP):

M/Caché procedure:

```
MyFun(array) ; Accept an oversize array node
             Set array("key to long string")="Section 1"
             Set array("key to long string",extra,1"="Section 2"
             Set array("key to long string",extra,2) ")="Section 3"
             Quit 1
             ;
```

Equivalent PHP function:

```
$a = array()
result = m_proc("MyFun^MyRoutine", $a);
```


5.7 Handling large PHP arrays in M/Caché

```
m_set_storage_mode(<mode>);
```

In addition to imposing limits on the maximum length of string that can be used, M/Caché also limits the amount of memory (or *partition* space) that each process can use. PHP, on the other hand, does not impose such limits but must, of course, work within the system limits imposed by the hosting computer. If large PHP arrays are sent to M/Caché, it may be necessary to specify that these arrays be held in a permanent storage within the M/Caché environment (i.e. a workfile) rather than in memory. The amount of memory that M/Caché allows for each process will depend on individual configurations.

The '**m_set_storage_mode'** function gives **mg_php** software some control over how array data is projected to the M/Caché environment.

Mode 0

```
m_set_storage_mode(0);
```

This is the default. PHP arrays are projected into (and out of) the M/Caché environment as simple memory-based arrays.

Mode 1

```
m_set_storage_mode(1);
```

PHP arrays are projected into (and out of) the M/Caché environment as equivalently structured global arrays (in permanent storage). These globals are structured as follows:

```
^WORKJ($Job,argn,
```

Where:

$Job – The M/Caché process ID (supplied by the M/Caché environment).
argn - The argument number in the function call.

Example 1 (Pass a PHP array into M/Caché global storage):

M/Caché procedure:

```
MyFun(array) ; Process an array held in a global
             Set key=""
             For  Set key=$Order(^WORKJ($Job,1,key)) Quit:key=""  Do
             . Set data=$Get(^WORKJ($Job,1,key))
             . Quit
             Quit 1
             ;
```

Equivalent PHP function:

```
$a = array();
m_set_storage_mode(1);
$a["key 1"]="value 1";
$a["key 2"]="value 2";
result = m_proc("MyFun^MyRoutine", $a);
```

Example 2 (Pass a M/Caché array held in global storage to PHP):

M/Caché procedure:

```
MyFun(array) ; Process an array held in a global
             Set ^WORKJ($Job,1,"key 1")="Value 1"
             Set ^WORKJ($Job,1,"key 2")="Value 2"
             Set ^WORKJ($Job,1,"key to long string")="Section 1"
             Set ^WORKJ($Job,1,"key to long string",extra,1)="Section 2"
             Set ^WORKJ($Job,1,"key to long string",extra,2)="Section 3"
             Quit 1
             ;
```

Equivalent PHP function (default server):

```
$a = array();
m_set_storage_mode(1);
result = m_proc("MyFun^MyRoutine", $a);
```

Equivalent PHP function (named server):

```
$a = array();
m_set_storage_mode(1);
result = m_proc("MyCachéServer", "MyFun^MyRoutine", $a);
```

Scope

The 'm_set_storage_mode' function operates on a per-page basis. The default mode (0) will be restored on calling further PHP pages.

5.8 Modifying the default M/Caché Server

```
m_set_server([server_name]);
```

You can specify the M/Caché server on a per-page basis using the '**m_set_server'** function.

For example:

```
m_set_server("MyCachéServer");
```

All **mg_php** function calls in the page will now target M/Caché server: `MyCachéServer`.

Scope

The **'m_set_server'** function operates on a per-page basis.

5.9 Modifying the default M/Caché UCI/NameSpace

```
m_set_uci([uci_name]);
```

You can specify the M/Caché server on a per-page basis using the '**m_set_uci'** function.

For example:

```
m_set_uci("MyUCI");
```

All **mg_php** function calls in the page will now target UCI: `MyUCI`.

Scope

The **'m_set_uci'** function operates on a per-page basis.

6 License

http://www.mgateway.com
Email: cmunt@mgateway.com