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

mg_php

M/Gateway Developments Ltd.
Chris Munt

Revision History:

13 June 2019
17 February 2020
12 January 2021
18 February 2021
14 March 2021

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** (*The M support routines*).
- A network configuration to bind the former two elements together.

The *M support routines* are required for:

- Network based access to databases.

Two M routines need to be installed **(%zmgsi** and **%zmgsis**). These can be found in the *Service Integration Gateway* (**mgsi**) GitHub source code repository.

https://github.com/chrisemunt/mgsi

Note that it is not necessary to install the whole *Service Integration Gateway (SIG)*, just the two M routines held in that repository, unless of course you intend to connect **mg_php** to the database via the SIG

3.1 InterSystems Caché or IRIS

Log in to the %SYS Namespace and install the **zmgsi** routines held in **/isc/zmgsi_isc.ro**.

```
do $system.OBJ.Load("/isc/zmgsi_isc.ro","ck")
```

Change to your development Namespace and check the installation:

```
do ^%zmgsi

M/Gateway Developments Ltd - Service Integration Gateway
Version: 4.0; Revision 16 (11 February 2021)
```

3.2 YottaDB

The instructions given here assume a standard 'out of the box' installation of **YottaDB** (version 1.30) deployed in the following location:

```
/usr/local/lib/yottadb/r130
```

The primary default location for routines:

```
/root/.yottadb/r1.30_x86_64/r
```

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

```
/root/.yottadb/r1.30_x86_64/r
```

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

```
cd /usr/local/lib/yottadb/r130
./ydb
```

Check the installation:

```
do ^%zmgsi

M/Gateway Developments Ltd - Service Integration Gateway
Version: 4.0; Revision 16 (11 February 2021)
```

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

3.3 Setting up the network service: the DB Superserver

The default TCP server port for the DB Superserver (**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 PHP installation to reside on the same host as the database server.

For InterSystems DB Servers, the DB Superserver is started from the DB Server command prompt. For YottaDB the DB Superserver can either be started from the DB Server command prompt or managed by the ***xinetd*** service.

3.3.1 Starting the DB Superserver from the DB Server command prompt

For InterSystems DB Servers the DB Superserver should be started in the %SYS Namespace.

```
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 Starting YottaDB Superserver processes via xinetd

Instead of starting the DB Supersever from the DB Server command prompt, network connectivity to **YottaDB** can be managed via the **xinetd** service. First create the following launch script (called **zmgsi_ydb** here):

```
/usr/local/lib/yottadb/r130/zmgsi_ydb
```

Content:

```
#!/bin/bash
cd /usr/local/lib/yottadb/r130
export ydb_dir=/root/.yottadb
export ydb_dist=/usr/local/lib/yottadb/r130
export
ydb_routines="/root/.yottadb/r1.30_x86_64/o*(/root/.yottadb/r1.30_x86_6
4/r /root/.yottadb/r) /usr/local/lib/yottadb/r130/libyottadbutil.so"
export ydb_gbldir="/root/.yottadb/r1.30_x86_64/g/yottadb.gld"
$ydb_dist/ydb -r xinetd^%zmgsis
```

Note that you should, if necessary, modify the permissions on this file so that it is executable. For example:

```
chmod a=rx /usr/local/lib/yottadb/r130/zmgsi_ydb
```

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/r130/zmgsi_ydb
}
```

- Note: sample copies of **zmgsi_xinetd** and **zmgsi_ydb** are included in the /unix directory of the **mgsi** GitHub directory (https://github.com/chrisemunt/mgsi).

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 Globals

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

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

PHP Array:

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

Corresponding M 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 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.

A M 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 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 globals, data nodes within PHP arrays are unable to simultaneously point to data and further subscripts. For example, in M 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, 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 database from within the PHP programming environment. Functions are also supplied to allow you to directly call M extrinsic functions and M procedures that are capable of generating sections of HTML form data.

In a number of cases it will be noticed that some operations are defined as two functions. This is done partly to maintain backwards compatibility with previous versions and partly to make **mg_php** accessible to developers not familiar with M. For example, a function to determine if a database record is defined named **m_data** may seem logical to M programmers familiar with the M **$Data** function. However, the name **m_defined** for the same operation will be more accessible to those not familiar with M.

5.1 Connecting to the database

By default, and assuming TCP based connectivity is used, the **mg_php** methods will address M ***Host*** '**localhost'** listening on TCP port **7041**.

The ***Host*** can also be the M/Gateway ***Service Integration Gateway*** (**mgwsi**). By default, the ***Service Integration Gateway*** listens on TCP port **7040**.

mg_php can connect to the M database via one of three methods:

- Directly to M over TCP. In this context the ***Host*** is the M server.
- Indirectly to M via the ***Service Integration Gateway (SIG).*** In this context, the ***Host*** is the ***SIG***.
- Directly bound to a local installation of M via its system-level API.

The methods described in this section allow you to redefine the default host, and to address hosts other than the default. The mechanism for connecting to M via the API will also be described.

5.1.1 Modifying the default M Host

```
m_set_host(<netname>, <username>, <password>)
```

In addition to addressing the default host (localhost:7041), other M servers can be specified on a per-call basis as shown in this document's many examples. You can specify the M host to be used within an instance of the **mg_php** module using the '**m_set_host'** method.

For example:

```
m_set_host("MyCachéServer", 7041, "", "");
```

All **mg_php** method calls in the page will now target M server: `MyCachéServer` listening on TCP port 7041.

Scope

The name of the host is scoped in accordance with the instance of the **mg_php** module and server handle in use.

5.1.2 Modifying the default M NameSpace

```
m_set_uci(<uci>)
```

This method will change the Namespace associated with the current instance.

For example:

```
m_set_uci("USER");
```

5.1.3 Modifying the default M Server Configuration Name

This method is only relevant to **mg_php** installations connecting to M server via the M/Gateway **Service Integration Gateway (SIG)**.

```
m_set_server(<server_configuration_name>)
```

This method will change the Server Configuration Name associated with the current instance. The Server Configuration Name is the name assigned to the M configuration as defined in the **SIG**.

For example:

```
m_set_server("LOCAL");
```

5.1.4 Non-network based access to the database via its API

As an alternative to connecting to the database using TCP based connectivity, **mg_php** provides the option of high-performance embedded access to a local installation of the database via its API.

5.1.4.1 InterSystems Caché or IRIS.

Use the following functions to bind to the database API.

```
m_set_uci(<namespace>)
m_bind_server_api(<dbtype>, <path>, <username>, <password>, <envvars>,
                  <params>)
```

Where:

```
namespace:  Namespace.
dbtype:     Database type ('Cache' or 'IRIS').
path:       Path to database manager directory.
username:   Database username.
password:   Database password.
envvars:    List of required environment variables.
params:     Reserved for future use.
```

Example:

```
m_set_uci("USER")
$result = m_bind_server_api("IRIS", "/usr/iris20191/mgr",
                            "_SYSTEM", "SYS", "", "");
```

The bind function will return '1' for success and '0' for failure.

Before leaving your PHP application, it is good practice to gracefully release the binding to the database:

```
m_release_server_api()
```

Example:

```
m_release_server_api();
```

5.1.4.2 YottaDB

Use the following function to bind to the database API.

```
m_bind_server_api(<dbtype>, <path>, <username>, <password>, <envvars>,
                  <params>)
```

Where:

```
dbtype:     Database type ('YottaDB').
path:       Path to the YottaDB installation/library.
username:   Database username.
password:   Database password.
envvars:    List of required environment variables.
```

```
params:     Reserved for future use.
```

Example:

This assumes that the YottaDB installation is in: `/usr/local/lib/yottadb/r130`
This is where the **libyottadb.so** library is found.
Also, in this directory, as indicated in the environment variables, the YottaDB routine interface file resides (zmgsi.ci in this example). The interface file must contain the following lines:

```
sqlemg: ydb_string_t * sqlemg^%zmgsis(I:ydb_string_t*, I:ydb_string_t *,
I:ydb_string_t *)
sqlrow: ydb_string_t * sqlrow^%zmgsis(I:ydb_string_t*, I:ydb_string_t *,
I:ydb_string_t *)
sqldel: ydb_string_t * sqldel^%zmgsis(I:ydb_string_t*, I:ydb_string_t *)
ifc_zmgsis: ydb_string_t * ifc^%zmgsis(I:ydb_string_t*, I:ydb_string_t *,
I:ydb_string_t*)
```

Moving on to the PHP code for binding to the YottaDB database. Modify the values of these environment variables in accordance with your own YottaDB installation. Note that each line is terminated with a linefeed character, with a double linefeed at the end of the list.

```
$envvars = "";
$envvars = $envvars . "ydb_dir=/root/.yottadb\n";
$envvars = $envvars . "ydb_rel=r1.30_x86_64\n";
$envvars = $envvars . "ydb_gbldir=/root/.yottadb/r1.30_x86_64/g/yottadb.gld\n";
$envvars = $envvars .
"ydb_routines=/root/.yottadb/r1.30_x86_64/o*(/root/.yottadb/r1.30_x86_64/r
/root/.yottadb/r) /usr/local/lib/yottadb/r130/libyottadbutil.so\n"
$envvars = $envvars . "ydb_ci=/usr/local/lib/yottadb/r130/zmgsi.ci\n";
$envvars = $envvars . "\n";

$result = m_bind_server_api("YottaDB",
                            "/usr/local/lib/yottadb/r130",
                            "", "", envvars, "")
```

The bind function will return '1' for success and '0' for failure.

Before leaving your PHP application, it is good practice to gracefully release the binding to the database:

```
m_release_server_api()
```

Example:

```
m_release_server_api();
```

5.2 Direct access to the M database

The functions in this section give you direct access to the M commands for manipulating global data. A M 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.

For each of the functions defined in this section there is the option of defining a target configuration in the ***Service Integration Gateway*** (if used). For example, the function to set a global database record can be defined as:

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

5.2.1 Set a global node (m_set)

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

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

Example 1:

M command:

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

Equivalent PHP function:

```
m_set(“^Customer”, 1234, “Chris Munt”);
```

5.2.2 Retrieve the data from a global node (m_get)

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

Example 1:

M command:

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

Equivalent PHP function:

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

5.2.3 Check that a global node exists (m_defined or m_data)

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

Example 1:

M command:

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

Equivalent PHP function:

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

5.2.4 Delete a global node (m_delete or m_kill)

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

Example 1:

M command:

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

Equivalent PHP function:

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

5.2.5 Get the next record (subscript) from a global node (m_order)

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

Example 1:

M command:

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

Equivalent PHP function:

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

5.2.6 Get the previous (subscript) from a global node (m_previous)

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

Example 1:

M command:

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

Equivalent PHP function:

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

5.2.7 Increment the value of a global node (m_increment)

```
value = m_increment(<global> {, subscript(s) …}, <increment_value>)
```

Example 1:

M command:

```
Set value=$increment(^Global("counter"))
```

Equivalent PHP function:

```
value = m_increment("^Customer", "counter");
```

This will increment the value of global node **^Global("counter")**, by 1 and return the new value.

5.2.8 Merge a PHP array to a global (m_merge_to_db)

```
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 global will be deleted at the level specified within the merge function before the actual merge is performed.

Example 1:

M 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:

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

Example 2:

M 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:

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


Example 3 (Using the 'ks' option):

Existing M database:

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

M 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 will hold the following records:

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

Example 4 (Dealing with multi-dimensional arrays):

M 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 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”
```


5.2.9 Merge a global to a PHP array (m_merge_from_db).

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

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

Example 1:

M database:

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

M 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:

M database:

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

M 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):

M 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”
```

M 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”;
```

5.3 Direct access to M functions and procedures

M 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 functions and procedures within the PHP environment.

A note on terminology: M 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.

For each of the functions defined in this section there is the option of defining a target configuration in the ***Service Integration Gateway*** (if used). For example, the function to invoke a M function can be defined as:

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

5.3.1 Call a M extrinsic function

Arguments can be passed ***by value*** or ***by reference***.

5.3.1.1 Passing arguments by value (m_function or m_proc)

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

Example 1 (A simple function call):

M procedure:

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

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

Equivalent PHP function:

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

Example 2 (Another simple function call):

M procedure:

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

Equivalent PHP function:

```
cust = m_function(“GetCust^MyRoutine”, 1234);
```

5.3.1.2 Passing arguments by reference (m_proc_byref)

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

The '***byref'*** version of the ***m_proc*** function should be used when passing PHP arrays to M as input parameters and in all cases where the corresponding M code can modify input parameters.

Example 1 (Passing arguments by reference):

M 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'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 2 (Passing an array from PHP to M):

M 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 3 (Passing an array from M to PHP):

M 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 4 (Using a M function to modify a PHP array):

M 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);
```

5.3.1.3 Passing complex argument lists (m_proc_ex)

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

This function provides a means by which complex arguments (arrays etc..) can be passed between M and PHP but without the need to modify PHP input arguments. The return value is always a numeric array, each element (1->n) of which represents a possibly modified input value. Element 1 corresponds to the first input parameter, element 2 the second etc … Element zero will contain the value returned from the corresponding M function. If an input value is not modified by the M code, then it will be returned unchanged.

Example:

Consider the following M function:

```
test(in, out) ; test function
   set in="updated string"
   for i=1:1:5 set out(i)="record #"_i
   quit "return value"
   ;
```

The corresponding PHP code:

```
<?php
   $in = "input string";
   $result = m_proc_ex('test^cm', $in, "");
   print("\r\nvar_dump for result ...\r\n");
   var_dump($result);
?>
```

The output from this code …

```
var_dump for result ...
array(3) {
  [1]=>
  string(14) "updated string"
  [2]=>
  array(5) {
    [1]=>
    string(9) "record #1"
    [2]=>
    string(9) "record #2"
    [3]=>
    string(9) "record #3"
    [4]=>
    string(9) "record #4"
    [5]=>
    string(9) "record #5"
  }
  [0]=>
  string(12) "return value"
}
```

As you can see, the return value is held in **$result[0]**, a modified value for **$in** in **$result[1]** and the array **($out**) in **$result[2].**

5.3.2 Return a block of HTML from a M function (m_html)

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

Example 1:

M procedure:

```
MyHtml      ; Return some HTML to PHP
```

```
            Write "<p>This text was returned from M"
            Write "<br>You can return as text much as you like …"
            Quit
            ;
```

Equivalent PHP function:

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

Example 2:

M procedure:

```
GetHTML(custID) ; Return some HTML to PHP
            Write "<p>This text was returned from M"
            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):

M 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);
```


5.4 Transaction Processing

M DB Servers implement Transaction Processing by means of the methods described in this section.

5.4.1 Start a Transaction

```
result = m_tstart()
```

On successful completion this method will return zero, or an error code on failure.

Example:

```
result = m_tstart()
```

5.4.2 Determine the Transaction Level

```
result = m_tlevel()
```

Transactions can be nested and this method will return the level of nesting. If no Transaction is active this method will return zero. Otherwise a positive integer will be returned to represent the current depth of Transaction nesting.

Example:

```
tlevel = m_tlevel()
```

5.4.3 Commit a Transaction

```
result = m_tcommit()
```

* On successful completion this method will return zero, or an error code on failure.

Example:

```
result = m_tcommit()
```

5.4.4 Rollback a Transaction

```
result = m_trollback()
```

On successful completion this method will return zero, or an error code on failure.

Example:

```
result = m_trollback()
```

5.5 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.

For each of the functions defined in this section there is the option of defining a target configuration in the ***Service Integration Gateway*** (if used). For example, the function to invoke a Caché method can be defined as:

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

5.5.1 Call a Caché method.

Arguments can be passed ***by value*** or ***by reference***.

5.5.1.1 Passing arguments by value (m_classmethod or m_method)

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

Example 1 (A simple method):

Caché method:

```
Class MyUtilities.MyClass Extends etc …

ClassMethod GetTime()
{
      ; Get the current date and time in M'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 (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);
```

5.5.1.2 Passing arguments by reference (m_method_byref)

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

The '***byref'*** version of the ***m_method*** function should be used when passing PHP arrays to Caché as input parameters and in all cases where the corresponding Caché code can modify input parameters.

Example 1 (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 2 (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 3 (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 4 (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);
```


5.5.2 Return a block of HTML from a Caché method (m_html_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);
```

5.6 Direct access to PHP and M 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 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>
```


5.7 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 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.

5.8 Handling PHP strings that exceed the maximum size allowed under M

M-based systems impose 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 and PHP. Oversize strings are broken up into individual sections in M as follows:

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

For example, take a string that exceeds the maximum length allowed in M 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 and vice versa. The PHP software is unaffected by these transformations.

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

M 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 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 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 process ID returned by the M environment.

M 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):

M 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 array node to PHP):

M 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);
```

5.9 Handling large PHP arrays in M

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

In addition to imposing limits on the maximum length of string that can be used, M 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, it may be necessary to specify that these arrays be held in a permanent storage within the M environment (i.e. a workfile) rather than in memory. The amount of memory that M 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 environment.

Mode 0

```
m_set_storage_mode(0);
```

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

Mode 1

```
m_set_storage_mode(1);
```

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

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

Where:

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

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

M 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 array held in global storage to PHP):

M 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.

5.10 Request timeout and failure

Functions described here provide some control over the amount of time that **mg_php** functions will wait for a response from M and the action the ***Service Integration Gateway*** (if used) will take on request timeout or failure.

5.10.1 Modifying the Server Response Timeout (m_set_timeout)

```
m_set_timeout(<timeout_secs>)
```

Example (Set the timeout to 5 minutes):

```
m_set_timeout(300);
```

Calling this function with a value of zero will reset the timeout value to the default of 60 seconds.

5.10.2 Modifying the retry and failover mechanism (m_set_no_retry)

```
m_set_no_retry(<flag>)
```

The 'no_retry' flag, if set, will suppress all request retry attempts and the failover mechanism in the ***Service Integration Gateway*** (if used).

Example (Disable the retry/failover mechanism):

```
m_set_no_retry(1);
```

5.11 Request logging facility (m_set_log_level)

```
m_set_log_level(<log_file>, <log_level>, <log_filter>)
```

Where:

- **log_file:** The name (and path to) the log file you wish to use. The default is c:/temp/m_php.log (or /tmp/m_php.log under UNIX).
- **log_level:** A set of characters to include one or more of the following:
  - **e -** log error conditions.
  - **f -** log all function calls (just the name of the function called).
  - **t -** log all transmissions between the PHP module and the Host.
- **log_filter:** A comma-separated list of functions that you wish the log directive to be active for. This should be left empty to active the log for all functions.

Example (limit logging to the 'm_proc_ex' and 'm_proc_byref' functions):

```
m_set_log_level("c:/temp/m_php.log", "et", "m_proc_ex,m_proc_byref");
```

This invocation will generate a log file similar to that shown below. For each function call, the process ID (**pid**), thread ID (**tid**), request no. (**req_no**) and function call no. (**fun_no**) will be the same so these values can be used to link request messages to the corresponding response payloads.

```
>>> Time: Sat Jan 11 16:19:34 2020; Build: 2.1.54 pid=24016;tid=24052;req_no=1;fun_no=3
    Logging Active
    Log File: c:/temp/m_php.log; Log Level: et; Log Filter: ,m_proc_ex,m_proc_byref,;
>>> Time: Sat Jan 11 16:19:34 2020; Build: 2.1.54 pid=24016;tid=24052;req_no=1;fun_no=8
    Transmission: Send to Host (size=74)
    PHPz^P^##0#0#0#2.1.54#0^X^00000\x0a\x015LOCAL\x0211MPROCX^%ZCMB14input argumemtQ0Y0
>>> Time: Sat Jan 11 16:19:34 2020; Build: 2.1.54 pid=24016;tid=24052;req_no=1;fun_no=8
    Transmission: Received from Host (size=120)
    0001occ\x0a\x0212return  value\x010\x010\x010B14updated  string\x110\x0911\x019record
#1\x0912\x019record #2\x0913\x019record #3\x0914\x019record #4\x0915\x019record #5\x190
```

6 License

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