The native PHP client

kbmMW supports a number of native clients. One of then is a PHP extension which allows PHP code to reach a kbmMW based application server.

This document will show how to compile the extension, how to install it and how to use it.

Compiling the PHP extension

For Win32

The Win32 extension is in the form of a DLL. The DLL is usually delivered with the kbmMW PHP client install, but in case it needs recompilation the steps described in this section must be followed.

Prerequisites:

- Compiled kbmMW C client library.
  http://www.components4developers.com

- The kbmMW PHP client source
  http://www.components4developers.com

- PHP source installation v. 4.3.11 or newer.
  http://www.php.net/releases.php

- PHP related bind lib package for Win32.
  http://www.php.net/extra/bindlib_w32.zip

- PHP related Win32 build utilities
  http://www.php.net/extra/win32build.zip

- If you plan to compile PHP as an Apache module (not part of this documentation) you will also need the Apache source code.
  http://www.apache.org/dist/httpd/

- Microsoft Visual Studio .Net with C/C++ compiler installed or Visual C++ v6
  http://www.microsoft.com

Build procedure:

Please make sure to follow these steps very carefully!
Compiling PHP is always an experience of its own, specially on Windows.

1. Create a working directory where all files end up after extracting, e.g: `C:\work`.

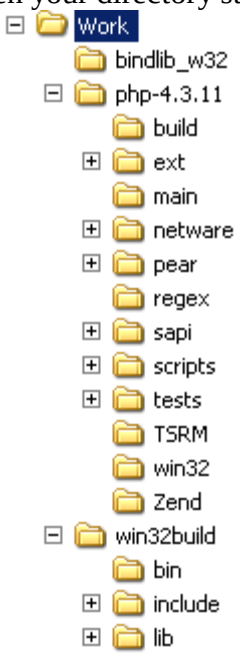
2. Unzip `win32build.zip` into the working directory (`C:\work`) including folders.

3. Create the directory `bindlib_w32` under your working directory (`C:\work`) and unzip `bindlib_w32.zip` into it (`C:\work\bindlib_w32`) including folders.

4. Extract the downloaded PHP source code into your working directory (`C:\work`).

5. Build the libraries you are going to need (or download the binaries if available) and place the headers and libs in the `C:\work\win32build\include` and `C:\work\win32build\lib` directories, respectively. This includes the kbmMW C client library (*.lib) and include files (*.h).

6. Then your directory structure should look similar to this:



7. Make a directory: `c:\usr\local\lib` and copy bison.simple from `C:\work\win32build\bin` to `c:\usr\local\lib`

8. Configure MS VS.Net or VC++ to know the appropriate paths for include, library and executable files.
   a) Select Tools/Options. In Visual C++ select Directories tab. In VS.Net select the Projects/VC++ Directories folder.
   b) Add to the executable files section the path: `c:\work\win32build\bin`
   c) Add to the include files section the paths:
   ```
   c:\work\win32build\include
   c:\work\php*\main
   c:\work\php*\TSRM
   c:\work\php*\zend
   c:\work\php*
   ```
   where php* is the proper name of the php directory.
   d) Add to the libraries files section the path: `c:\work\win32build\lib`

9. In VC++ or VS.Net open the workspace bindlib.dsw in c:\work\bindlib_w32. VS.Net will ask if its ok to update the workspace file from v6 to current version. Accept.
   Set active configuration to either 'bindlib – Win32 Debug' or 'bindlib – Win32 Release'.
   Select Rebuild or Rebuild All. If its asking to save the solution file, just accept.

10. Copy the newly compiled `resolv.lib` file from either the `c:\work\bindlib_w32\Release` or `c:\work\bindlib_w32\Debug` directory (depending on which configuration you selected) to `c:\work\win32build\lib` overwriting any file with the same name there.

11. In VC++ or VS.Net open the workspace `php4ts.dsw`, select the appropriate active configuration, either Release-TS or Debug-TS, and Rebuild All. If its asking to save the solution file, just accept.
    Some packages in the workspace may not compile without error, but don't worry as not all packages are needed for us to generate the kbmMW PHP extension DLL.
    After compile you should have a file called `php4ts.lib` in c:\work\php*\Release or c.\work\php*\Debug (depending on the active configuration you selected).
    Copy it to `c:\work\win32build\lib`.

12. In VC++ or VS.Net open the project `kbmmw.dsp` from the kbmMW PHP Client source directory. Allow VS.Net to convert the project if required.
    Select appropriate active configuration (should match the configuration chosen when compiling the `php4ts.dsw` workspace).
    Rebuild All. If its asking to save the solution file, just accept.

Now a DLL with the name `php_kbmmw.dll` should be placed in either Release_TS or Debug_TS (depending on which active configuration you selected).

This DLL is the ready to use kbmMW PHP dynamic extension for the PHP version it was compiled for. A general rule of thumb is that an extension can run on the same and higher levels of PHP as it was compiled with as long as it's the same major version number. I.e. the kbmMW PHP extension compiled for v. 4.3.11 can be used on PHP v. 4.4.x without problems.

Installation and configuration:

1. Copy the DLL to the `extentions` directory of the Win32 PHP runtime installation. (Notice that this is different from the PHP source installation that we installed to compile the kbmMW PHP extension DLL.)

2. If VS.Net was used for compiling the kbmMW extension DLL, make sure you have `msvcr71.dll` in your `system32` folder. If it is missing, copy it from your VS.Net installation.

3. Open `php.ini` from the Win32 PHP runtime installation, in a text editor like notepad. If php.ini do not exist, you need to configure PHP first.

4. Locate the text: Windows Extensions in the file. This should lead you to a place with lots of lines (possibly commented) like: **`extension=something.dll`**

5. Insert a new line: **`extension=php_kbmmw.dll`**

6. If PHP is used within a webserver (IIS, Apache etc) restart those.

Remember to configure php.ini to minimum specify extension_dir to point on the appropriate directory in which the dll extensions are placed.

Generally php.ini should be placed in your Windows directory, although other configurations may be recommended by your webserver provider, or by PHP.org.

For Linux/Unix

The Linux extension is compiled into the PHP executable and for that reason the steps described in this section must be followed.

Prerequisites:

- PHP source installation v. 4.3.11 or newer.
  http://www.php.net/releases.php

- Full C development SDK matching your system (GCC primarly supported by PHP).
  http://gcc.gnu.org/

- libtool v. 1.4.x (not 1.5.x as that wont work with the standard PHP building procedures. Please notice that PHP automatically generates a local symbolic link to /usr/local/bin/libtool which is placed in PHP's root directory.)
  http://www.gnu.org/software/libtool/

- automake v. 1.5
  http://sources.redhat.com/automake/

- Autoconf v. 2.13 (earlier versions have problems with cleaning up its cache properly. To circumvent, delete the contents of the autom4te.cache directory before doing buildconf.)
  http://www.gnu.org/software/autoconf/

- kbmMW pure C-client.
  The include files must be placed in /usr/include/kbmmw and the compiled library file libkbmMWClient.a must be placed in /usr/lib/kbmmw
  http://www.components4developers.com

- kbmMW PHP extension. This must be placed in a directory named kbmmw in the PHP subdirectory named ext
  http://www.components4developers.com

Build procedure:

1. When these prerequisites are in place, configuration of PHP with kbmMW can begin. Usually it is required to be root to succeed with the PHP configuration.

2. First optionally clean the files from the `autom4te.cache` directory in the PHP install directory.

3. **./buildconf –force**
   Will generate the configuration scripts needed by the build process.

4. **./configure --enable-kbmmw**
   Will generate the compile scripts needed to compile PHP with kbmMW extension support. More extensions can be given on the command line.

5. **make**
   Will compile PHP and the required extensions (in our case including the kbmMW extension).

After PHP has been compiled, kbmMW will now be an integral part of it. Copy the compiled PHP executable to the place where its getting executed (depends on installation).

The Demo application explained

```
<html>
<body>
<?php
 echo "<p>";
 echo "This sample will call a kbmMW application server to obtain inventory<br>";
 echo "information about the server using STANDARD streamformat,<br>";
 echo "VerifyTransfer=true and StringConversion=mwscFixed.<br>";
 echo "<br>";
 echo "The kbmMW PHP client is a PHP extension which can be loaded at runtime by the PHP<br>";
 echo "framework. The extension is written in pure C and is portable to numerous platforms.<br>";
 echo "<br>";
 echo "</p>";
```

Standard HTML for embedded PHP code producing some description about the sample in the users browser.

```
 // Allocate a transport and a client.
 $transport = kbmMW_New_ClientTransport();
 $client = kbmMW_New_Client($transport);

 // Connect the client to a kbmMW application server using the given transport.
 kbmMW_Client_Connect($client,"127.0.0.1",3000);
```

Allocate a new transport object and a new simple client object which use the transport object. Connect the client to localhost on port 3000.

```
 // Send a request for
 // Service:        KBMMW_INVENTORY
 // ServiceVersion: KBMMW_1.0
 // Function:       LIST
 // And no arguments.
 // Any number of arguments could be passed, including simple types and arrays.
 // Afterwards print out the result.
 $res = kbmMW_Client_SendRequest($client,"KBMMW_INVENTORY","KBMMW_1.0","LIST");
 echo "<p>$res</p>";
```

Send a request to the application server. SendRequest can take a variable number of arguments and returns a result in the $res variable, which is then displayed.

```
 // Disconnect the PHP client from the kbmMW application server.
 kbmMW_Client_Disconnect($client);
```

Disconnect from the application server. Its important to call this before the client is disposed of.

```
 // Dispose of the client and transport ressources. Its not mandatory, as PHP will
 // garbage collect variables going out of scope automatically.
 kbmMW_Dispose_Client($client);
 kbmMW_Dispose_ClientTransport($transport);
```

Dispose of the client and transport objects. Its not a requirement (but recommended) to do this in PHP, as PHP automatically keeps track of scope of ressources and lets the kbmMW extention automatically dispose of them when they go out of scope.

```
?>
</body>
```

```
</html>
```

Start a kbmMW Demo application server on your local machine, listening on port 3000 and with VerifyTransfer set to true.

Put the testkbmmw.php demo script into a directory accessible by a browser via your web server, and in the browser refer to it:

Eg. http://localhost/testkbmmw.php

This should display some text in your browser, including a string containing the names and versions of all services supported by your application server.

Exception handling

All exception handling is using the standard PHP/Zend error management routines. In addition the client object can be queried for status from the application server. Please refer to the documentation of the client object.

Standard error codes (provided by the application server via the client object):

KBMMW_ERR_EXCEPTION	100000

Provided if an undefined exception has occurred in a call to a kbmMW based application server.
An error message explaining the problem is usually provided.

KBMMW_ERR_ABORT	100001

Provided if an abort exception has occurred in a call to a kbmMW based application server.
An error message explaining the reason is usually provided.

KBMMW_ERR_SERVICENOTAVAIL	100100

Provided if a call has been made to an unavailable service.
An error message explaining the problem is usually provided.

KBMMW_ERR_FUNCNOTAVAIL	100101

Provided if an unknown function has been called on a kbmMW based application server.
An error message explaining the problem is usually provided.

KBMMW_ERR_AUTHFAILED	100200

Provided if the call to the application server was not authorized.
An error message explaining the reason is optionally provided.

KBMMW_ERR_MESSAGETYPE	108000

Provided if an unknown message has been received from the kbmMW based application server.
Currently not in use.

KBMMW_SYS_REDIRECT	110000

Provided if the kbmMW based application server requests the client to disconnect and reconnect to another application server. The connection string to the new application server is in the error message.

Variant support

PHP already supports untyped variables, which in many ways function like variants known in the MS world. However the possible native types are limited to a few basic types and the custom ressource type.

Conversion between PHP's untyped values and the typed variants used in the application server is handled automatically.

To be able to express types that are not directly supported by PHP, some variant routines has been provided.

```
bool kbmMW_Dispose_Variant(ressource aVariant)
```

Dispose the variant object given by aVariant.

```
ressource kbmMW_New_Variant(long aType)
```

Generate a new variant object of the given type, having an undefined content. Do not obtain values from the variant until a value has been set by one of the setters.
The variant should optionally be disposed of after use by using kbmMW_Dispose_Variant.

```
bool kbmMW_Variant_SetAsNull(ressource aVariant)
```

Set an existing variant variable to the NULL value. If it already contained other variant data, those will be automatically disposed of.
If the operation was successful, true is returned, else a Zend error is thrown.

```
bool kbmMW_Variant_SetAsString(ressource aVariant, string aString)
```

Set an existing variant variable to contain a string value. If it already contained other variant data, those will be automatically disposed of.
The variant will have type kbmMWvarString.
If the operation was successful, true is returned, else a Zend error is thrown.

Eg.

```
$aVariant=kbmMW_New_Variant(1);
kbmMW_Variant_SetAsString($aVariant,"This is a string");
…
kbmMW_Dispose_Variant($aVariant);
```

```
bool kbmMW_Variant_SetAsOleStr(ressource aVariant, string aString);
```

Same as kbmMW_Variant_SetAsString except that the variant type is set to be kbmMWvarOleStr which can contain 16 bit Unicode/Widestring data. If it already contained other variant data, those will be automatically disposed of.
If the operation was successful, true is returned, else a Zend error is thrown.

```
bool kbmMW_Variant_SetAsInt(ressource aVariant, long aValue);
```

Set the variant variable to an integer value (max 32 bit) . The variant will have type kbmMWvarInt. If it already contained other variant data, those will be automatically disposed of.
If the operation was successful, true is returned, else a Zend error is thrown.

```
bool kbmMW_Variant_SetAsByte(ressource aVariant, long aValue);
```

Set the variant variable to a byte value (8 bit). The variant will have type kbmMWvarByte. If it already contained other variant data, those will be automatically disposed of.
If the operation was successful, true is returned, else a Zend error is thrown.

```
bool kbmMW_Variant_SetAsBool(ressource aVariant, bool aValue);
```

Set the variant variable to a boolean value. The variant will have type kbmMWvarBoolean. If it already contained other variant data, those will be automatically disposed of.
If the operation was successful, true is returned, else a Zend error is thrown.

```
bool kbmMW_Variant_SetAsShortInt(ressource aVariant, long aValue);
```

Set the variant variable to a short int (16 bit) value. The variant will have type kbmMWvarShortInt. If it already contained other variant data, those will be automatically disposed of.
If the operation was successful, true is returned, else a Zend error is thrown.

```
bool kbmMW_Variant_SetAsSmallInt(ressource aVariant, long aValue);
```

Set the variant variable to a small int (16 bit) value. The variant will have type kbmMWvarSmallInt. If it already contained other variant data, those will be automatically disposed of.
If the operation was successful, true is returned, else a Zend error is thrown.

```
bool kbmMW_Variant_SetAsLongWord(ressource aVariant, long aValue);
```

Set the variant variable to a long word (32 bit signed) value. The variant will have type kbmMWvarLongWord. If it already contained other variant data, those will be automatically disposed of.
If the operation was successful, true is returned, else a Zend error is thrown.

```
bool kbmMW_Variant_SetAsDate(ressource aVariant, long aValue);
```

Set the variant variable to a date/time value. The variant will have type kbmMWvarDate. If it already contained other variant data, those will be automatically disposed of.
If the operation was successful, true is returned, else a Zend error is thrown.
The date/time value is a Unix style long value indicating an offset compared to an epoch. Usually Unix style is used which is normally defined as number of seconds since Jan 1st 1970 at 0 hours.

```
bool kbmMW_Variant_SetAsSingle(ressource aVariant, double aValue);
```

Set the variant variable to a floating point single value. The variant will have type kbmMWvarSingle. If it already contained other variant data, those will be automatically disposed of.
If the operation was successful, true is returned, else a Zend error is thrown.

```
bool kbmMW_Variant_SetAsDouble(ressource aVariant, double aValue);
```

Set the variant variable to a floating point double value. The variant will have type kbmMWvarDouble. If it already contained other variant data, those will be automatically disposed of.
If the operation was successful, true is returned, else a Zend error is thrown.

```
bool kbmMW_Variant_SetAsDecimal(ressource aVariant, double aValue);
```

Set the variant variable to a decimal value. The variant will have type kbmMWvarDecimal. If it already contained other variant data, those will be automatically disposed of.
If the operation was successful, true is returned, else a Zend error is thrown.

```
bool kbmMW_Variant_SetAsCurrency(ressource aVariant, double aValue);
```

Set the variant variable to a currency value. The variant will have type kbmMWvarCurrency. If it already contained other variant data, those will be automatically disposed of.
If the operation was successful, true is returned, else a Zend error is thrown.

```
bool kbmMW_Variant_SetAsBinary(ressource aVariant, string aValue);
```

Set the variant variable to a binary value. The variant will have type kbmMWvarBinary. If it already contained other variant data, those will be automatically disposed of.
As PHP strings can contain all sorts of characters, they work well holding binary data.
If the operation was successful, true is returned, else a Zend error is thrown.

```
bool kbmMW_Variant_GetAsNull(ressource aVariant)
```

Returns true if the variant is NULL, else false. This function can't be used directly on a variant array, but only on its elements.

```
string kbmMW_Variant_GetAsString(ressource aVariant)
```

Returns the contents of the variant as a string.
If an error occurs (like missing/wrong arguments or invalid conversion), a Zend error is thrown.

This function can't be used directly on a variant array, but only on its elements.

If the variant type is kbmMWvarBoolean, the appropriate string "TRUE" or "FALSE" will be returned.

Valid conversions are from the following variant types:
`kbmMWvarNull, kbmMWvarBoolean, kbmMWvarByte, kbmMWvarShortInt, kbmMWvarSmallInt, kbmMWvarInteger, kbmMWvarLongWord, kbmMWvarSingle, kbmMWvarDouble, kbmMWvarDecimal, kbmMWvarCurrency, kbmMWvarString, kbmMWvarOleStr (no Unicode conversion done!)`

Eg.

```
$aVariant=kbmMW_Variant_GetAsString($variant);
```

```
string kbmMW_Variant_GetAsOleStr(ressource aVariant);
```

Same as kbmMW_Variant_GetAsString.

```
long kbmMW_Variant_GetAsInt(ressource aVariant);
```

Returns the contents of the variant as an integer value.
If an error occurs (like missing/wrong arguments or invalid conversion), a Zend error is thrown.

This function can't be used directly on a variant array, but only on its elements.

```
long kbmMW_Variant_GetAsByte(ressource aVariant);
```

Returns the contents of the variant as a byte (8 bit) value.
If an error occurs (like missing/wrong arguments or invalid conversion), a Zend error is thrown.

This function can't be used directly on a variant array, but only on its elements.

```
bool kbmMW_Variant_GetAsBool(ressource aVariant);
```

Return the contents of the variant as a boolean value.
If an error occurs (like missing/wrong arguments or invalid conversion), a Zend error is thrown.

This function can't be used directly on a variant array, but only on its elements.

```
long kbmMW_Variant_GetAsShortInt(ressource aVariant);
```

Returns the contents of the variant as a short int value.
If an error occurs (like missing/wrong arguments or invalid conversion), a Zend error is thrown.

This function can't be used directly on a variant array, but only on its elements.

```
long kbmMW_Variant_GetAsSmallInt(ressource aVariant);
```

Returns the contents of the variant as a small int value.
If an error occurs (like missing/wrong arguments or invalid conversion), a Zend error is thrown.

This function can't be used directly on a variant array, but only on its elements.

```
long kbmMW_Variant_GetAsLongWord(ressource aVariant);
```

Returns the contents of the variant as a long word.
If an error occurs (like missing/wrong arguments or invalid conversion), a Zend error is thrown.

This function can't be used directly on a variant array, but only on its elements.

```
long kbmMW_Variant_GetAsDate(ressource aVariant);
```

Returns the contents of the variant as a date/time value since epoch.
If an error occurs (like missing/wrong arguments or invalid conversion), a Zend error is thrown.

This function can't be used directly on a variant array, but only on its elements.

The date/time value is a Unix style long value indicating an offset compared to an epoch. Usually Unix style is used which is normally defined as number of seconds since Jan 1st 1970 at 0 hours.

```
double kbmMW_Variant_GetAsSingle(ressource aVariant);
```

Returns the contents of the variant as a single floating point value.
If an error occurs (like missing/wrong arguments or invalid conversion), a Zend error is thrown.

This function can't be used directly on a variant array, but only on its elements.

```
double kbmMW_Variant_GetAsDouble(ressource aVariant);
```

Returns the contents of the variant as a double floating point value.
If an error occurs (like missing/wrong arguments or invalid conversion), a Zend error is thrown.

This function can't be used directly on a variant array, but only on its elements.

```
double kbmMW_Variant_GetAsDecimal(ressource aVariant);
```

Returns the contents of the variant as a decimal floating point value.
If an error occurs (like missing/wrong arguments or invalid conversion), a Zend error is thrown.

This function can't be used directly on a variant array, but only on its elements.

```
double kbmMW_Variant_GetAsCurrency(ressource aVariant);
```

Returns the contents of the variant as a currency floating point value.
If an error occurs (like missing/wrong arguments or invalid conversion), a Zend error is thrown.

This function can't be used directly on a variant array, but only on its elements.

```
string kbmMW_Variant_GetAsBinary(ressource aVariant);
```

Returns the contents of the variant as a binary value string.
If an error occurs (like missing/wrong arguments or invalid conversion), a Zend error is thrown.

This function can't be used directly on a variant array, but only on its elements.

Client transport

The client transports responsibility is to manage a connection to a kbmMW based application server which is then used by the kbmMW PHP simple client.

Its designed to use a standard Posix compliant TCP/IP stack as the communication layer. This should make it compatible with most TCP/IP libraries for most platforms.

```
ressource  kbmMW_New_ClientTransport(void);
```

Allocates a new client transport object. The object should optionally be deallocated after use by kbmMW_Dispose_ClientTransport. No connections are being created at this stage.

```
void kbmMW_Dispose_ClientTransport(ressource aTransport);
```

Deallocates a client transport object. Its important to disconnect the client transport from the application server by using the kbmMW_ClientTransport_Disconnect or kbmMW_Client_Disconnect functions before disposing the object in case a connection has been established.

```
bool kbmMW_ClientTransport_Connect(ressource aTransport,
                                   string aHost, int aPort);
```

Connects to the given host on the given port number if possible. Its important later on to disconnect the client transport from the application server by using the kbmMW_ClientTransport_Disconnect or kbmMW_Client_Disconnect functions.
A Zend error is thrown if the arguments are invalid.
If the operation was successful, true is returned, else false.

```
bool kbmMW_ClientTransport_Disonnect(ressource aTransport);
```

Disconnects the client transport. A Zend error is thrown if the argument is invalid.
If the operation was successful, true is returned.

```
bool kbmMW_ClientTransport_IsConnected(ressource aTransport);
```

Returns true if the clienttransport is connected to an application server, else false.

Client

The clients responsibility is to provide an interface to the developer allowing requests to be made against a kbmMW based application server, and returned responses to be handled.

The client itself do not contain the actual transport layer. For that purpose the client transport is used.

```
ressource kbmMW_New_Client(ressource aTransport);
```

Allocates a new kbmMW_Client 'object'. An already allocated kbmMW_ClientTransport 'object' must be provided. The allocated TkbmMWSimpleClient should optionally be disposed using kbmMW_Dispose_Client after use.

```
bool kbmMW_Dispose_Client(ressource aClient);
```

Deallocates a kbmMW_Client 'object'. Notice that disposing the client object do not automatically disconnect the client transport from the application server, if it is connected. Hence it's good practice to call kbmMW_Client_Disconnect before disposing the client object, unless the client transport will be reused by other kbmMW_Client instances later on.

```
bool kbmMW_Client_Connect(ressource aClient, string aHost, long aPort);
```

Connects to a specified application server via the client transport provided when allocating the simple client object.

aClient is the simple client instance.
aHost is an IP address or internet server address to the kbmMW application server.
aPort is the port number that the kbmMW application server is listening on.

Returns true if connection succeeded, else false.

It is important to disconnect the client if already connected, before trying to connect again.

Eg.

```
$client=kbmMW_New_Client($aClientTransport);
if (!kbmMW_Client_Connect($client,"127.0.0.1",3000)) {
   echo "<p>Connection failed</p>";
}
else {
  // Do the call.
…
…
}

kbmMW_Client_Disconnect($client);
kbmMW_Dispose_Client($client); // Optional
```

```
bool kbmMW_Client_Disconnect(ressource aClient);
```

Disconnect a client from the application server.

aCliente is the simple client instance that is to disconnect from the app. server..
Returns true if connection succeeded, else false.

See Connect for an example.

```
result kbmMW_Client_SendRequest(ressource aClient,
                                string aServiceName,
                                string aServiceVersion,
                                string aFunction
                                [, arg0, …, argN ]);
```

Sends request to the application server. The client must already be connected to call this function.

aClient is the client instance to use to send the request.
aServiceName is a string identifying the name of the service to call.
aServiceVersion is a string identifying the version of the service to call.
aFunction is a string identifying the name of the function to call.

Following an optional number of arguments can be provided. The arguments can include arrays.

The function returns a native PHP value which can also be an array.

The SendRequest call will either result in a returned result or a Zend error message.

If the call succeeded (no Zend error message was thrown), then the actual status of the call can be checked using the kbmMW_Client_IsError, kbmMW_Client_IsOk and kbmMW_Client_IsWarning functions. Its also possible to probe the error/exception string and code by using kbmMW_Client_GetStatusCode and kbmMW_Client_GetStatusText.

An application server can return several error codes. Please check the error handling section for more information.

Eg.

```
$client=kbmMW_New_Client($aClientTransport);
if (!kbmMW_Client_Connect($client,”127.0.0.1”,3000)) {
   echo “<p>Connection failed.</p>”;
}

// Make the call.
$result=kbmMW_Client_SendRequest($client,
                                 ”KBMMW_INVENTORY”,
                                 ”kbmMW_1.0”,
                                 ”LIST”);
// Check for server exception.
if (kbmMW_Client_IsOK($client)) {

   // Is ok. Display result.
   Echo “<p>Result is:$result</p>”;
}
else {
   $errorcode=kbmMW_Client_GetStatusCode($client);
   $errormsg=kbmMW_Client_GetStatusText($client);
   echo “<p>Server signals exception. Errorcode=$errorcode, Text=$errormsg</p>”
}

kbmMW_Client_Disconnect($client);
kbmMW_Dispose_Client($client); // Optional.
```

Its possible to send a datastream/binary string with the call to the server in the following way:

```
kbmMW_Client_SetRequestStream($client,”some string that can be binary”);
```

Similarly one can receive a stream as a result from the application server by accessing in the following way:

```
$somestring=kbmMW_GetResponseStream($client);
```

```
ressource kbmMW_Client_SendRequestEx(ressource aClient,
                                     string aServiceName,
                                     string aServiceVersion,
                                     long aStateID,
                                     string aFunction
                                     [, arg0, …, argN ]);
```

Same as SendRequest except that its possible to provide a specific state identifier which do not have to be the same as the one stored within the client object itself.

```
bool kbmMW_Client_IsOK(ressource aClient);
```

Returns true if the last call to the application server resulted in no errors or exceptions. Else returns false.

```
bool kbmMW_Client_IsError(ressource aClient);
```

Returns true if an error exception has been raised on the client after last call to server. Else returns false.

```
bool kbmMW_Client_IsWarning(ressource aClient);
```

Returns true if a warning exception has been raised on the client after last call to server. Else returns false.

```
long kbmMW_Client_GetStatusCode(ressource aClient);
```

Returns the status code provided from the last call to the application server. If no error or warning was raised, 0 is returned. See the Exception section for a description of the predefined error codes a kbmMW application server can return (in addition to 0).

```
string kbmMW_Client_GetStatusText(ressource aClient);
```

Returns the status text provided from the last call to the application server. If no error or warning was raised, the string “OK” is returned.

Eg.

```
$s:=kbmMW_Client_GetStatusText(client);
echo “<p>Last status from server was $s</p>”;
```

```
long kbmMW_Client_GetStateID(ressource aClient);
```

Provides access to a state identifier. It will be –1 if the last call to the application server was a stateless call…ie. the service called is a stateless service.
In case a stateful service was called, kbmMW_Client_GetStateID will return a unique value which should be remembered by the caller. On later calls, the specific stateful service (currently 'owned' by the client) can be accessed by providing the state identifier. Its the responsibility of the developer to remember state id's returned from the application server and later to release the given state id's (except for state id –1).

```
long kbmMW_Client_SetStateID(ressource aClient, long aStateID);
```

Prepare the client to use the given state ID on the next call to the application server.
As an alternative, one can also choose to provide the state ID in a call using kbmMW_Client_SendRequestEx.
Returns the previous state ID of the client.

```
string kbmMW_Client_GetUserName(ressource aClient);
```

Provides access to the current username which the client should use when contacting the application server. It can be set at any time using kbmMW_Client_SetUserName.

```
bool kbmMW_Client_SetUserName(ressource aClient, string aUserName);
```

Set the username used by the client on the next call to the application server.

Eg.

```
$client=kbmMW_New_Client($aClientTransport);
kbmMW_Client_SetUserName($client,"JENS HANSEN");

// Do the call
…
…

kbmMW_Dispose_Client($client); // Optional
```

```
string kbmMW_Client_GetPassword(ressource aClient);
```

Provides access to the current password which the client should use when contacting the application server. It can be set at any time using kbmMW_Client_SetPassword.

```
bool kbmMW_Client_SetPasword(ressource aClient, string aPassword);
```

Set the username used by the client on the next call to the application server.

Eg.

```
$client=kbmMW_New_Client($aClientTransport);
kbmMW_Client_SetPassword($client,"SOMEPASSWORD");

// Do the call
…
…

kbmMW_Dispose_Client($client); // Optional
```

```
string kbmMW_Client_GetToken(ressource aClient);
```

Provides access to the token returned from the application server, after an authentication process have happened where a username and password has been validated and accepted. As result of the validation, the application server usually returns a unique and temporary value, called a token, which identifies the current client/user session.

Until the client disconnects, the client should forward the provided Token instead of Username and Password, by remembering the Token value returned as the result of a call to the application server, and later setting the Token property with the remembered value on subsequent calls to the application server. A client can choose to have multiple logins at the same time, in which case multple tokens should be remembered.
For a client to log in 2nd time, Its important to set the token value on the client object to an empty string, and then set the appropriate 2nd username and 2nd password.

```
bool kbmMW_Client_SetToken(ressource aClient, string aToken);
```

Set the token used by the client on the next call to the application server.
`kbmMW_Client_SetPassword($client,"SOMEPASSWORD");`

```
zval kbmMW_Client_GetData(ressource aClient);
```

Provides access to a developer defined variant which can be used to piggy back any type of data on the request calls to the application server. Similarly the application server can optionally return a piggy backed variant along with the result of the call. kbmMW do not use the Data value itself, and thus its fully available for the developers use.

```
bool kbmMW_Client_SetData(ressource aClient, zval aData);
```

Sets the developer defined variant on the client object.

Eg.

```
$variant=kbmMW_New_Variant();
kbmMW_Variant_SetAsString($variant,”This is a test”);

$client=kbmMW_New_Client($aClientTransport);
kbmMW_Client_SetData($client,$variant);

// Do the call.
…
…

kbmMW_Dispose_Client($client); // Optional
```

```
string kbmMW_Client_GetLocation(ressource aClient);
```

Provides access to a location string, which essentially can contain any string based value. It can be used by the client to tell the application server where the client is located, and thus the value can be used for extra security validation, logging, auditing etc. on the application server.
The value can be set at any time using kbmMW_Client_SetLocation.

```
bool kbmMW_Client_SetLocation(ressource aClient; string aLocation);
```

Set the location string on the client. See kbmMW_Client_GetLocation for more info.

Eg.

```
$client=kbmMW_New_Client($aClientTransport);
kbmMW_Client_SetLocation($client,"SOMEWHERE");

// Do the call
…
…

kbmMW_Dispose_Client($client); // Optional
```

```
string kbmMW_Client_GetRequestStream(ressource aClient);
```

Returns the currently defined request stream as a string (possibly binary). The request stream is a data stream that can be forwarded to the application server in addition to the normal arguments. Some services in the application server require use of request stream, like the file service for file upload functionality.
As a result, the application server can choose to return a response stream in addition to the normal result.

Eg.

```
$stream=kbmMW_Client_GetRequestStream($client);
```

```
bool kbmMW_Client_SetRequestStream(ressource aClient, string aRequestStream);
```

Sets the contents of the request stream. After the next call to the application server, the request stream will automaticallyhave been cleared out. Thus its important to set it each time stream like data is to be transmitted to the application server.

Eg.

```
kbmMW_Client_SetRequestStream($client,"Some data that can be binary");
```

```
string kbmMW_Client_GetResponseStream(ressource aClient);
```

Returns the response stream of the last call if any has been provided by the application server. The response stream is for example needed to handle in case of using the standard kbmMW file service's download facility.

Eg.

```
$stream=kbmMW_Client_GetResponseStream($client);
```

```
bool kbmMW_Client_SetResponseStream(ressource aClient; string aResponseStream);
```

Sets the contents of the response stream. This is only provided for completeness. It usually do not server any purpose to set the contents of the response stream directly.

```
long kbmMW_Client_ReleaseState(ressource aClient,
                               string aServiceName,
                               string aServiceVersion);
```

Release the state of the service instance identified by aServiceName, aServiceVersion and the currently set state ID of the client object.

After a stateful service has been called its very important to release the state as soon as the stateful service is not to be used by the client any longer. Not releasing it can result in serious server ressource drain which can lead to the server not being able to serve requests for the specific service, until the application server times the relevant services out (optional server side feature).

Eg.

```
$id=kbmMW_Client_GetStateID($client);
echo “<p>releasing state with state id=$id</p>”;
kbmMW_Client_ReleaseState($client,”SOMESTATEFULSERVICE”,”ver1.0”);
```

```
long kbmMW_Client_ReleaseStateEx(ressource aClient,
                                 string aServiceName,
                                 string aServiceVersion,
                                 long aStateID);
```

Same as ReleaseState except its possible to provide a state identifier directly in the call.

This concludes the whitepaper about the kbmMW PHP extension.

Kim Madsen
Components4Developers