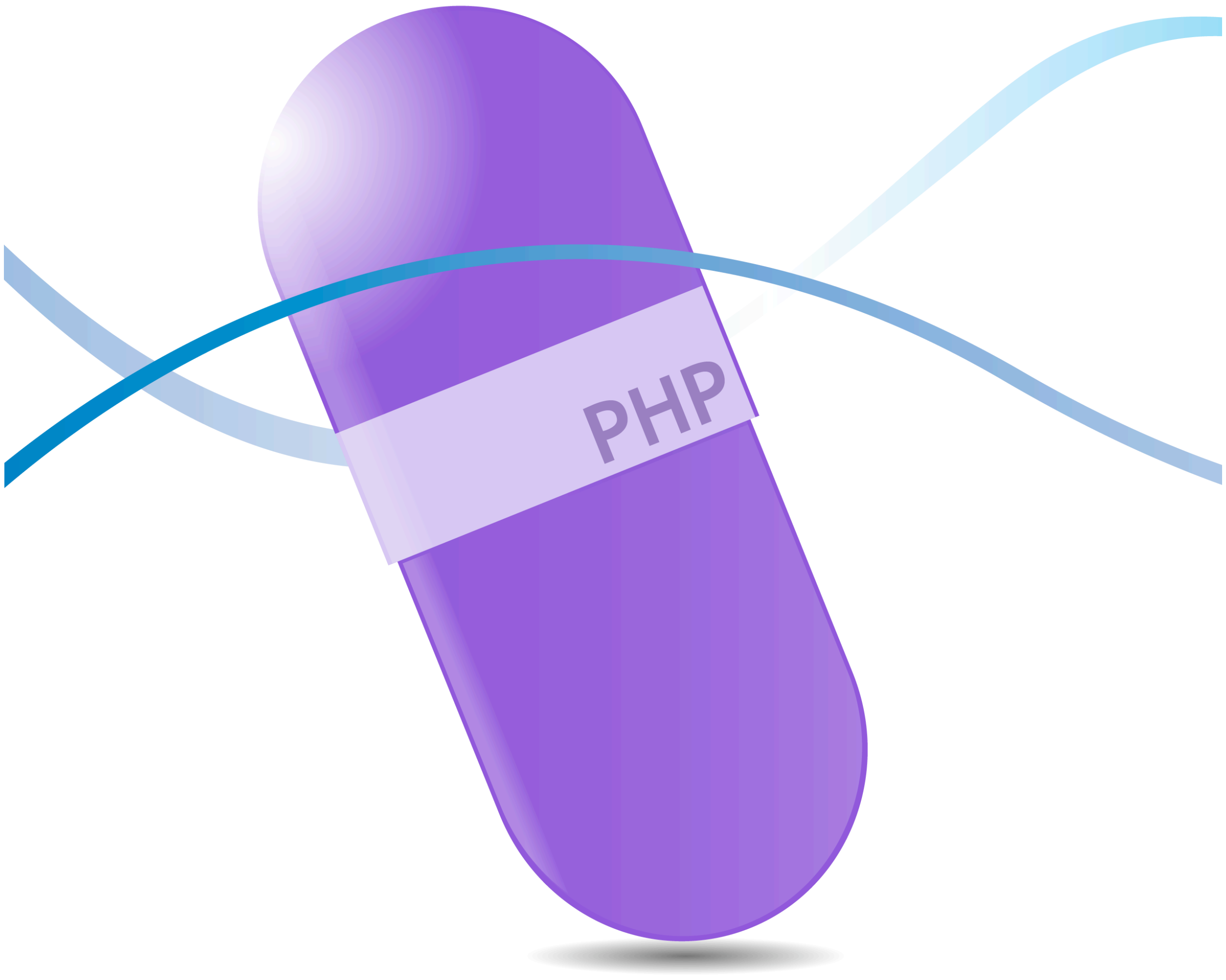
SmartPill

PHP Edition™



scodigo

User Manual

Version 1.5

Published December 1, 2007

Thanks to Pumbaa80 for the smiley.
http://commons.wikimedia.org/wiki/Image:Smiley.svg

Contents

Support

We’re here to help! Please contact us if you have any issues with installation or questions regarding how the plug-in works. We’re also very interested in feedback and would like to hear about how you’re using SmartPill.

The best way to contact us for support is by using our plug-in support request form located at http://www.scodigo.com/support. You can also send email to scodigo.support@scodigo.com.

In addition, we have forums dedicated specifically to installation and support: http://www.scodigo.com/forums.

System Requirements

FileMaker 8 or higher (including version 9).

Macintosh PowerPC or Intel
OSX version 10.3.9 for PowerPC and 10.4.5 with Intel

Pentium III 500MHz or higher
Windows 2000 (Service Pack 4)
Windows XP (Service Pack 2)

Although formal testing has not been performed, no issues have been encountered when using Microsoft Terminal Services or Citrix.

Trial License Available

Our plug-ins will function without restriction for 30 minutes each time FileMaker is launched. After this grace period is exceeded, registration is required. Please visit http://www.scodigo.com/support/trial-license if you would like a fully functional 30-day license.

Installation

SmartPill comes in 3 versions, so it's important that you install the correct version for your operating system. Use the following table to determine the correct version:

Operating System	File Name
Macintosh OSX PowerPC (PPC)	PHP_PPC.fmplugin
Macintosh OSX Intel	PHP_Intel.fmplugin
Windows	PHP.fmx

Macintosh Installation

Installation for both Macintosh platforms is the same; simply place the appropriate file (PPC or Intel version) in the Extensions folder. The Extensions folder is located inside the FileMaker application folder or in a new alternative location for FileMaker 9. The typical location is here (assuming FileMaker Pro 8.5 Advanced):

```
/Applications/FileMaker Pro 8.5 Advanced/Extensions/
```

New alternative location for FileMaker Pro 9:

```
/Users/UserName/Library/Application Support/FileMaker/
Extensions/
```

Windows Installation

Place the PHP.fmx file in the Extensions folder. The Extensions folder is located inside the FileMaker application folder or in a new alternative location for FileMaker 9. The typical location is here (assuming FileMaker Pro 8.5 Advanced):

```
C:\Program Files\FileMaker\FileMaker Pro 8.5 Advanced\
Extensions\
```

New alternative location for FileMaker Pro 9:

XP:
```
C:\Document Settings\UserName\Local\Settings\
ApplicationData\FileMaker\Extensions\
```

Vista™:
```
C:\Users\UserName\AppData\Local\FileMaker\
Extensions\
```

PHP Support Folder

If you would like to use a license file so that the plug-in is automatically registered or if you would like to use a custom php.ini file, you will need to create a folder named “PHP Support” inside the Extensions folder. We also recommend using this folder as a place to store other PHP files or libraries like PEAR.

The plug-in looks for the PHP Support folder relative to the plug-in file. This means that if the plug-in is installed in the new alternative location for FileMaker 9, the plug-in will look in this Extensions folder, NOT the Extensions folder located in the application folder.

License.txt File

You can use a license file to automatically register the plug-in when FileMaker starts up. To do so, create a text file named “License.txt” and place it in the “PHP Support” folder (which is inside the Extensions folder). You may have to create the “PHP Support” folder if it doesn’t already exist. The contents of the file should follow this format:

```
Name: Joe Smith
Company: Smith Manufacturing, Inc.
License Key: GKQJN2-MHY2TX-1DD1HI-WP0201-0100F3-3E3201
```

php.ini File

Advanced users may want to use a customized php.ini file instead of the ini file used when the plug-in was compiled. To do so, place your custom php.ini file in the “PHP Support” folder (which is inside the Extensions folder). You may have to create the “PHP Support” folder if it doesn’t already exist. We’ve included a sample php.ini file with the distribution and would recommend that you use it as a starting point for your customization.

There are two very important settings that must be set as follows:

```
max_execution_time = 0
max_input_time = 0
```

If you use a php.ini file with different settings FileMaker will crash!

Using PHP Extensions

Loading extensions is not supported in version 1.5. We are looking into the possibility of releasing a separate version where PHP is not compiled into the plug-in. This would allow extensions to load. We've worked hard to include the most popular extensions and we are open to requests. For now, if you need to load extensions, please use version 1.0. If loading extensions is important to you, please let us know.

Parameters

Many of the SmartPill's functions require you to include parameters. In some cases, all of the parameters are required and in other cases, only some of the parameters are required. In addition, some functions don't require any parameters at all. Here's a quick look at how to tell what's required.

```
PHP_Register( {licenseOwnerName ; licenseOwnerCompany ;
licenseKey} )
```

The PHP_Register function requires either **all 3 parameters** or **no parameters**. Because all 3 parameters are enclosed in curly braces, they are optional.

```
PHP_Execute( phpCode {; keepMemory {; timeoutSeconds}} )
```

The PHP_Execute function is a bit trickier; it will accept **1, 2 or 3 parameters**. The first parameter is required because it's not enclosed in curly braces. The second two parameters are optional because they are enclosed in curly braces. The third parameter is also optional because it's enclosed in it's own set of curly braces.

```
PHP_Version
```

The PHP_Version function **doesn't accept any parameters**. It simply returns the version information so no parameters are required.

Functions

PHP_Register({licenseOwnerName ; licenseOwnerCompany ; licenseKey})

This function allows the plug-in to be registered. Using PHP_Register with no parameters will return the current registration information. The plug-in will function for 30 minutes without being registered each time FileMaker is launched. After a 30 minute grace period, attempting to use any function other than PHP_Version will result in error 100 (plug-in is not registered).

Examples

```
// valid registration
PHP_Register( "Beta Tester" ; "Beta" ; "GKQJN2-
MHY2TX-1DD1HI-WP0201-0100F3-3E3201" )
```

Returns:
0 (no error, plug-in is registered)

```
// get registration info
PHP_Register
```

Returns:
Name: Beta Tester
Company: Beta
License Type: Temporary
User Count: 1
Expiration Date: 01-31-2007

```
// invalid registration
PHP_Register( "Boogie Man" ; "BMI" ;
"123456-123456-123456-123456-123456-123456" )
```

Returns:
101 Invalid registration (error, plug-in is not registered)

PHP_Version

This function returns both the plug-in version and the PHP version.

Examples

```
// check the current setting
PHP_Version
```

Returns:
1.5.0
PHP Version 5.2.4
Copyright © 2007 Scodigo, Inc. Visit http://www.scodigo.com for more info. This product includes PHP software, freely available from http://www.php.net/software.

PHP_Control({enableDisable ; password})

This function allows the plug-in to be disabled if access to the plug-in needs to be restricted. To disable, pass in a 0 as the first parameter and a password of your choosing as the second parameter. Once the plug-in is disabled, it can only be enabled by passing in a 1 as the first parameter and the correct password as the second parameter. Using PHP_Control with no parameters will return the current setting, 1 if enabled, 0 if not enabled.

Examples

```
// disable the plug-in
PHP_Control( 0 ; "secret" )
```

Returns:
0 (no error, plug-in is disabled)

```
// check the current setting
PHP_Control
```

Returns:
0 (plug-in is disabled)

```
// attempt to enable the plug-in with the wrong password
PHP_Control( 0 ; "boogie man" )
```

Returns:
105 Password to enable is wrong (error, plug-in is still disabled)

```
// enable the plug-in with the correct password
PHP_Control( 1 ; "secret" )
```

Returns:
0 (no error, plug-in is enabled)

PHP_Timeout({seconds})

This function allows the timeout duration to be set. Using PHP_Timeout with no parameters will return the current setting. The default setting is 60 seconds.

Examples

```
// check the current setting
PHP_Timeout
```

Returns:
60 (assuming the default setting hasn't been changed)

```
// set time out to 10 seconds
PHP_Timeout( 10 )
```

Returns:
0 (no error, timeout setting is changed)

PHP_GetLastError({selector})

This function provides an easy way to check for plug-in errors. If there is no error, this function will return 0, otherwise it will return an error number with a description of the error. Pass in a 1 if you only want the error number to be returned. Pass in a 2 to return only the error description.

See the fm_set_error function in the PHP FileMaker Extension section below for information on how to return user defined errors from your PHP code.

Examples

```
PHP_GetLastError
```

Returns:
0 (assuming no error)

Possible Errors:

0	No error
100	Plug-in is not registered
101	Invalid registration
102	License is for an older version of the plug-in (upgrade required)
103	License is expired (new license required)
104	Plug-in is currently disabled
105	Plug-in could not be enabled (incorrect password)
106	Invalid parameter
200	Plug-in is busy (main)
201	Timeout

PHP_Execute(phpCode {; keepMemory {; timeoutSeconds }})

This is the main function. It allows PHP code to be executed and for the results to be returned to FileMaker. The default behavior for the plug-in is to clear any previous variables, objects, etc. from memory before starting a new execution. In some cases, you may not want memory to be cleared. Pass in a 1 as the second parameter to “keepMemory”. The third parameter can be used to control the timeout duration for a single execution.

Examples

```
// the obligatory hello world
PHP_Execute( "echo 'hello world';" )
```

Returns:
hello world

```
// set $my_var to 'hello world'
PHP_Execute( "$my_var = 'hello world';" ) &

// use the PHP var_dump function to show that $my_var
// still exists when the keepMemory parameter is
// set to 1
PHP_Execute( "var_dump($my_var);" ; 1 ) &

// now execute the same code with keepMemory set to 0
// and notice that $my_var is now NULL
PHP_Execute( "var_dump($my_var);" ; 0 )
```

Returns:
string(11) "hello world" NULL

```
// use the PHP sleep function to show what happens when
// the timeoutSeconds parameter is set to a duration
// that's less than the execution time
PHP_Execute( "sleep( 2 );" ; 0 ; 1 )
```

Returns:
201 Timeout

PHP_Execute_To_Container(phpCode ; fileName {; keepMemory {; timeoutSeconds }})

This function allows you to save a file directly to a container field. Other than the fileName parameter, this function is identical to PHP_Execute. The file type should be included in the file name, for example, “scodigo-logo.png”. If the plug-in recognizes the file type as a graphic file, it will generate a preview for display. The supported graphic file types are bmp, gif, giff, jpeg, jpg, pct, pdf, png, psd, tif, tiff. Files are stored in the database just as if you had imported them without the “Store only a reference to the file” checked.

Examples

```
$url = 'http://www.scodigo.com/images/scodigo-logo.png';

$file = file_get_contents($url);

if (! $file) {
     die("Could not load file.");
}

echo $file;
```

Returns:
scodigo-logo.png file with preview

PHP Include Path

PHP uses a setting named include_path to store a list of directories where the require(), include() and fopen_with_path() functions look for files. When SmartPill starts up, it will append the path to the PHP Support folder, and an includes folder in the PHP Support folder, to the include_path. For example:

```
/Users/UserName/Library/Application Support/FileMaker/
Extensions/PHP Support:/Users/UserName/Library/Application
Support/FileMaker/Extensions/PHP Support/includes
```

Keep in mind that the PHP Support folder isn't created by default so you have to create it. It should be created in the Extensions folder where the plug-in is installed.

This gives you a reliable place to put your include files along with the convenience of not having to set the include_path with each execution.

PHP FileMaker Extension

SmartPill includes a PHP extension that adds a number of powerful functions to PHP. These are actual PHP functions that can be used in your PHP code. With these functions you can return the results of any FileMaker calculation, trigger scripts, execute SQL against your FileMaker data, create globally available variables, and define your own errors.

Evaluate Function

string **fm_evaluate** (string $expression)

Description

Evaluates the specified expression using FileMaker’s calculation engine and returns the results.

Parameters

expression (required)
Any expression that can be used with FileMaker’s Evaluate function.

Return Values

Returns the result of the expression.

Examples

```
// this example uses one of FileMaker's get functions
$expression = "Get ( ApplicationVersion )";

// print the expression and results
print("Example 1:\n" . $expression . " = " . fm_evaluate
($expression));
```

Returns:
Example 1:
Get (ApplicationVersion) = ProAdvanced 8.5v1

```
// this example returns the contents of a field
$expression = "GetField ( \"Test::Note\" )";

// print the expression and results
print("\n\nExample 2:\n" . $expression . " = " .
fm_evaluate($expression));
```

Returns:
Example 2:
GetField ("Test::Note") = Hi, I'm the contents of the note field.

```
// this example uses one of FileMaker's design functions
$expression = "Substitute ( ValueListItems ( Get
( FileName ) ; \"Colors\" ) ; \"¶\" ; \", \" )";

// print the expression and results
print("\n\nExample 3:\n" . $expression . " = " .
fm_evaluate($expression));
```

Returns:
Example 3:
Substitute (ValueListItems (Get (FileName) ; "Colors") ; "¶" ; ", ") = Red, Green, Blue

```
// this is an example of an invalid expression
// notice that the quotes are missing from the field
name
$expression = "GetField ( Test::Note )";

// print the expression and results
print($expression . " = " . fm_evaluate($expression));
```

Returns:
GetField (Test::Note) = ?

Script Functions

int **fm_perform_script** (string $file_name, string $script_name [, string $script_parameter [, int $control]])

Description

Used to perform FileMaker scripts.

Parameters

file_name (required)
The name of the file that contains the script to be performed.

script_name (required)
The name of the script to be performed.

script_parameter (optional)
ScriptParameter to be sent to the script.

control (optional)
Tells FileMaker what to do when other scripts are already running, 0 = halt, 1 = exit, 2 = resume, 3 = pause, which is the default.

Return Values

Returns one of the following FileMaker error codes.

0	No error
100	File is missing
104	Script is missing

Examples

```
// this example triggers a script named "Test Script"
// in the file named "FM_Perform_Script_Example"
// with a script parameter value of "Hello World!"
echo fm_perform_script("FM_Perform_Script_Example",
"Test Script", "Hello World!");
```

Returns:
0

bool **fm_script_control** ([bool $enable_disable])

Description

Allows script triggering to be disabled when needed. To disable, pass in false. To enable, pass in true. Using fm_script_control with no parameter will return the current setting, true if enabled, false if not enabled. The current setting is always returned, so enabling will return true, disabling will return false. Script triggering is enabled when the plug-in first starts up.

Parameters

enable_disable (optional)
The name of the file that contains the script to be performed.

Return Values

Returns **TRUE** or **FALSE** depending on the current setting.

Examples

```
// to disable script triggering
fm_script_control(false)
```

Returns:
FALSE

```
// to enable script triggering
fm_script_control(true)
```

Returns:
TRUE

```
// to return the current setting
fm_script_control()
```

Returns:
TRUE or FALSE depending on the current setting

Error Function

mixed **fm_get_last_error** ([int $selector])

Description

Used to check for errors returned by the "fm" functions included with the FileMaker PHP extension.

Parameters

selector (optional)

Use 1 to return only the error code or 2 to return only the error message. By default, both the error code and error message are returned.

Return Values

Returns the last error after calling one of the "fm" functions included with the PHP FileMaker extension. 0 is returned if there was no error. If only the error code is returned (using a selector value of 1) OR when there is no error, the return type will be int, otherwise the return type will be string.

Examples

```
echo fm_get_last_error();
```

Returns:

106 ERROR: Table missing (106) // return type is string

```
echo fm_get_last_error(1);
```

Returns:
106 (return type is int)

```
echo fm_get_last_error(2);
```

Returns:

ERROR: Table missing (106) (return type is string) bool **fm_set_error**(int $error_number [, string $error_message])

Description

Used to set a user-defined error number and message. The error number and message can be retrieved using the plug-in's PHP_GetLastError function. Any error that's less than 1,000 is a plug-in error. Any error that's equal to or greater than 1,000, is a user defined error.

Parameters

error_number (required)
User defined error number. Must be equal to or greater than 1000.

error_message (optional)
A message explaining the error.

Return Values

Returns **FALSE** if an invalid *error_number* is specified, otherwise **TRUE**.

Examples

```
$error = fm_set_error(1000, 'The supplied URL is
invalid, please check the format and try again.');

var_dump($error);
```

Returns:
bool(true)

SQL Functions

mixed **fm_sql_select** (string $sql_query [, int $column_separator [, int $row_separator]])

Description
Used to perform an SQL select and return the selected data.

Calls the ExecuteSQL function in the FileMaker plug-in API. This function is considered experimental by FileMaker, so use with caution. See the FileMaker ODBC and JDBC Developer's Guide for reference to the supported SQL.

Parameters
sql_query (required)
Any supported SQL select query.

column_separator (optional)
ASCII or Unicode character code. The pipe character "|" (code 124) is the default.

row_separator (optional)
ASCII or Unicode character code. The return character "¶" (code 167) is the default.

Return Values
Returns the **results** of the select query or **FALSE** if an error occurs. Most errors are caused by invalid SQL statements. Use fm_get_last_error to get error details and keep in mind that the error that's returned will be a FileMaker error, so you can reference FileMaker help for more information.

Examples

```
echo fm_sql_select("SELECT * FROM Company WHERE
Status_ID = 'A'");
```

Returns:
All columns from the Company table for records that have a Status_ID of 'A'. Columns are separated by the default pipe character and rows are separated by the default return character.

bool **fm_sql_execute** (string $sql_query)

Description

Used to execute any supported SQL query **except CREATE, ALTER and DROP**. Use this function for INSERT, UPDATE and DELETE. Use fm_sql_select for SELECT.

Parameters

sql_query (required)
Any supported SQL query except SELECT, CREATE, ALTER or DROP.

It's very important that you do not attempt to execute an SQL query that includes CREATE, ALTER or DROP statements. This is not supported by this function and will require you to force quit FileMaker.

Return Values

Returns **TRUE** if execution is successful, otherwise **FALSE**.
Use fm_get_last_error to get error details and keep in mind that the error that's returned will be a FileMaker error, so you can reference FileMaker help for more information.

Examples

```
echo fm_sql_execute("UPDATE Company SET Name = 'Scodigo'
WHERE Company_ID = '1000'");
```

Returns:
TRUE

Variable Functions

mixed **fm_set_variable** (string $name, string $value)

Description

Used to set the value of a new or existing variable in the plug-in’s memory.

These variables are not the same as FileMaker variables and they are available “application-wide” meaning that they can be accessed from any file. Once a variable is created, it will persist in memory until 1) you delete it using fm_unset, 2) you quit FileMaker or 3) you stop the plug-in by un-checking it in the preferences.

All variables are stored as text so you may need to apply one of FileMaker’s “GetAs” functions to convert variables back to the desired type.

Parameters

name (required)
The name of the variable to set. Note that any variable that begins with double underscores “__” will be considered hidden and will not be returned by the fm_get_defined_variables function.

value (required)
The value to set.

Return Values

Returns the **value** on success or **FALSE** on failure.

Examples

```
echo fm_set_variable('foo', '100');
```

Returns
100

mixed **fm_get_variable** (string $name)

Description

Used to get the value of an existing variable.

Parameters

name (required)
The name of the variable to get.

Return Values

Returns the **value** if the variable exists, otherwise **NULL**. Keep in mind that there’s a difference between an empty string ‘’ and NULL. It’s perfectly legitimate for the value to be empty.

Examples

```
fm_get_variable('foo');
```

Returns:
100

```
fm_get_variable('bar');
```

Returns:
NULL

bool **fm_isset** (string $name)

Description

Used to determine whether a variable is set.

Parameters

name (required)
The name of the variable to check.

Return Values

Returns **TRUE** if the variable exists, otherwise **FALSE**.

Examples

```
echo fm_isset('foo');
```

Returns:
TRUE

```
echo fm_isset('bar');
```

Returns:
FALSE

bool **fm_unset** (string $name)

Description

Used to destroy the specified variable.

Parameters

name (required)
    The name of the variable to destroy.

Return Values

Destroys the specified variable. Returns **TRUE** if the variable existed, otherwise **FALSE**.

Examples

```
echo fm_unset('foo');
```

Returns:
TRUE

```
echo fm_unset('bar');
```

Returns:
FALSE

array **fm_get_defined_variables** (void)

Description

Returns an array of all defined variables, along with their values. If no defined variables exist, or only hidden variables exist, an empty array is returned. Any variable that begins with double underscores “__” is considered hidden and will not be returned by this function. Use this naming convention for any variables that you don’t want to be exposed by this function.

Parameters

None.

Return Values

Array.

Examples

```
print_r(fm_get_defined_variables());
```

Returns:

```
Array
(
   [foo] => 100
)
```

Thanks for using SmartPill!