

PHP SETUP WIZARD VERSION 2.2

IMPLEMENTATION DOCUMENTATION

JANUARY 8, 2012

AUTHOR:

THORSTEINN SAEVAR HJARTARSON
CODECANYON.NET EXCLUSIVE AUTHOR

Table of contents

1	Introduction	6
1.1	Description	6
1.2	Features	6
1.3	File Overview	7
1.4	Change Log	9
2	Documentation	12
2.1	Configuration.php	12
2.2	Installer.php	12
2.3	Overview.php	13
2.4	Assets/	13
2.4.1	class.Database.php	14
2.4.2	class.Masks.php	14
2.4.3	class.HtmlMaker.php	14
2.4.4	helper.AdminAccount.php	15
2.4.5	helper.Destroyer.php	15
2.4.6	helper.Functions.php	16
2.4.7	helper.Sessions.php	16
2.5	Css/	16
2.5.1	Icons.css	16
2.5.2	Style.css	17
2.6	Images/	17
2.6.1	Nuvola	17
2.6.2	FamFamFam	18
2.7	Masks/	18
2.7.1	SQL masks	18
2.7.2	Other masks	19
2.7.3	Keep in mind!	19

3	Installation steps	20
3.1	Welcome Message	20
3.2	PHP Requirements	20
3.3	I/O File Permissions Test	21
3.4	Terms Of Use Agreement	21
3.5	Serial Key Confirmation	21
3.5.1	Obfuscation	22
3.5.2	Webservices	22
3.6	Language Selection	22
3.7	Time Zone Selection	23
3.8	Additional Information - <i>the customizable step</i>	23
3.8.1	Textbox and Textarea	24
3.8.2	Checkbox	25
3.8.3	Radiobox	25
3.8.4	Label and Paragraph	25
3.9	Database Server Connection	26
3.10	Database Selection / Creation	26
3.11	Database Access Test	26
3.12	Database Table Prefix	26
3.13	Install Database Tables	27
3.14	Create Administrator Account	27
3.15	Create Configuration File	27
3.16	Finished Message	28
4	Configuration	29
4.1	Installer Title	29
4.2	PHP Error Messages	29
4.3	Show Database Error Messages	29
4.4	Masks Folder name	29
4.5	Ignore Installer When Process Is Done	30
4.6	Allow Overriding Current Config	30
4.7	Allow Complete Self-Destruction	30
4.7.1	Automatically launch Self-Destruction	30
4.8	Self-Destruction Filter / Folder Removal	30
4.9	Session Prefix	31
4.10	Session Encryption Hash	31
4.11	Debug Sessions / Posts / Gets	31
5	Keywords	32
5.1	Opening-Closing Brackets	32
5.2	Installer Keywords : Connection	32
5.3	Installer Keywords : Additional	32
5.4	Installer Keywords : Serial Key Confirmation	33
5.5	Installer Keywords : Administrator Account	33

5.6	Installer Keywords : Special / Custom	34
6	PHP Requirements	35
6.1	Directives	35
6.1.1	Commands	35
6.1.2	Example	35
6.2	Version	36
6.2.1	Example.1 - Exact range	37
6.2.2	Example.2 - No range	37
6.2.3	Example.3 - PHP5 only!	37
6.2.4	Example.4 - More spesific	37
6.2.5	Example.5 - Disabled	38
7	How to use the Installer	39
7.1	Preconditions	39
7.2	Launching the Installer	39
7.2.1	Simplified approach	40
7.3	Clearing sessions when developing or testing	40
7.4	From Author	40

Requirements

Running the script

- Account at some hosting service, or a
- Webserver installed, Apache is preferred
 - Any Linux distribution: Look for LAMP with PHP
 - Windows or MAC: XAMPP at www.ApacheFriends.org

Editing the source-code

- Your favorite text editor
- Knowledge of PHP

Editing the PDF documentation

- MiKTeX 2.7 or later installed
- TeXnicCenter RC1 or later is preferred editor
- Knowledge of \LaTeX document markup language

Part 1

Introduction

1.1 Description

With the PHP Setup Wizard you can improve your own PHP project with a powerful setup system. With just few lines of code and some configuration changes, you can have a powerful Installer for your system deployment in few minutes.

1.2 Features

- 16 available setup steps, each with individual configuration
- One fully customizable step by configuration
- Lists available databases for installation, with table count
- New databases can be created if server supports it
- Automatic session handling with optional encryption
- Database table prefix selection supported
- Installation of database tables and admin users supported
- Time-zone and Language selection supported
- Required License approval is supported
- Creates the PHP config file for the system being installed
- Supports multiple configs
- Template support, referred to as *Mask files*
 - Welcome/Outro messages (HTML)

- License agreement text (TXT)
- Create table queries (SQL)
- Insert root user query (SQL)
- Output system config files (PHP)
- Three availability tests:
 - PHP Requirements test
 - Read/Write permission test
 - Database privileges test
- When installation is completed, the installer can:
 - Ignore itself and continues to run the installed system
 - Delete itself and all files in the Installer folder
 - Remove current system config and re-launch installation

1.3 File Overview

The PHP Setup Wizard is a combination of classes, helpers and mask files and image 1.1 on the following page gives a good overview of the directory structure.

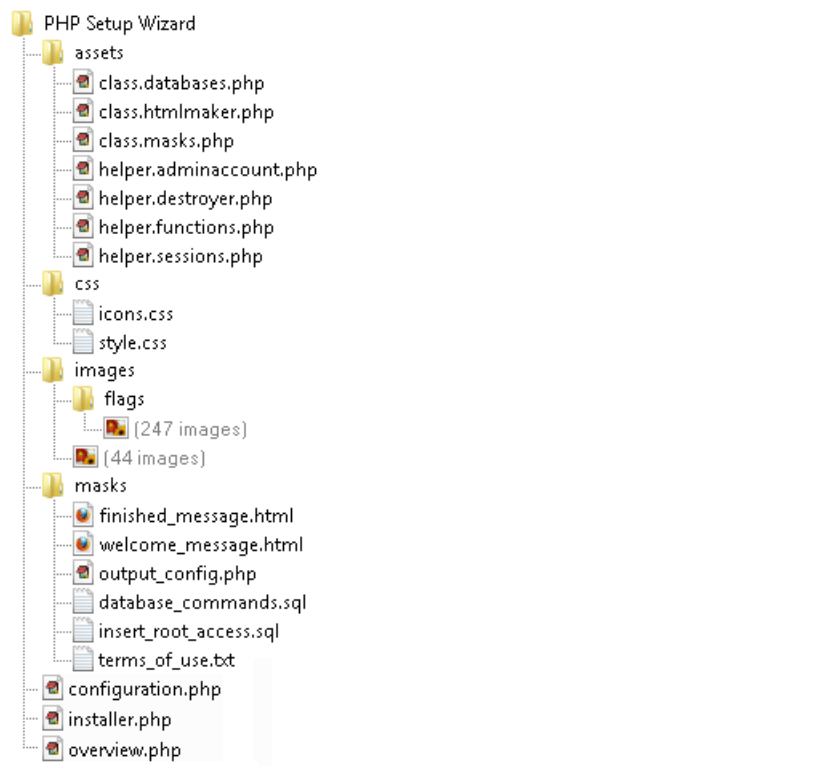


Image 1.1: Overview of files included

1.4 Change Log

Version 2.2 - January 8, 2012

Installer.php :

NEW!: Serial Key Confirmation, or `STEP_SERIALKEY` has been added. Requires user to enter serial key or allows the user to press "Trial" button for limited time.

Configuration.php :

NEW!: `STEP_SERIALKEY` step has been added, and new 'serial' section has been added to `$keywords` array.

class.Masks.php :

Improved: The keyword replacement has been improved when it comes to boolean values.

Bug-fix: In mask files where boolean values are assigned based on installer keywords such as `: define('SOME_CONST', {check1});` where the keyword `{check1}` is an boolean, `true` would donate 1 and `false` an empty string. - Meaning if the boolean value `false` would be put in a mask file by the installer, it would not write the string 'false' but instead nothing (or the empty string). This has now been fixed.

Version 2.1 - July 21, 2011

Installer.php :

NEW!: Additional Information, or `STEP_ADDEDINFO` has been added. Fully customizable by config for the users need.

NEW!: Multiple configs are now supported, and each config can be placed in different folder.

Improved: The *File Permission* step, or `STEP_IOFILES` has been re-written.

Overview.php :

NEW!: The custom form defined in `STEP_ADDEDINFO` is shown and evaluated. Any missing attributes will prompt warning.

Improved: In the *Configuration Overview* section, the custom form keywords in `STEP_ADDEDINFO` are shown with the others

Configuration.php :

NEW!: STEP_ADDEDINFO step has been added.

Rename: `maskname` and `savetofolder` in `STEP_WRITECONFIG` have been replaced by `configs`. See the documentation in section 3.15 on page 27 on how to migrate to the new version.

Asset/class.HtmlMaker.php :

NEW!: Previous queued HTML can now be altered with new `GetQueueIndex()` and `UpdateQueueAtIndex()` methods

Improved: Textareas now have the same CSS style as textboxes

Rename: `FormFormTextarea()` is now `FormTextarea()`

Asset/class.Masks.php :

Rename: `GetConfigFile()` is now `GetConfigFiles()`

Asset/helper.Functions.php :

NEW!: 5 new methods supporting STEP_ADDEDINFO step, and 5 new methods for the tests in STEP_IOFILES, plus minor refactoring.

Version 2.0 - April 23, 2010**Installer.php :**

NEW!: *Database Connection* and *Selection* steps are now optional

NEW!: PHP Version and PHP Directives (PHP . INI settings) checks have been added to *PHP Requirements*

Improved: *Database Selection*, *File Permissions Test* and *Language Selection* steps have all been improved

Bug-fix: When IO permission check was disabled, configs where not created if `savetofolder` had some value

Overview.php :

NEW!: PHP Directives evaluated in *Configuration Overview*

NEW!: PHP Version and Extension checks are performed in *PHP Requirements*

NEW!: See how the Installer parses your SQL queries for better error detecting

NEW!: See what the mask files become after keywords have been replaced

Improved: Colors have been changed and improved

Configuration.php :

NEW!: `selecttype` and `enabled` have been added for `STEP_DBSELECT`

NEW!: `enabled` has been added for `STEP_DBCONNECT`

NEW!: `default` has been added for `STEP_LANGUAGE`

NEW!: `directives`, `phpversion` and `maxversion` have been added for `STEP_PHPREQUIRES`

NEW!: `$php_directives` config has been added at the bottom of the file

Rename: `STEP_MODULECHECK` is now `STEP_PHPREQUIRES`

Rename: `autoskip` is now `ishidden` for all steps affected

Asset/class.HtmlMaker.php :

Bug-fix: Generated XHTML is now valid by standards set by *W3C Markup Validation Service*, found at <http://validator.w3.org>

Asset/class.Masks.php :

NEW!: Proper SQL query parsing has been added

Removed: SQL Separator functions and code has been completely removed

Asset/helper.Sessions.php :

Improved: `session_destroy()` is also used when Installer is reset by URL, example: `website.com/index.php?reset=all`

Bug-fix: When steps are shuffled in `$steps`, the first step in `$steps` will be the initial step, not `STEP_WELCOME`.

Asset/helper.Functions.php :

NEW!: Functions have been added, including PHP Version check functions

Version 1.0 - April 5, 2010

The first debut version of the installer.

Part 2

Documentation

This part outlines the purpose and functionality of each asset in the PHP Setup Wizard. To get a good understanding of how it all works, this will be a very helpful read.

Note: through out the documentation the word *Installer* will be used as a shorten name for "PHP Setup Wizard". Also, in all error and/or success messages, *Installer* is used instead of "PHP Setup Wizard" to not cause confusion.

2.1 Configuration.php

`configuration.php` is the one you will be spending most of your time changing and updating, as it controls and configures the Installer. In here the three major variables are configured which all the other assets use.

\$steps configures in what order the steps occur and contains settings for each step such as title, enabled and autoskip. See Part 3 on page 20 for more information.

\$config configures the installer itself such as error messages, session id and encryption passphrase. See Part 4 on page 29 for more information.

\$keywords define what information will be posted in the forms, and what data should be automatically stored in sessions. See Part 5 on page 32 for more information.

\$php_directives define how the PHP directives should be configured, such as `safe_mode`. See Part 6 on page 35 for more information.

2.2 Installer.php

The `installer.php` file contains the core functionality of the installer. If you need to add custom steps to the installer you will have to add it in this file. All steps are implemented in this file, one after the other.

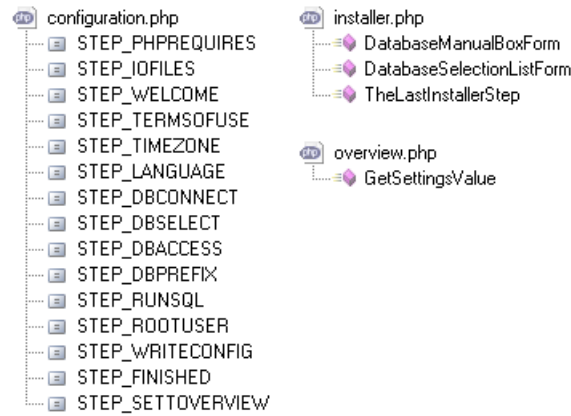


Image 2.1: Constants and functions in configuration.php, installer.php, and overview.php

2.3 Overview.php

overview.php is very helpful as it shows what each setting is set to, what the default keyword values are, what steps are enabled and so on. This file should, however, not be included with the actual setup, this file is a tool for the one configuring the Installer.

2.4 Assets/

The assets folder contains classes and helpers that the Installer uses.

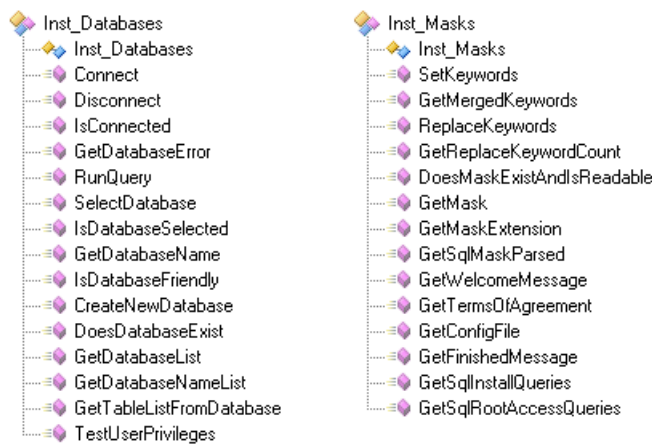


Image 2.2: Functions in Inst_Database and Inst_HtmlMasks classes

2.4.1 class.Database.php

This class takes care of all database functionality such as connection, running queries, creating databases and perform access test on the database. You can use this class in your own projects as a base class for other database/query classes.

2.4.2 class.Masks.php

Mask files are partial files and as the setup progresses, the data provided from the user is used to complete the files. Ones completed the mask files are then displayed on page, run as database queries or written to file-system as config. This class handles all functionality related to mask files.

2.4.3 class.HtmlMaker.php

Anything that the Installer outputs is done with this class. Every HTML element is added to a global queue and is displayed at the end when everything has been processed. That way, the installer can "pop" elements from the HTML queue (text- and radioboxes) and re-insert the HTML into the queue again, for example into a table.

To change the page layout itself you have to modify the `HtmlHeader()` and `HtmlFooter()` functions.



Image 2.3: Functions in Inst_HtmlMaker class

2.4.4 helper.AdminAccount.php

If the *Create Administrator Account* step is enabled, it might need to be customized to fit the system that is being installed. That includes custom functions for evaluating the required data and those functions are placed in this helper.

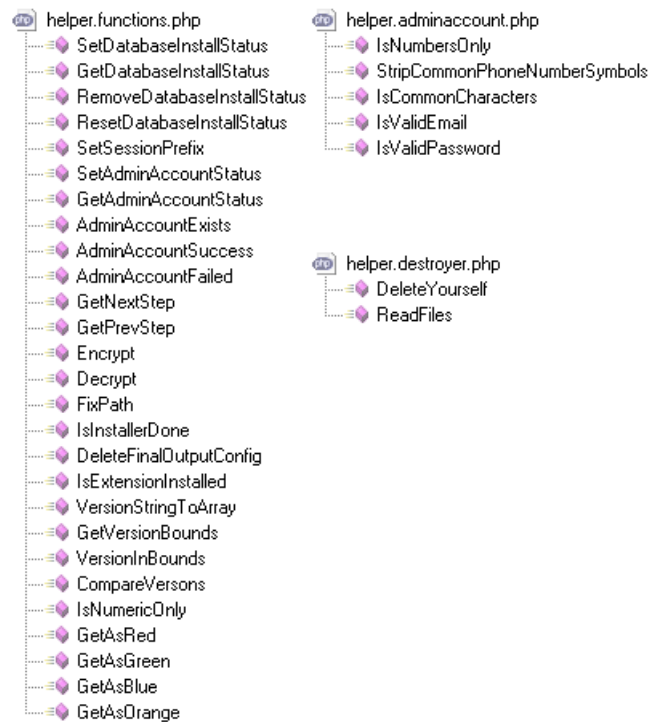


Image 2.4: Functions in the helper files

2.4.5 helper.Destroyer.php

This helper contains rather dangerous code, if included and the function `DeleteYourself()` is called, all files in the base Installer folder will be deleted including images if the web-server permits PHP to do so.

The code is recursive and takes the last folders first. Therefore it does not matter how many sub-folders there are and how many files, `DeleteYourself()` will delete everything if out-commented (and enabled and allowed by the web-server).

The lines that actually do the removal from the filesystem are out-commented such that the installer will not delete itself as you are testing the setup arrangement. When an actual deployment package is ready, these lines should be turned to code again. Look for the following code in `helper.destroyer.php` and remove the `#` symbol:

```
// Begin with deleting the files
foreach($files as $file)
{
    #if (unlink($file['dir'], $file['name']))
        $fileDelete['success']++;
    #else $fileDelete['failed']++;
}

if ($config['self-destruct.removes.folders'])
{
    // And then delete the directories
    foreach($dirs as $dir=>$scope)
    {
        #if (rmdir($dir))
            $dirDelete['success']++;
        #else $dirDelete['failed']++;
    }
}
```

2.4.6 helper.Functions.php

`helper.functions.php` is simply to aid `installer.php` and `helper.sessions.php` little bit and to prevent repetition. `helper.sessions.php` processes all posted data and saves to sessions, but in some cases the `installer.php` needs to be able to modify sessions directly when necessary.

That occurs when the data needs to be evaluated before setting the sessions like when adding a root user, `installer.php` must check if the data is valid and then execute the *install-root-user* queries and after update sessions to store success/failure status.

2.4.7 helper.Sessions.php

The `helper.sessions.php` processes all POST data and stores into SESSION for memory. First, if a posted key matches a keyword in `$keywords`, then a session is set with the posted value. The current `$step` is determined from POST, from GET or from SESSION, which controls the process of the Installer.

At the end of the `helper.sessions.php`, all session values are saved to `$keywords` so `installer.php` only has to work with that array, not the sessions directly.

2.5 Css/

There are two cascading style sheets, one mainly for the look of the Installer and the other to define all icons used by it.

2.5.1 Icons.css

`icons.css` style sheet covers styling for message box icons, the progress step dialog and all heading icons for main- and subtitles. The following example shows how it is used.

```
<h1 class="hicon connection">Server Connection</h1>
<h1 class="hicon timezone">Select Time zone</h1>
```

```
<div class="msg success">You have approved the License</div>  
<div class="msg warning">Unable to establish a connection</div>
```

2.5.2 style.css

`style.css` style sheet covers the layout of the Installer. The structure of the page is shown in the following code example, and for CSS veterans it should be quite easy to change the layout.

To change the HTML output of the page you have to modify the `class.HtmlMaker.php` class, see section 2.4.3 on page 14 for more information.

```
<body>  
<div id="container">  
    <div id="steps">  
        <!--  
            Step process dialog  
        -->  
    </div>  
  
    <div id="content">  
        <div id="installer">  
            <!--  
                Installer main content  
            -->  
        </div>  
        <div id="footer">  
            <!--  
                Footer text  
            -->  
        </div>  
    </div>  
</div>  
</body>
```

2.6 Images/

2.6.1 Nuvola

All the icons used by the PHP Setup Wizard (apart from the flags) are from an icon package named *Nuvola Icon Theme for KDE 3.x*, created by David Vignoni¹.

¹<http://www.icon-king.com>

2.6.2 FamFamFam

The country flags icons used by the PHP Setup Wizard are from an icon package named *Flag Icons*, created by Mark James².

2.7 Masks/

2.7.1 SQL masks

The SQL masks are little bit different from other masks because they contain executable commands that will affect the database. In other words, these mask files contain queries that will be executed.

The only requirement to SQL masks is that you must have a newline after each query, and that every query must end with a semi-colon ;.

The following example shows how an SQL mask should look like for the *Install Database Tables* step:

```
CREATE TABLE '{dbprefix}mytable' (  
  'id' int(11) NOT NULL AUTO_INCREMENT,  
  'name' varchar(100) DEFAULT 'dude',  
  'desc' text NOT NULL,  
  PRIMARY KEY ('id')  
);  
  
CREATE TABLE '{dbprefix}myusers' (  
  'id' int(11) NOT NULL AUTO_INCREMENT,  
  'username' varchar(50) NOT NULL,  
  'password' char(32) NOT NULL,  
  'displayname' varchar(255) NOT NULL,  
  'emailaddress' varchar(255) NOT NULL,  
  PRIMARY KEY ('id'),  
  UNIQUE KEY 'username' ('username'),  
  UNIQUE KEY 'emailaddress' ('emailaddress')  
);
```

Notice that all in front of the table names is a {dbprefix} wrapped in curly braces to indicate that "put table prefix string here". That prefix is set in *Database Table Prefix* step, or if disabled, just a value defined in \$keywords.

Insert root user mask

The following example shows how the insert into query might look like for the *Create Administrator Account* step. Normally the query would not be wrapped like this but this is just to make it fit in the box :).

Notice that the keywords used here are from the \$keywords array, more specifically the 'admin' part of it.

²<http://www.famfamfam.com>

```
INSERT INTO '{dbprefix}myusers'
(
  'id', 'username', 'password', 'displayname',
  'emailaddress', 'level', 'hashkey'
)
VALUES
(
  NULL, '{admin_username}', '{admin_password}',
  '{admin_realname}', '{admin_email}', 10, '{admin_hashkey}'
);
```

2.7.2 Other masks

Other masks such as License agreements and Welcome message contain just a block of text or HTML that will be displayed to the user. Their correctness does not matter as much as SQL masks. Here is an example of a welcome message mask:

```
<p>You are about to install <b>{product}</b>,
version <b>{version}</b>
developed by <b>{company}</b>.</p>
```

2.7.3 Keep in mind!

Always be sure that the keywords will be *replaced* in your mask files, in other words, keep your mask files synchronized with the `$keywords` array. If a keyword is not replaced, it is left alone.

Just an example, if the keyword `version` is not defined in `$keywords` in the `special` part, the outcome of the mask file would be the following:

```
You are about to install Some Product,
version {version} developed by Some Company.
```

Always double-check your Installer before releasing it!

Use the `overview.php` tool!

The `Overview.php` script will help you verify that all your masks are correct. It also checks if there are any replaceable keywords, how many times they are replaced and even shows you the content of the mask files to let you know how they will be interpreted.

Part 3

Installation steps

This part outlines all the steps in more detail along with configuration for each step. All of these steps have *title* setting, and many of them also have *enabled* and *ishidden* settings. In all cases they mean the same, so they are described here and not repeated for every step.

title: The title of the current step, which is displayed in the main-title and in the process dialog.

enabled: Should the current step be included in the setup process or not. If disabled it will be completely ignored.

ishidden: This setting lets the Installer hide a *test* step if it successful. The testing steps do not require any input from the user and can be silently processed without the user knowing about them. However, if the testing step fails the Installer will make it appear and display an error message and/or suggestion to fix the problem.

3.1 Welcome Message

Show welcome message or short introduction to what is being installed.

maskname: The name of the mask file that contains the welcome message. *HTML* is supported in this mask file.

3.2 PHP Requirements

Makes sure that the required PHP version is installed, the necessary PHP extensions are loaded and that the required PHP directives are configured correctly. To see more on the directives, see Part 6 on page 35.

Ishidden is highly recommended here, because if these requirements are all fulfilled, it does not concern the user at all!

extensions: A list of PHP extension names to require

directives: Set to `true` if any PHP directives are required

phpversion: The minimum required PHP version

maxversion: The first occurrence of a PHP version that does NOT support your system.

Here is a code example on how to set the array with extension names and their titles

```
'extensions' => array(
  'mysql' => 'MySQL Databases',
  'mcrypt' => 'Encryption',
  'zlib' => 'ZIP Archives',
),
```

3.3 I/O File Permissions Test

Checks whether if the installer can in fact create and read/write files on the server, which is the core purpose of the Installer. A test file will be created and deleted in this test to see if the Installer can do so.

- Does the config folder exist
- If not - can I create it
- Is the config folder writeable

This step will verify the *Create Configuration File* step configuration, So if this step has been passed the output config will be able to be created/updated.

3.4 Terms Of Use Agreement

If enabled the user must approve a license agreement before continuing with the install.

maskname: The name of the mask file that contains the license, plain-text only as the text is displayed in a textarea

3.5 Serial Key Confirmation

If enabled the user must enter serial key before continuing with the install. This step also supports trial button in case the product allows such thing.

allowtrial: if `true` there will be an extra section displayed below the serial key textbox that displays some minor text and then shows a trial button. The trial length time is specified in the `$keywords` array. If `false` the user must enter valid serial key.

serialkeys: An array containing one or more serial keys that the user must match in order to continue.

3.5.1 Obfuscation

This step will be rather useless if used in your setup and the PHP code is not obfuscated. Meaning, PHP is in clear text and anyone with very basic understanding of programming could easily just copy one key from `serialkeys` and paste it during installation. Consider using some obfuscation, and with that said - obfuscating these array keys might also be a good idea. How you go around doing so is totally up to you.

3.5.2 Webservices

If you plan on using a webservice for your core functionality and serial validation, you will probably have no use for `serialkeys` in this step configuration.

Simple:

The simplest way to alter the installer to support what you need is to open the function helper and change one method. Open `helper.functions.php` and go down to line 210 and change the `IsSerialKeyMatch($serialInput)` function. That method is called when the serial input is processed, and the parameter `$serialInput` is the user input (or what the user typed in).

Advanced:

If the webservice needs to return some values other than just true/false for the serial evaluation, and you need to store some data in `$keywords` array, then consider changing the very core that handles session processing.

Open `helper.sessions.php` and modify the contents between 212-216 and 221-225. These two if checks are there to verify that the form posted serial input from user, or that the user pressed trial button.

If you need any new keywords for this, make sure you add them into `$keywords` array, section 'serial' in the `configuration.php` file. See section 5.4 on page 33 for more information on how to configure keywords for this step.

3.6 Language Selection

If the system being installed requires a language selection, a list of languages can be defined along with country codes which the user selects from. The country-code must match the image name in `images/flags/..` in order for the Installer to display correct country flag. If you do not use the same country-codes, simply rename the flag images to match your country-codes.

supported: An array of supported languages, with country code and name.

default: The default selected language when the list is generated

Here is a code example on how to set the array with country codes and languages:

```
'supported' => array(
    'gb' => 'English (UK)',
    'us' => 'English (USA)',
    'de' => 'German',
    'fr' => 'French'),
```

3.7 Time Zone Selection

If the system being installed requires a timezone selection, this step can be used though it is assumed the system has already some sort of time zone mechanism, though a time zone list is offered to get you started.

3.8 Additional Information - *the customizable step*

If the system being installed needs a custom step to fill in information, you can fully customize this step to fit whatever you need. If you are familiar with form creation in HTML, this shouldn't be complicated at all.

The form itself is defined in the configuration, and there are six different components you can build up your custom form:

textbox, textarea: are input boxes where the user types in text. Same as `<input type="text">` or `<textarea>` in HTML.

radiobox, checkbox: are clickable components. Same as `<input type="radio">` or `<input type="checkbox">` in HTML.

label, paragraph: are for text display in between the form components. Same as `<h3>` and `<p>` in HTML.

Each type of component has its own set of attributes, along with minor data validation such as `numeric`, `mustfill` and few others. The whole form is defined into the `'form'` element in the steps configuration section.

```
STEP_ADDEDINFO => array(
    'title' => 'Additional Information',
    'enabled' => true,
    'form' => array( ... ),
),
```

3.8.1 Textbox and Textarea

type: is either `textbox` or `textarea` depending on how much text input you are expecting of the user. Textbox is a single line input where as textarea is multiple lines and expected for larger inputs.

keyword: is the replaceable keyword that is used in the mask files. This value should be unique to all existing keywords.

value: the default text value represented in the box when the installer opens this step for the first time. Keep this empty if no default value should be presented.

numeric: if `true`, numeric values will only be accepted. If the input is not numeric the installer will prompt a warning and will not go further.

mustfill: if `true`, blank inputs will not be accepted. If the input is blank the installer will prompt a warning and will not go further.

minlen: *[optional]* Minimum length of the string input. If the input string is shorter than this value the installer will prompt a warning and will not go further. This attribute is only used if `numeric=false`. Set this to `false` to disable it.

maxlen: *[optional]* Maximum length of the string input. If the input string is longer than this value the installer will prompt a warning and will not go further. This attribute is only used if `numeric=false`. Set this to `false` to disable it.

minval: *[optional]* Lowest numeric value accepted. If the numeric input value is lower than this value the installer will prompt a warning and will not go further. This attribute is only used if `numeric=true`.

maxval: *[optional]* Highest numeric value accepted. If the numeric input value is lower than this value the installer will prompt a warning and will not go further. This attribute is only used if `numeric=true`.

Here is an example on how a textbox array should look. The keyword that would be used in a mask file would then be `{debut_year}` and it will have a numeric value, ranging from 1920 and up to the year the script is executed (the PHP date function gives the current year in 4 digits). Minlen and Maxlen are disabled.

```
array( 'type'      => 'textbox' ,
       'keyword'   => 'debut_year' ,
       'value'     => '' ,
       'numeric'   => true ,
       'mustfill'  => true ,
       'minlen'    => false ,
       'maxlen'    => false ,
       'minval'    => 1920 ,
       'maxval'    => date( 'Y' ) ,
```

3.8.2 Checkbox

type: is `checkbox` and is a clickable component.

keyword: is the replaceable keyword that is used in the mask files. This value should be unique from all existing keywords.

value_on: is the value of the keyword if the checkbox was checked, indicating *on* state.

value_off: is the value of the keyword if the checkbox was not checked, indicating *off* state.

text: is the descriptive caption for this type and is displayed at the right hand side of the clickable component.

checked: *[optional]* indicates if this component should be pre-checked when the user enters this step or not.

3.8.3 Radiobox

type: is `radiobox` and is a clickable component.

keyword: is the replaceable keyword that is used in the mask files. All radioboxes that should be together (meaning multiple choice) should have the same keyword. Same keyword means only one of the radiobox that carries the same keyword name can be checked.

value: is the value for this given radiobox. Since some radiobox will be checked, the keyword will always have some value.

text: is the descriptive caption for this type and is displayed at the right hand side of the clickable component.

checked: *[optional]* indicates if this component should be pre-checked when the user enters this step or not.

3.8.4 Label and Paragraph

type: is `label` if it should be text on top of other input types (like text- or checkboxes). If it should be like a summary or description then it is `paragraph`. The main difference is that `label` is in bold and has no margin, where as `paragraph` is non-bold and will push text up or down to stand out little bit more.

text: is the descriptive caption for this type and the text fills the entire width of the installer body.

3.9 Database Server Connection

Prompts *host*, *username*, *password* and optional *port* input fields for the user to fill in login information to MySQL server.

portoptional: If enabled, the user has the option to fill in port value. If kept empty it will be ignored when connection is made

encryptlogin: When login is successful, this flag will make sure the host, username, password and database name will be encrypted in session using the Blowfish encryption. It is highly recommended that you enable this feature to enhance security during the installation.

3.10 Database Selection / Creation

Lists all databases available and total number of tables in each database. Offers the user to choose a database to install system tables on. New databases can also be created in this step.

allowcreate: Can the user create new databases as well

selecttype: This controls how databases are selected. `manual` means only manually-type the database name, `dblist` shows list of databases only, but if empty the manual-type box is displayed, and the last one `both` shows both options always.

3.11 Database Access Test

A test is made to see if the selected user has the permission to do basic database actions. If this step is set to `ishidden`, then it will only be shown if something in the test will fail. The queries that will be tested are *create table*, *insert*, *select*, *update*, *delete*, *alter* and *drop*.

3.12 Database Table Prefix

A very useful step to make sure that the table names will be unique and not collide with other table names. Either from other systems or another version of this same system.

optional: The table prefix is not always wanted, though it is a nice feature to offer. If you want the table prefix to be enabled but not forced - set `optional` to `true` so the user can self decide if he wants a prefix. Set it to `false` and the prefix must have a value, thus forced to have some value

separator: This string value is added at the end of the prefix. If it is already at the end (the user added it) then it will not be repeated. Keep the box empty if you wish not to use a separator. (Default value is `_` underscore)

3.13 Install Database Tables

A block of SQL will be executed, assuming the system needs to run SQL commands to create tables and perhaps insert few rows as well. During the installation, the user can install tables on numerous databases.

viewsql: Should the install script be visible to the user, if true then show in a box with scroll-bars

maskname: The mask filename containing the installation queries, see section 2.7.1 on page 18 for more detail

3.14 Create Administrator Account

The administrator account could be created in *Install Database Tables* step with hard-coded username and password. This step allows the user to pick his own username and password, along with other information.

This step has to be customized to fit the needs of the system being installed. It is nearly impossible to count for all possibilities when it comes to install users to various systems.

encryptdata: The administrator data will be encrypted in sessions for security

maskname: Name of the mask file containing the *insert* query for adding users to the system

3.15 Create Configuration File

Creates a configuration file(s) for the system being installed, and writes all the information collected throughout the installation.

updateonzero: In some cases, the config file(s) has to be created manually due to security reasons. When that happens, the installer cannot rely on the fact that if the config exists, the installation is done. So, the work around is that you create the file manually and make it totally empty!

If the installer detects the config(s), and this setting is set to *true*, then if the file has zero bytes, the installation is not done and Installer will continue.

configs: is an array that contains definitions for all the configs the system needs to be created.

- **maskname:** The name of the mask file. **Note** that the name of this mask file will become the name of the output config. So rename the config file accordingly.

- **savetofolder:** Where will the config file be saved. This setting will create a folder from where the installer was launched from, which usually is `index.php`. See the documentation for `STEP_WRITECONFIG` in `configuration.php` for more information on how to configure this setting.

3.16 Finished Message

Show finished message or short "outro" to what has been installed.

maskname: The name of the mask file that contains the finished message. HTML is supported in this mask file.

Part 4

Configuration

4.1 Installer Title

What should the name of the installer be, this will be the value in `<title></title>` in the page header.

```
'installer_title_name' => 'Your PHP Setup',
```

4.2 PHP Error Messages

Should the Installer show PHP parser error messages or not. Set to `true` if you are creating custom steps and need some debugging.

```
'show_php_error_messages' => false ,
```

4.3 Show Database Error Messages

Error messages generated by the database server when SQL queries are unsuccessful. Useful when users are at *advanced* level, but might not be needed when users are at *novice* level.

```
'show_database_error_messages' => true ,
```

4.4 Masks Folder name

What is the name of the masks folder, default value is `masks`.

```
'mask_folder_name' => 'masks',
```

4.5 Ignore Installer When Process Is Done

When the installer detects that the output config file exists and it contains some data, it can simply *ignore itself* and continue running the website. Else it will notify the user to remove the installer folder or prompt options what-to-do if enabled (see below).

```
'ignore_installer_when_done' => true ,
```

4.6 Allow Overriding Current Config

When the installer detects that the output config file exists and it contains some data, and `ignore_installer_when_done` setting is set to `false`, should the installer offer a *start all over* button. If pressed, the current config will be deleted from the server as the installer reloads, and then acknowledges that there is a need for config creation

Note: `ignore_installer_when_done` must be `false` for this setting to work!

```
'allow_overriding_oldconfig' => true ,
```

4.7 Allow Complete Self-Destruction

When the installer detects that the output config file exists and it contains some data, and `ignore_installer_when_done` is set to `false`, should the installer offer a *remove files* button. If pressed, all files in the Installer folder will be deleted from the server, or at least an attempt will be made to do so.

4.7.1 Automatically launch Self-Destruction

If the setting `allow_self_destruction` is set to `true`, this setting can enforce that mechanism to execute automatically when installer is done, and the user simply presses "Done" or "Finished" button and the installer files will be removed as the installed system is starting up for the first time

```
'allow_self_destruction' => true ,  
'automatically_self_destruct' => true ,
```

4.8 Self-Destruction Filter / Folder Removal

If for some reason, you do not want the installer to remove some specific extensions from the Installer folder, specify the ones that you want to remove, but keep the array empty to remove everything.

```
// Example: array('php', 'css'); = Removes PHP and CSS only
// Example: array();           = Removes ALL files!

'self_destruct_filter' => array(),
'self_destruct_removes_folders' => true,
```

4.9 Session Prefix

Uniquely identifies the sessions for the installer, important! Default value is `INST_` but change it if you are using it yourself

```
'session_prefix' => 'INST_' ,
```

4.10 Session Encryption Hash

The sever connection credentials will be encrypted using a blowfish encryption. It will need a key that is used to encrypt/decrypt the login info. To keep the encryption safe and unique for your installer - replace the key below and try to obscure it as much as you can for added security.

```
'encryption_key' => '*r3p14ce_tHiz-w1Th>y0uR<paS5phr4ze!*' ,
```

4.11 Debug Sessions / Posts / Gets

Do you want to know the values of `SESSION`, `POST` and `GET` at the end of each reload? This is very helpful when adding custom steps.

```
'debug_sessions' => false ,
'debug_posts'    => false ,
'debug_gets'     => false ,
```

Part 5

Keywords

5.1 Opening-Closing Brackets

These are the symbols that represents starting and closing of keywords, curly braces are default values. Example: username

```
'open_bracket' => '{',  
'close_bracket' => '}',
```

5.2 Installer Keywords : Connection

RESERVED! These keywords in this list **MUST** be here in order for the installer to function properly! Fill in the empty brackets if you want to set some default values. If default values are *correct* or *accepted* by some of the steps, the user will be prompted a success message when that step is entered, like it was posted by the user himself.

```
'connection' => array(  
  'hostname' => 'localhost',  
  'username' => '',  
  'password' => '',  
  'database' => '',  
  'dbport' => '',  
  'dbprefix' => '', # used in STEP_DBPREFIX  
) ,
```

5.3 Installer Keywords : Additional

These keywords are generated from the configured components in the `STEP_ADDEDINFO` step. Any value here is going to be discarded anyways so just keep it empty. Use the `Overview.php` tool to view the keywords available during installation.

5.4 Installer Keywords : Serial Key Confirmation

If the step `STEP_SERIALKEY` is enabled, these are the keywords that will be used by that step. Here you can specify the default values and add more keywords that you might need it.

`keyvalue` stores the default value (which is empty string), but if you put something in there it will be prompted to the user as "pre-filled" serial. So you can use that to your advantage if the user downloads full version and you would like the user to see the serial he is getting (and possibly the option to copy it).

The two `isTrial` and `isMatch` are set by the installer automatically, depending on what the user does. If the user presses "Trial" button the state will be:

- `isTrial = true`
- `isMatch = false`

If the user enters the correct serial and presses next, the state will be:

- `isTrial = false`
- `isMatch = true`

You can of course add any other additional keywords if needed

```
'serial' => array(
// The "default" serial value, keep empty if
// no pre-filled serial should be prompted
'keyvalue' => '',

// Just something to indicate some time length,
// if changed remember to change it in installer.php
'trialtime' => 10,

// These two are set at runtime by the installer
// to indicate what the user did in the step
'isTrial' => false, # was the "Trial" button clicked
'isMatch' => false # did the user enter valid serial
),
```

The keyword `trialtime` is just something I added to have some indication on how "long" the trial is. This can be fully customizable by you, and you can add your own as well. This is only to make the mask files (templates) easier to maintain as there will be no hardcoding in them.

5.5 Installer Keywords : Administrator Account

If the step `STEP_ROOTUSER` is enabled, these are the keywords that will be used with that step. Here you can specify the default values and add more keywords that you might need it.

The current keys are only a demonstration on how to use the installer mechanics to your advantage. Change these keys to what ever you like and add more if you need. Remember to keep your mask file updated with the keywords here.

The prefix `admin_` is added here to prevent the `connection` keywords to override these. In other words, if a keyword here is `username` then the `username` in `connection` will override the value here and the installer would be broken!

```
'admin' => array(
  'admin_username' => '',
  'admin_password' => '',
  'admin_passagain' => '',
  'admin_realname' => '',
  'admin_email' => '',
  'admin_phonennr' => '',
),
```

5.6 Installer Keywords : Special / Custom

These keywords are custom to your installation. Any keyword can be added here, as long as it does not collide with above reserved keywords.

If collision occurs, the reserved keyword will override the special one. You can either use these keywords for some custom steps or just to have them available to use in the mask files (like welcome message etc.)

```
'special' => array(
  'company' => 'MyCompany',
  'product' => 'MyScript',
  'version' => '1.2.3',

  // These keywords are used with Timezone
  // and Language steps
  'timezone' => '0',
  'language' => 'us',

  // Want to show the todays date
  // in welcome/outro message?
  'datenow' => date('H:m:s, F j, Y'),
),
```

Part 6

PHP Requirements

6.1 Directives

If your system requires some PHP directive to be set to true, false, have some specific `string` value, or numeric value that can be either higher or lower than something, this is where you specify your requirements.

6.1.1 Commands

title: More readable title of the directive, makes the display of these checks more user friendly.

inikey: The actual `php.ini` key, used with PHP core function `ini_get('*key*')`

mustbe: This can be `Off`, `On`, `123` or some `string`. This is simply what the setting must be "equal" to.

If the set value is `Off` or `On`, then the value will be treated as boolean. Numeric values and strings are compared as strings, but case is ignored (case-insensitive).

orhigher: `ini` values will be turned into numeric values and they must be equal or HIGHER than the one defined

orlower: `ini` values will be turned into numeric values and they must be equal or LOWER than the one defined

6.1.2 Example

Here is an example on how you define the directive title, the ini key and what the value must be, or the range it must be in. These are fairly standard settings and many system require these settings to have these values.

```
$php_directives = array
(
    array('title' => 'Running Safe Mode',
          'inikey' => 'safe_mode',
          'mustbe' => 'Off',
        ),
    array('title' => 'Register Globals',
          'inikey' => 'register_globals',
          'mustbe' => 'Off',
        ),
    array('title' => 'Short Open Tags',
          'inikey' => 'short_open_tag',
          'mustbe' => 'On',
        ),
    array('title' => 'File Uploading',
          'inikey' => 'file_uploads',
          'mustbe' => 'On',
        ),
    array('title' => 'Max Simultaneous Uploads',
          'inikey' => 'max_file_uploads',
          'orlower' => '12', // _____ OR LOWER
        ),
    array('title' => 'Floating Point Precision',
          'inikey' => 'precision',
          'orhigher' => '10', // _____ OR HIGHER
        )
);
```

6.2 Version

This check had to satisfy all possible version requirements I could think off. It had be easy to configure and support ranges. Two settings for this have been added which you can configure what your system requires.

phpversion: The minimum required PHP version. The version installed is approved if it matches this value.

maxversion: The first occurrence of a PHP version that does NOT support your system. The version installed is NOT approved if it matches this value.

Any missing version parts will fixed and given \times instead, so lets say '5' is configured, then the Installer will treat that value as '5.x.x'. Depending on the upper and lower bound, 0 is replaced for lower bound and 99 is replaced for upper bound.

6.2.1 Example.1 - Exact range

If you require version from exact point to another exact point, you put the values:

```
'phpversion' => '4.3.10',  
'maxversion' => '5.1.1',
```

And the versions approved will be any version from 4.3.10 and to 5.1.0 (5.1.1 is NOT approved).

6.2.2 Example.2 - No range

If you require PHP 5 or later, then you set `maxversion` to `false`:

```
'phpversion' => '5',  
'maxversion' => false,
```

In this case, '5' is the lower bound (minimum required version) which will be transformed into 5.x.x, and because it is the lower bound, 5.0.0 will become the lower bound.

`false` means simply x.x.x, and will be treated as 99.99.99. That means: Does the installed version fit in the range 5.0.0 - 99.99.99 ?

6.2.3 Example.3 - PHP5 only!

If you require PHP 5 only:

```
'phpversion' => '5',  
'maxversion' => '5',
```

In this case, '5' will become 5.0.0 for lower bound, and '5' for upper bound will become 5.99.99. That means any PHP 5 version is accepted, but no PHP4 or PHP6 (or lower/higher) will be accepted.

6.2.4 Example.4 - More specific

If your system needs a specific version to start from, due to some bug fix in the PHP interpreter. And because some function has been removed or changed in 5.3.0 it will no longer be supported.

However, 5.2 is still supported by the PHP team (when this guide was written) and 5.2 could still be released with newer versions. So to support all 5.2.x versions but not 5.3 and up, you would configure this:

```
'phpversion' => '5.0.2', // Bug fix in that version needed!  
'maxversion' => '5.2', // Depreciated function removed in 5.3!
```

Here 5.0.4 is exact lower bound, and '5.2' will become 5.2.99 - meaning all 5.2.x versions. The range is then between 5.0.4 - 5.2.99 (*max version 5.3.0 it would also work*).

6.2.5 Example.5 - Disabled

If you do not require any PHP version, you set these to `false` and it will be ignored by the Installer.

```
'phpversion' => false ,  
'maxversion' => false ,
```

If you are wondering what this would be transformed into, lower bound `x.x.x` and upper bound `x.x.x` would become: `0.0.0 - 99.99.99`, thus all versions will be accepted.

Part 7

How to use the Installer

7.1 Preconditions

The Installer needs to be included from somewhere else, as any direct execution of the Installer scripts will not work. To get the Installer running the `INST_BASEDIR` constant must be defined, among three others.

INST_RUNSCRIPT: Get the name of the executing script (`index.php`)

INST_BASEDIR: The path to the directory of the executing script

INST_RUNFOLDER: The name of the folder that contains the Installer files

INST_RUNINSTALL: The name of the installer file that will be launched

The constants start with `INST_` to prevent collision with your own constants or the framework you are using.

7.2 Launching the Installer

The following code block is designed to figure out the current filename and directory automatically. This will work in all cases as long as the original naming has not been changed. Put the following code block in your `index.php`

```
define('INST_RUNSCRIPT', pathinfo(__FILE__, PATHINFO_BASENAME));
define('INST_BASEDIR', str_replace(INST_RUNSCRIPT, '', __FILE__));
define('INST_RUNFOLDER', 'installer/');
define('INST_RUNINSTALL', 'installer.php');
if (is_dir(INST_BASEDIR.INST_RUNFOLDER) &&
    is_readable(INST_BASEDIR.INST_RUNFOLDER.INST_RUNINSTALL))
{
    require(INST_BASEDIR.INST_RUNFOLDER.INST_RUNINSTALL);
}
```

7.2.1 Simplified approach

If you do not want the constants be defined unless the Installer exists, and the code should be as simple as possible then try this code block instead.

```
if (is_readable('installer/installer.php'))
{
    define('INST_RUNSCRIPT', 'index.php');
    define('INST_BASEDIR', str_replace('index.php', '', __FILE__));
    define('INST_RUNFOLDER', 'installer/');
    define('INST_RUNINSTALL', 'installer.php');
    require('installer/installer.php');
}
```

If you use this approach you must be sure that these are the file- and folder names. You cannot skip the constant defines as they are used throughout the whole implementation.

7.3 Clearing sessions when developing or testing

If you are testing your setup and you would like to just "reset" the installer since it remembers the progress using sessions, you can use the following methods to do so:

reset=all will reset all sessions used by the installer and then calls `session_destroy()`

reset=additional will reset inputs used in *Additional Information* step

reset=admin will reset inputs used in *Create Administrator Account* step

reset=connection will reset data used by the *Database Server Connection* step

reset=serial will reset the data used in *Serial Key Confirmation* step

reset=install will reset the installation status of the *Install Database Tables* step

reset=special will reset the `special` part of the `$keywords` array.

And you put this into the active url, for example

<http://localhost/MyScript/index.php?step=welcomemsg&reset=all>

7.4 From Author

I hope you enjoy using PHP Setup Wizard as much as I had fun making it =)

Best regards,
Thorsteinn Saevar