FormHandler 3

manual

Manual FH3v1.2.12

1 - Getting Started

In this chapter I will tell about FormHandler. What is formhandler, how did it become like it now is and how it works. Just keep reading!

1.1 Introduction

FormHandler is a PHP written "module" which allows you to create dynamic forms in an easy way. So easy that you can build a fully working form, including field validations, within 10 lines!

Why is FormHandler Built?

Around 2002 Teye Heimans, 21 years came to the conclusion that at least 40% of his webapplications applications consists of forms. To write all these forms is an irritating and time-taking job. That's why He made the FormHandler.
In the beginning it was a simple class which could generate some fields. After sharing it with some friends he decided to put it on the internet. Version 1 is downloaded over 5000 times and version 3 over 15000 time (Version 2 was never launched, it was too complex).

In 2007 Teye decided he didn't have the time to maintain this package and placed a news item about selling the package on this site. Since a lot of our projects are depending on FH3, we wanted to ensure FormHandler stayed in good hands so PHP-GLOBE became owner of this package.

Some advantages of FormHandler

With FormHandler you can:

- Very easy to generate a form
- Easy to save/edit data from a database
- Easy to validate the values
- You can use templates!
- The form is generated by the XHTML 1.0 standard
- Possibility to change to style of the form
- Possibility to add CSS / Javascript to the fields
- There can be generated special fields (like datefields)
- It's very easy to upload files
- Possibility to generate an online text editor!
- It's completely FREE!

Sounds good huh? So, what are the requirements?

To use the FormHandler you must have a webserver with PHP version 4.0.6 or higher installed. To use some fields the visitor needs to have a browser which supports JavaScript.
If you want to use the database options of the class, you need a database.

How does the FormHandler actually work?

The FormHandler generates a form and shows this to the visitor. After the visitor has filled in his "data" the form is sent (to itself). The class validates the values (if wanted, with your own validate function). When all the data is correct, the data is saved (if wanted) and your 'commit after form' function is called. In this function you can do whatever you want...

1.2 What Is FormHandler

1.3 History

The first version of FormHandler was made during Teye's first internship for school. He was doing an education for system/network manager, but during his internship computer systems did not catch his interests. However, PHP took. Programming took. The company where he was doing his internship wanted a stock-manage program. And so he begun...

Soon he found out that making forms in PHP and HTML is a very time taking job. You can imagine that a stock-manage program consists out 50% of forms. So, time to take some action: The first version of FormHandler was build. This version has grown but it's structure is still the same. It is still used in the stock-manage program (still in the download section (dutch!)).

Over time FormHandler has been completly rewritten multiple times. Version 1 (which consisted out of one file) was so succesful that he decided to publish it on the internet. He recieved many requests and FH1 grew fast. However, it became to big for one file (in his opinion). He started to write a new version, but it was so complicated that it never was published. He was not happy with version 2 so he decided to make a new version. Version 3 was born. The structure is much better and he was very happy with it.

In 2007 Teye decided he didn't have the time to maintain this package and placed a news item about selling the package on this site. Since a lot of our projects are depending on FH3, we wanted to ensure FormHandler stayed in good hands so PHP-GLOBE became owner of this package.

Of course there are still requests for new functionality and sometimes a bug is found. So, it's still in development...

1.4 A Simple Tutorial

Okay, so you have downloaded FormHandler (FH) and uploaded it to your server. What now?
First of all you need to realize that FH is a tool that will allow you to build forms on a easy way. It is not a drag and drop application which will make your forms. You still need to write your own forms (only then with the help of FH)!

Ok, now that that is out of the way, we should be getting started with using FH. Please note that I will not explain in the examples below how every function works! Use the manual! There are examples how to use the function on allmost every page in the manual!

To start using FH, you should have a script where you are going to make the form. For example, "test.php".
Now you have to make sure that the file "class.FormHandler.php" is included into that file:

```
<?php

// include the class
include("FH3/class.FormHandler.php");

?>
```

Now you have to make a new object of the FormHandler class.

```
<?php

// include the class
include("FH3/class.FormHandler.php");

// create a new FormHandler object
$form = new FormHandler();

?>
```

Now you can start making your form. (So setting some fields e.g.)
When you are done with that you have to give FormHandler the name of the function you want to use if the form is submitted and valid. You can do this with the function onCorrect. Finally you have to flush the form so that it will be send to the visitors browser.

```php
<?php

// include the class
include("FH3/class.FormHandler.php");

// create a new FormHandler object
$form = new FormHandler();

// some fields.. (see manual for examples)
$form->textField( "Name", "name", FH_STRING, 20, 40);
$form->textField( "Age", "age", FH_INTEGER, 4, 2);

// button for submitting
$form->submitButton();

// set the 'commit-after-form' function
$form->onCorrect('doRun');

// display the form
$form->flush();

// the 'commit-after-form' function
function doRun( $data )
{
   echo "Hello ". $data['name'].", you are ".$data['age'] ." years old!";
}

?>
```

Of course you can change the fields and so. It's all in the manual!
After writing the script, upload it and request it with your browser. Your'e done!

If you still cant make FH work, use the forum.

1.5 How To Use The Database Option

When you use the database option, FormHandler will take care of the saving and loading of the data. Below is explained how it works and what you can expect from FormHandler.

Database Requirements

FormHandler can use the following databases:

- **mysql** - MySQL (www.mysql.com)
- **postgresql** - PostgreSQL (www.postgresql.org)
- **access** - Microsoft Access (Windows only)
- **mssql** - Microsoft SQL Server (www.microsoft.com/sql)

By default, FormHandler uses the MySQL database communication layer. You can change this default

value in the config file. You can also change this when you set the table which FormHandler should use.

To let FormHandler save and load the data from your database, you have to make a table. This table has to have at least one primary key!!! This is **very** important, otherwise FH cannot save and load the data from your database!

After creating the table you have to let FormHandler know which table it should use. You can do this in two ways, depentend on if you have a database connection open:

- Let FormHandler open a new connection to the database
- Let FormHandler use an already opened connection

Please note that when you want to use the database functionality, you have to include the file class.**db**FormHandler.php and use the constructor $form = new **db**FormHandler()!

Below is explained how each way works:

Let FormHandler open a new connection to the database

If you want to let FormHandler create a new connection to the database, you should supply the needed data to make a new connection. You can do this with the functions dbInfo and dbConnect. In these functions you have to set all the needed data to connect to the database. Check them out in the manual to find out what arguments are passed trough.

Example:

```php
<?php

// include the dbFormHandler
include("FH3/class.dbFormHandler.php");

// create a new form
$form = new dbFormHandler();

// set the database info
$form -> dbInfo( "myDB", "myTable", "mysql" );
$form -> dbConnect( "localhost", "username", "password" );

// here comes the rest of the form
// ............

// set the data handler
// (NOTE the onSaved, this is different then onCorrect!)
$form -> onSaved("doSomething");

// display the form
$form -> flush();

// the data handler...
// NOTE the two arguments!!!!
function doSomething( $id, $data )
{
   // do something here...
}

?>
```

After passing all the needed data you have to add some fields to the form. This time however you have to name the fields exactly the same as in the table! All fields which are simular will be saved / loaded correctly. All other fields will be ignored!

This is all whats needed to make a good working form which is using a database to save all the data. Easy huh?! ;-)

Let FormHandler use an already opened connection

When you want to let FormHandler use an already opened connection, you have to give the connection resource and the table name to FormHandler. You can do this with the function setConnectionResource.

Example:

```php
<?php
/**
 * Your Database Connection (In This Example MySQL)
 */
$connection = mysql_connect( "localhost", "username", "password" );

// connection made ?
if( $connection )
{
   // select the database
   if( !mysql_select_db( "database", $connection ) )
   {
      // could not select database!
      die('Could not select the database! Error: ' . mysql_error() );
   }
}
// could not connect to the database
else
{
   die('Could not make a connection to the database! Error: ' . mysql_error() );
}

// include the formhandler
include('FH3/class.dbFormHandler.php');

// create a new form
$form = new dbFormHandler();

// set the database info
$form -> setConnectionResource( $connection, "myTable", "mysql" );

// here comes the rest of the form
// .....

// set the data handler
// (NOTE the onSaved, this is different then onCorrect!)
$form -> onSaved("doSomething");

// display the form
$form -> flush();

// the data handler...
// NOTE the two arguments!!!!
function doSomething( $id, $data )
{
   // do something here...
}

?>
```

Inserting

Inserting data is extreme easy. If you just have created a form like explained above, the only thing you have to do is run the script.

Remember: Fields which have the same name as the column names in the table will be saved into the database. All the other fields are ignored.

So, if you have a MySQL table which looks like this:

```
CREATE TABLE `myTable` (
id int(6) unsigned NOT NULL auto_increment,
title varchar(50) NULL,
text text NULL,
PRIMARY KEY (id)
)
```

A form like below wil be correct and the data will be saved into the table.

```php
<?php

// include the class
include("FH3/class.dbFormHandler.php");

// create a new formhandler object
$form = new dbFormHandler();

// set the database info
$form -> dbInfo   ( "myDb", "myTable" );
$form -> dbConnect( "localhost", "username", "password" );

// the fields + button
$form -> textField( "Title", "title", FH_STRING, 20, 50 );
$form -> editor   ( "Text", "text", FH_TEXT, "images/uploads" );
$form -> submitButton( "Save" );

// set the data handling function
$form -> onSaved  ( "doRun" );

// display the form
$form -> flush();

// the data handling function
// NOTE: 2 arguments! This differs from the function onCorrect!!
function doRun ( $id, $data )
{
   echo " The data is saved with id ".$id."!\n";
}
?>
```

Editing records

Editing records works the same as inserting records. Your form is excactly the same. The only thing

which you have to do is to let FormHandler know which record should be edited. You can do this by passing the id of the record trough the URL:

myScript.php?id=4

In this example, record 4 will be edited. If you have multiple primary keys, you have to pass them as an array:

myScript.php?id[]=4&id[]=en

That's all! Really!

If you want you can change the var name which FormHandler will check. By default, this is set to "id". You can change it in the config file with the var FH_EDIT_NAME.

NOTE!!! If you don't want users to edit specified records you have to make sure that they can't! FormHandler DOES NOT HANDLE THIS!

1.6 Validating Fields

You can validate the value of a specific field at two ways.

1. **Use your own validator**
2. **Use a build-in validator from FormHandler**

Below is explained how to do each option.

1. Use your own validator

To use your own validator you have to write a function which checks the value of the field. The value of the field is passed to the function as argument.
After checking the value of the field you have to return if the value was valid or not.

- The value is **valid** if you return **true**.
- The value is **invalid** if you return **false** or **a string**. When you return a string, the text is used as error message.

After building the function you can give FormHandler the name of the function by the field you want to validate. You have to do this at the argument called "validator" (this argument is present in each field).

Example:

```php
<?php

// include the class
include("FH3/class.FormHandler.php");

// create a new formhandler object
$form = new FormHandler();

// textfield which we want to validate
// with our OWN function called myValidator
$form -> textField("Name", "name", "myValidator");

// submitbutton
$form -> submitButton();

// set the handler
$form -> onCorrect("doRun");

// flush it
$form -> flush();

// the handler function
function doRun( $data )
{
   echo "Hello " . $data["name"];
}

// Our own validation function!!!!
function myValidator( $value )
{
   // check the value
   if( strLen( $value ) == 0)
   {
      return "You have to enter your name!";
   }
   // there is somehting subitted.. value is OK
   else
   {
      return true;
   }
}
?>
```

The result after submitting an empty field:

2. Use a build-in validator from FormHandler

FormHandler has several build-in functions to check if the value of a field is valid or not. When not, the default error message is shown (from the language file).

FormHandler has these build in functions:

- **FH_STRING** - Any string that doesn't have control characters (ASCII 0 - 31) but spaces are allowed
- **FH_ALPHA** - Only letters a-z and A-Z
- **FH_DIGIT** - Only numbers 0-9
- **FH_ALPHA_NUM** - Letters and numbers
- **FH_INTEGER** - Only numbers 0-9 and an optional - (minus) sign (in the beginning only)
- **FH_FLOAT** - Like FH_INTEGER, only with , (comma) or . (dot)
- **FH_FILENAME** - A valid file name (including dots but no slashes and other forbidden characters)
- **FH_BOOL** - A boolean (TRUE is either a case-insensitive "true" or "1". Everything else is FALSE)
- **FH_VARIABLE** - A valid variable name (letters, digits, underscore)
- **FH_PASSWORD** - A valid password (alphanumberic + some other characters but no spaces. Only allows ASCII 33 - 126)
- **FH_URL** - A valid URL
- **FH_URL_HOST** - A valid URL (http connection is used to check if url exists!)
- **FH_EMAIL** - A valid email address (only checks for valid format: xxx@xxx.xxx)
- **FH_EMAIL_HOST** - Like FH_EMAIL only with host check
- **FH_TEXT** - Like FH_STRING, but newline characters are allowed
- **FH_NOT_EMPTY** - Check if the value is not empty
- **FH_NO_HTML** - Check if the value does not contain any HTML
- **FH_IP** - Check if the value is an valid ip adres (xxx.xxx.xxx.xxx:xxxx). Port number is allowed)
- **FH_POSTCODE** - *For dutch people!)* A valid dutch postcode (eg. 9999 AA)
- **FH_PHONE** - *For dutch people!)* A valid dutch phone-number(eg. 058-2134778)

Note: *When you want to allow an empty value but when the field is not empty it has to be valid you can use the same functions as above only beginning with a underscore (_).* Example: **_FH_STRING**

The functions above can be set at the same way you should set your own validator, only **do not put quotes arround them**!

Example: Usage of a build in validator

```php
<?php

// include the class
include("FH3/class.FormHandler.php");

// create a new formhandler object
$form =& new FormHandler();

// textfield which we want to validate
// with the build in validator FH_STRING
$form -> textField("Name", "name", FH_STRING);

// submitbutton
$form -> submitButton();

// set the handler
$form -> onCorrect("doRun");

// flush it
$form -> flush();

// the handler function
function doRun( $data )
{
   echo "Hello " . $data["name"];
}

?>
```

The result after submitting an empty field:

1.7 enableAjaxValidator

Enables on the fly validation by AJAX for the buildin validators

2 - Installation & Configuration

2.1 Installation

Installing FormHandler is quite easy. To be honest, it only needs to be uploaded to the server and you can use it.
If you want, you can place the FormHandler dir above the webroot so that visitors can not requirest the pages directly. If you want to do this, you have to place the dir FHTML below or in the webroot! The dir FHTML contains pages which are directly requested by the browser. After doing this, you have to set up the correct
path to the FHTML dir in the config file. You can do this by changing the config var FH_FHTML_DIR. Really, thats all! :-)

If you want to know how to use FormHandler please read the chapter Getting started.

2.2 Configuration

You can change to configuration of FormHandler by editing the file

FH3/includes/config.inc.php

In this file are the configuration options are commented with a small description. When you change this file you will change the default settings of FormHandler.

You can also set a specific configuration option per form. You can do this by defining them before you create a new FormHandler object.

Example:

```
<?php

// Set the location of the FHTML dir
// (so overwrite the default location!)
define('FH_FHTML_DIR', '/includes/FH3/FHTML');

// create new formhandler object
$form = new FormHandler('myForm');

// etc ....

?>
```

3 - Functions

In this section of the manual you will find all functions and methods you can use.

3.1 Fields

In this section all the fields are described which you can use in your form. Please note that the fields which resides on the database option (like dbSelectField are explained under the section Database Functions)

3.1.1 TextField

This function will create a textfield on the form.

The arguments are:
- $title: Title of the field

- $name: Name of the field

- $validator: This can either be the name of your own validation function or the constant name of a predefined validator function. For more information about validators see Validators.
It is also possible to use a method as a validation function. In this case you should pass an array where the first item is the object and the second item the name of the method.

- $size: The size of the field.

- $maxlength: The maximum allowed characters in this field. 0 is interpretated as no limit.

- $extra: This can be extra information which will be included in the fields html tag. You can add for example some CSS or javascript.

Example:

```php
<?php

// include the class
include('FH3/class.FormHandler.php');

// make a new $form object
$form = new FormHandler();

// a textfield
$form->textField("Your name", "name", FH_STRING);

// submitbutton beneath it
$form->submitButton("Save");

// set the 'commit after form' function
$form->onCorrect("doRun");

// send the form to the screen
$form->flush();

// function to show a message
function doRun($data)
{
   return "Hello ". $data["name"];
}
?>
```

3.1.2 PassField

Creates a passfield on the form.

The value of the field is never filled into the field, so the password is never vissible in the source code of the page. When the database option is used and the form is in edit mode, the password will not be saved when no value is given (so the original is kept)

The arguments are:
- $title: Title of the field

- $name: Name of the field

- $validator: This can either be the name of your own validation function or the constant name of a predefined validator function. For more information about validators see Validators.
It is also possible to use a method as a validation function. In this case you should pass an array where the first item is the object and the second item the name of the method.

- $size: The size of the field.

- $maxlength: The maximum allowed characters in this field. 0 is interpretated as no limit.

- $extra: This can be extra information which will be included in the fields html tag. You can add for example some CSS or javascript.

Example:

```php
<?php

// include the class
include('FH3/class.FormHandler.php');

// make a new $form object
$form = new FormHandler();

// a textfield
$form -> passField("Your password", "pass", FH_PASSWORD);

// submitbutton beneath it
$form -> submitButton("Save");

// set the 'commit after form' function
$form -> onCorrect("doRun");

// send the form to the screen
$form -> flush();

// function to show a message
function doRun($data)
{
   return "Your password is: ". $data["password"];
}
?>
```

Here is an example with password's check and saving the password encrypted into the database:

```php
<?php

// include the database formhandler class
include("FH3/class.dbFormHandler.php");

// create new dbFormHandler object
$form = new dbFormHandler();

// set the database connection info
$form -> dbInfo( "myDatabase", "myTable" );
$form -> dbConnect( "myHost", "myUsername", "myPassword" );

// create the two password fields
$form -> passField( "Password", "myPass" );
$form -> passField( "Retype password", "myPass_re" );

// check the password fields.
// See for more info about this function the manual
$form -> checkPassword( "myPass", "myPass_re" );

// Only save the value when it's an insert form
// and when a value is given
if( !$form -> edit || $form -> value("myPass") != '' )
{
   // now save the value encrypted!
   $form->addValue("myPass", md5( $form->value("myPass") ) );
}

// button to submit the form
$form -> submitButton();

// what to do when the form is saved ?
$form -> onSaved( "doSomething" );

// show the form
$form -> flush();

// this is the function which is called after the data is saved into
// the database.
function doSomething( $id, $data )
{
   echo "The data is saved!";
}
?>
```

3.1.3 HiddenField

You can use this function to create a hidden field on the form.

Do not use this function if you want to add a value into the datababse! Use AddValue instead! This function is only usefull if you want to change the value of the hidden field with javascript or something like that.

The arguments are:
- $name: Name of the field

- $value: The value for the hidden field.

- $validator: This can either be the name of your own validation function or the constant name of a predefined validator function. For more information about validators see Validators.
It is also possible to use a method as a validation function. In this case you should pass an array where the first item is the object and the second item the name of the method.

- $extra: This can be extra information which will be included in the fields html tag. You can add for example some CSS or javascript.

Example:

```php
<?php

// include the class
include('FH3/class.FormHandler.php');

// make a new $form object
$form = new FormHandler;

// set a hidden field
$form -> hiddenField("language", "nl");

// a normal textfield
$form -> textField("Your name", "name", FH_STRING);

// button beneath it
$form -> submitButton("Save");

// set the 'commit after form' function
$form -> onCorrect("doRun");

// send the form to the screen
$form->flush();

// function to show a message
function doRun($data)
{
   return
   "Hello ". $data["name"] .",\n".
   " your prefered language is " . $data["language"];
}
?>
```

3.1.4 TextArea

This function generates a TextArea on the form

Note: When submitting very much data please dont use the FH_TEXT validator. The validator is to time/memory expensive and submitting will fail. For small and normal amounts of data you can just use FH_TEXT.

The arguments are:
- $title: Title of the field

- $name: Name of the field

- $validator: This can either be the name of your own validation function or the constant name of a predefined validator function. For more information about validators see Validators.
It is also possible to use a method as a validation function. In this case you should pass an array where the first item is the object and the second item the name of the method.

- $cols: The width of the textarea.

- $rows: The height of the textarea

- $extra: This can be extra information which will be included in the fields html tag. You can add for example some CSS or javascript.

Example:

```php
<?php

// include the class
include('FH3/class.FormHandler.php');

// new $form object
$form = new FormHandler();

// the textarea
$form -> textArea("Message", "message", FH_TEXT);

// button beneath it
$form -> submitButton("Save");

// set the 'commit after form' function
$form -> onCorrect("doRun");

// flush the form
$form -> flush();

// function to show the selected items
function doRun($data)
{
   return "Your message: <br>\n".
   nl2br($data["message"]);
}

?>
```

3.1.5 SelectField

This function generates a SelectField on the form.

Note: the possibility to group certan options with LABEL only works with Microsoft Internet Explorer 6, Mozilla, Netscape Navigator 6 + and Opera 7.

Note: when "multiple" is enabled FH will return an array as value (otherwise a string). When the value is saved into a database it will be comma seperated.

The arguments are:
- $title: Title of the field

- $name: Name of the field

- $options: The options used for the field. It is possible to group certain options. This can be done by adding LABEL in the key of the array. Offcourse every key must have a unique name. See the example below.

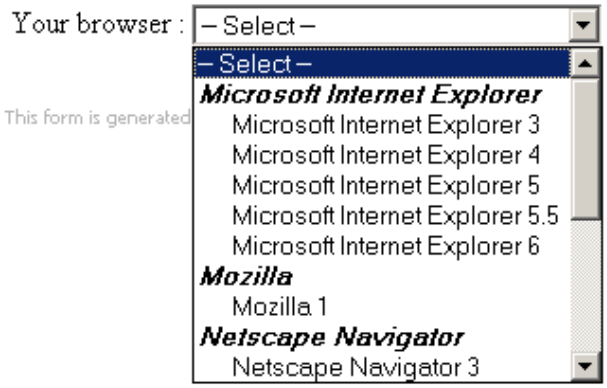
- $validator: This can either be the name of your own validation function or the constant name of a predefined validator function. For more information about validators see Validators.
It is also possible to use a method as a validation function. In this case you should pass an array where the first item is the object and the second item the name of the method.

- $useArrayKeyAsValue: Should the key of the array be used as the value for the field? If not (false) the array value is used. The default option for this can be set in the config file (see FH_DEFUALT_USEARRAYKEY).

- $multiple: Should it be possible to select more then one option?

- $size: The number of items which should be shown. Default 1, but when $multiple is set to true the size will be 4.

- $extra: This can be extra information which will be included in the fields html tag. You can add for example some CSS or javascript.



Example:

```php
<?php

// the options
$browsers = array(
   ""          => "-- Select --",
   "__LABEL(IE)__" => "Microsoft Internet Explorer",
   "msie3"       => "Microsoft Internet Explorer 3",
   "msie4"       => "Microsoft Internet Explorer 4",
   "msie5"       => "Microsoft Internet Explorer 5",
   "msie55"       => "Microsoft Internet Explorer 5.5",
   "msie6"       => "Microsoft Internet Explorer 6",
   "__LABEL(MO)__" => "Mozilla",
   "moz1"        => "Mozilla 1",
   "__LABEL(NN)__" => "Netscape Navigator",
   "nn3"         => "Netscape Navigator 3",
   "nn4"         => "Netscape Navigator 4",
   "nn6"         => "Netscape Navigator 5",
   "nn6"         => "Netscape Navigator 6",
   "nn7"         => "Netscape Navigator 7",
   "__LABEL(OP)__" => "Opera",
   "op3"         => "Opera 3",
   "op35"        => "Opera 3.5",
   "op4"         => "Opera 4",
   "op5"         => "Opera 5",
   "op6"         => "Opera 6",
   "op7"         => "Opera 7"
);

// include the class
include('FH3/class.FormHandler.php');

// create a new formhandler object
$form = new FormHandler();

// the field
$form -> selectField("Your browser", "browser", $browsers, FH_NOT_EMPTY, true);

// button beneath it
$form -> submitButton("Save");

// set the 'commit after form' function
$form -> onCorrect("doRun");

// send the form to the screen
$form -> flush();

// the function witch handles the code after the form
function doRun($data)
{
   return "Your browser: ". $data["browser"];
}

?>
```

3.1.6 CheckBox

This function generates one ore more CheckBoxes.

When an array of options is given, an array of the selected items will be returned.

The arguments are:
- $title: Title of the field

- $name: Name of the field

- $value: The value which is used for the field. When you give an array multiple checkboxes will be generated. The array value is used for the field label. The array key is used as field value, but can be changed with the argument $useArrayKeyAsValue.

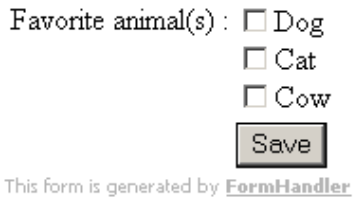
If a string is passed it will be used as value (No label is used!).

- $validator: This can either be the name of your own validation function or the constant name of a predefined validator function. For more information about validators see Validators.
It is also possible to use a method as a validation function. In this case you should pass an array where the first item is the object and the second item the name of the method.

- $useArrayKeyAsValue: Should the key of the array be used as the value for the field? If not (false) the array value is used. The default option for this can be set in the config file (See FH_DEFUALT_USEARRAYKEY).

- $extra: This can be extra information which will be included in the fields html tag. You can add for example some CSS or javascript.

- $mask: With this argument you can position the field(s). In the given mask a the placeholder `%field%` will be replaced with the field. When the mask is full and there are still fields, the mask will be repeated. This works the same as SetMask.
When no mask is given the mask will be used which is set in the config file in the var FH_DEFAULT_GLUE_MASK.



Example:

```php
<?php

// include the class
include('FH3/class.FormHandler.php');

// the options for the checkbox
$animals = array(
  "Dog",
  "Cat",
  "Cow"
);

// make a new $form object
$form = new FormHandler();

// make the checkbox
$form -> checkBox("Favorite animal(s)", "animal", $animals, null, false);

// button beneath it
$form -> submitButton("Save");

// set the 'commit after form' function
$form -> onCorrect("doRun");

// send the form to the screen
$form -> flush();

// the function which handles the code after the form
function doRun($data)
{
   // do something here
   echo "Your favorite animal(s):\n ";

   foreach($data['animal'] as $animal)
   {
      echo " - $animal\n";
   }
}

?>
```

Example use which just one checkbox:

```php
<?php

// include the class
include('FH3/class.FormHandler.php');

// make a new $form object
$form = new FormHandler();

// make the checkbox
$form -> checkBox("Ok?", "ok", 1);

// button beneath it
$form -> submitButton("Save");

// set the 'commit after form' function
$form -> onCorrect("doRun");

// send the form to the screen
$form -> flush();

// the function which handles the code after the form
function doRun($data)
{
   // do something here

   if( $data['ok'] )
   {
      echo "Yeah!";
   }
   else
   {
      echo "Booo!";
   }
}

?>
```

In this example the field will return 1 when it's checked. When it's not checked, the value will be "" (an empty string)
Because *1* is evaluated as *true* and "" as *false* this is easy to use for if statements...

3.1.7 RadioButton

This function generates a group of radiobuttons on the form.

The arguments are:
- $title: Title of the field

- $name: Name of the field

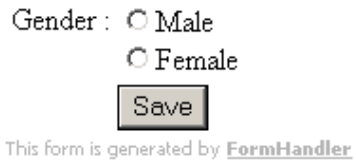
- $options: The options used for the field

- $validator: This can either be the name of your own validation function or the constant name of a predefined validator function. For more information about validators see Validators.
It is also possible to use a method as a validation function. In this case you should pass an array where the first item is the object and the second item the name of the method.

- $useArrayKeyAsValue: Should the key of the array be used as the value for the field? If not (false) the array value is used. The default option for this can be set in the config file (See FH_DEFUALT_USEARRAYKEY).

- $extra: This can be extra information which will be included in the fields html tag. You can add for example some CSS or javascript.

- $mask: With this argument you can position the field(s). In the given mask a the placeholder %field% will be replaced with the field. When the mask is full and there are still fields, the mask will be repeated. This works the same as SetMask.
When no mask is given the mask will be used which is set in the config file in the var FH_DEFAULT_GLUE_MASK.



Example:

```php
<?php

// the options for the radiobutton
$gender = array(
  "m" => "Male",
  "f" => "Female"
);

// include the class
include('FH3/class.FormHandler.php');

// create new formhandler object
$form = new FormHandler();

// make the radiobutton
$form -> radioButton("Gender", "gender", $gender);

// button beneath it
$form -> submitButton("Save");

// set the 'commit after form' function
$form -> onCorrect("doRun");

// send the form to the screen
$form -> flush();

// the function witch handles the code after the form
function doRun($data)
{
   // show the selected gender
   echo "Hello, you are a ";

   switch($data["gender"])
   {
     case "m":
       echo "boy.";
       break;
     case "f":
       echo "girl.";
       break;
     default:
       echo "shemale!! Whaaaaa...";
       break;
    }
}

?>
```

If you want to set a default selected option, you can use the function SetValue.

In the example above that sould be:

```php
<?php

// default select the male option
$form->SetValue("gender", "m");

?>
```

3.1.8 UploadField

This function generates a UploadField.

Only the filename is saved, not the path! (When using the database option.)

The extension of the file is automaticly checked by javascript (if FH_UPLOAD_JS_CHECK is set to true (default)), so that the visitor don't have to wait until his file is uploaded!

Mime types

By default, FormHandler checks the mime type of the uploaded file. For this, the file FH3/includes/mimeTypes.php is used. By doing this, it is not possible to upload a php file by renaming it to .jpg (for example).
However, it can be that the build-in formhandler mime type check is too strict. In this case, you can specify which mime types are allowed.

You can specify the mime types which are (also) allowed in 2 ways:
- match all
- per file extension

If you don't want to set the mime types per extension, all the extensions will be checked for this mime type. You can do this as follows:

```
<?php
// setting mime types for all extensions
$config = array(
  "type" => "jpg jpeg jpe"
  "mime" => "image/jpeg image/jpg"
);

// or like this
$config = array(
  "type" => "jpg jpeg jpe"
  "mime" => array("image/jpeg", "image/jpg")
);

?>
```

If you want to specify mime types per extension, it can be done like this:

```php
<?php
// setting mime types per file extension
$config = array(
  "type" => "jpg gif png",
  "mime" => array(
    "jpg" => "image/jpeg image/jpg",
    "gif" => "image/gif",
    "png" => "image/png"
  )
);

// or like this
$config = array(
  "type" => "jpg gif png",
  "mime" => array(
    "jpg" => array("image/jpeg", "image/jpg"),
    "gif" => array("image/gif"),
    "png" => array("image/png")
  )
);

?>
```

The arguments are:
- $title: Title of the field

- $name: Name of the field

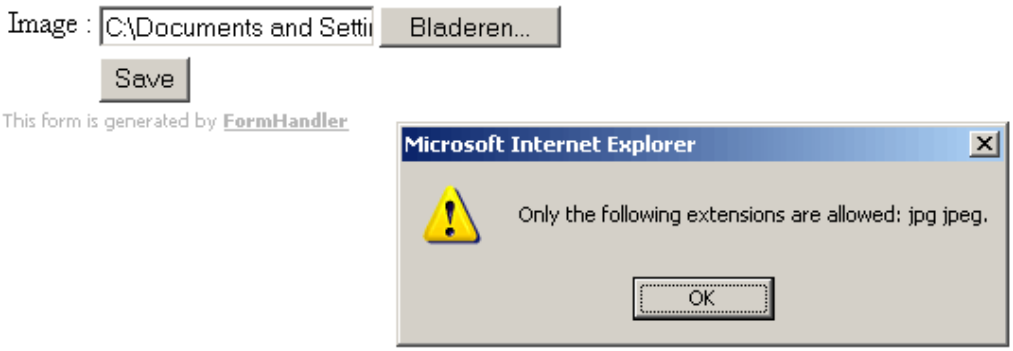
- $config: The configuration options for the upload field. The default options are defined in the config.inc.php file with the var FH_DEFAULT_UPLOAD_CONFIG.

- **path -** Directory name where the file must be uploaded (default: "./uploads").
- **type -** The types which are permitted seperated by a space(like: "jpg gif png")
- **mime -** The allowed mime types. For more information about the mime types see section below.
- **size -** The maximum size of the image (in bytes). Default this is the maximum permitted size (from the php.ini).
- **name -**The name of the file (without extension!). When no name is set, the original name is kept.
- **width -** The maximum allowed width of the uploaded image. This option will only be used on images!
- **height -** The maximum allowed height of the uploaded image. This option will only be used on images!
- **required -** if the field is required or not (default: false).
- **exists -** What to do when the file exists:
  "overwrite" -> overwrite the current file
  "rename" -> rename the new file
  "alert" -> show an alert message and do not upload the file

- $validator: This can either be the name of your own validation function or the constant name of a predefined validator function. For more information about validators see Validators.
It is also possible to use a method as a validation function. In this case you should pass an array where the first item is the object and the second item the name of the method.

- $extra: This can be extra information which will be included in the fields html tag. You can add for example some CSS or javascript.

- $alertOverwrite: When using the database option and there was already a file uploaded (saved in the field), should a message being showed that uploading a new file will overwrite the current value.



Example:

Example: config array

```
<?php
$config = array (
  "path"       => "./uploads",
  "type"       => "jpg jpeg png gif",
  "mime"       => array(
    "jpg" => "image/jpeg image/jpg",
    "png" => "image/png",
    "gif" => "image/gif"
  ),
  "size"       => 1000,
  "name"       => "test",
  "width"      => 800,
  "height"     => 600,
  "required"   => false,
  "exists"     => "rename"
);
?>
```

Example:

```php
<?php

// include the class
include('FH3/class.FormHandler.php');

// make a new $form object
$form = new FormHandler();

// The upload configuration
// NOTE: You dont have to set every value!
// Like below, we have not set the "size", so the default configuration
// value is used (max size which is possible).
$cfg = array(
  "path"      => $_SERVER['DOCUMENT_ROOT'].dirname($_SERVER['PHP_SELF']).'/uploads/images',
  "type"      => "jpg jpeg",
  "name"       => "", // <-- keep the original name
  "required"   => true,
  "exists"     => "rename"
);

// upload field
$form -> uploadField("Image", "image", $cfg);

// buttons
$form -> submitButton("Save");

// set the 'commit after form' function
$form -> onCorrect("doRun");

// flush the form
$form -> flush();

// the 'commit after form' function
function doRun($data)
{
   echo
   "The image is saved: \n".
   "<img src='uploads/images/". $data["image"] ."'>\n";
}

?>
```

3.1.9 ListField

This function generates a ListField (two select fields where you can move values from one to another)

When you double click on the buttons < or > all items are moved to the other field. When you double click on an item it will be moved to the other side.

This function returns an array as value. When the value has to be saved into a database the values are comma seperated inserted (like: option1, option2, option3, etc..).

The arguments are:
- $title: Title of the field

- $name: Name of the field

- $options: The options for the field

- $useArrayKeyAsValue: Should the key of the array be used as the value for the field? If not (false) the array value is used. The default option for this can be set in the config file (See FH_DEFUALT_USEARRAYKEY).

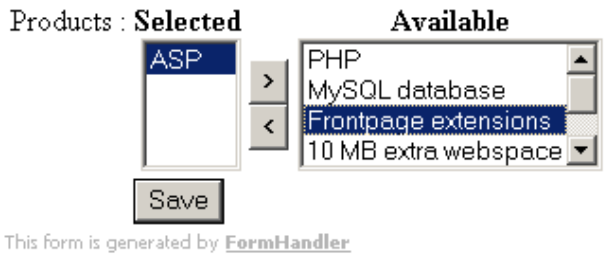
- $onTitle: The title which will be displayed above the field where the items are in which will be saved. By default this is "Selected" but is language depented.

- $offTitle: The title which will be displayed above the field where the items are displayed where the user can select from. By default this is "Available" but is language depented.

- $size: The number of items which are displayed (the height of the field)

- $extra: This can be extra information which will be included in the fields html tag. You can add for example some CSS or javascript.

- $verticalMode: indicates whether the two selectfields should be stacked horizontally or vertically.



Example:

```php
<?php

// the values for the listfield
$values = array(
  1 => "PHP",
  2 => "MySQL database",
  3 => "Frontpage extensions",
  4 => "ASP",
  5 => "10 MB extra webspace",
  6 => "Webmail",
  7 => "Cronjobs"
);

// include the class
include('FH3/class.FormHandler.php');

// new $form object
$form = new FormHandler();

// the listfield
$form->ListField("Products", "products", $values);

// buttons beneath it
$form->SubmitButton("Save");

// set the 'commit after form' function
$form->OnCorrect("doRun");

// flush the form
$form->Flush();

// function to show the selected items
function doRun($data) {
    global $values;

    $message = "You have selected the following products:\n";

    foreach($data["products"] as $id) {
        $message .= " - ".$values[$id]." \n";
    }

    return $message;
}

?>
```

3.1.10 Editor

With this function you can generate a online Text Editor (FCKEditor):

The Editor is compatible with most internet browsers which include: IE 5.5+, Firefox 1.0+, Mozilla 1.3+ and Netscape 7+.
Browsers which are not compatible with the Editor will see a normal TextArea.

Note: When submitting vary much data please dont use the FH_TEXT validator. The validator is to time/memory expensive and submitting will fail. For small and normal amounts of data you can just use FH_TEXT.

Make sure the path to FCKeditor is correct. You can change the the path in the config file:
fh_conf('FH_FHTML_DIR', 'path/to/your/dir/FH3/FHTML/');

It is also possible to make your own toolbar for the editor. You can define this in the file FHTML/FCKeditor/fckconfig.js and after defining it you can use it as argument.

Since version RC2 the argument skin is available. Also is the argument path moved 1 place forward, before toolbar.

The arguments are:
- $title: Title of the field

- $name: Name of the field

- $validator: This can either be the name of your own validation function or the constant name of a predefined validator function. For more information about validators see Validators.
It is also possible to use a method as a validation function. In this case you should pass an array where the first item is the object and the second item the name of the method.

- $path: The path where images can be browsed and uploaded. This directory will also be used for browsing for links, flash files and other media.
Set to false if you want to disable this feature.

The path has to be a relative path from the dir where your script is located. **Don't** use complete paths like this: /home/sites/etc/.
The dir which is used for browsing should be located in or below the web root.
Example:

correct!
- uploads
- /uploads
- uploads/
- ../uploads
- ./uploads
wrong!
- /home/sites/site1234/web/uploads/
- http://www.mysite.com/uploads/

- C:\Program Files\apache\htdocs\uploads

- $toolbar: The toolbar which is shown. You can configure the toolbars in the FH3/FHTML/FCKeditor/fckconfig.js file.

- $skin: The skin used for the toolbar (the style of the buttons etc.). You can use "default", "silver" or "office2003".

This argument has been added since version RC2 and the argument $path has been moved one place forward, before $toolbar.

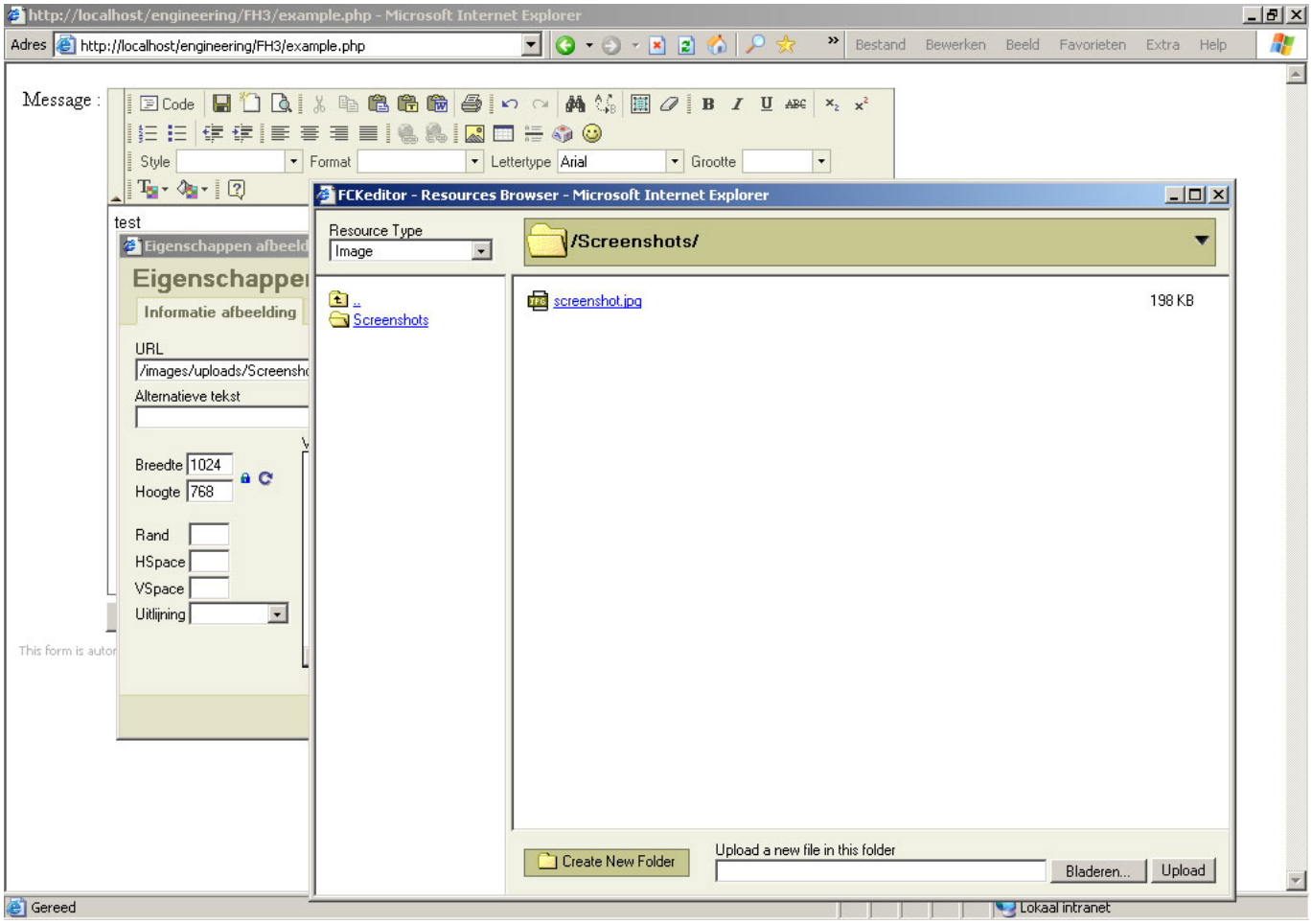
- $width: The width of the editor in pixels

- $height: The height of the editor in pixels

- $config: In this array you can set some config values for the editor. You can overwrite the config vars defined in the FH3/FHTML/FCKeditor/fckconfig.js file. The array key has to be the config var name. Example:

```
<?php
$config = array(
  "StartupFocus" => true
);

?>
```

This function was added in FH3 v1.2.



Example:

```
<?php

// include the class
include('FH3/class.FormHandler.php');

// make a new formhandler object
$form = new FormHandler();

// make the editor
$form -> editor("Message", "message", null, "images/uploads/");

// button beneath it
$form -> submitButton("Save");

// set the 'commit after form' function
$form -> onCorrect("doRun");

// send the form to the screen
$form -> flush();

// the function witch handles the code after the form
function doRun( $data )
{
   // show the data
   return $data["message"];
}

?>
```

3.1.11 DateField

This function generates a DateField.

The arguments are:
- $title: Title of the field

- $name: Name of the field

- $validator: This can either be the name of your own validation function or the constant name of a predefined validator function. For more information about validators see Validators.
It is also possible to use a method as a validation function. In this case you should pass an array where the first item is the object and the second item the name of the method.

(**NOTE:** the date is automaticly checked for a correct day-month-year combination)

- $required: When the field is required, the user has to select something. If the field is not required, the user can select empty values.

- $mask: The mask which will be used to lay out the date fields. If no mask is given, the default mask is loaded which is set in the config file with the var FH_DATEFIELD_DEFAULT_DISPLAY.

In the mask the following placeholders are replaced:

- **d** = selectfield day
- **m** = selectfield month
- **y** = selectfield year
- **D** = textfield day
- **M** = textfield month
- **Y** = textfield year

The date will be changed to the correct format when it's being saved into the database. The date will have the same format as given in the mask when it's returned to the user (in your onSaved or OnCorrect function)

- $interval: The interval between the current year and the years to start/stop. By default, the years are starting at 90 yeas from now. It is also possible to have years in the future. This is done like this: "90:10" (10 years in the future).

- $extra: This can be extra information which will be included in the fields html tag. You can add for example some CSS or javascript.

Example:

```php
<?php

// include the class
include('FH3/class.FormHandler.php');

// make a new $form object
$form = new FormHandler();

// make the datefield
$form -> dateField("Birthdate", "birthdate");

// button beneath it
$form -> submitButton("Save");

// set the 'commit after form' function
$form -> onCorrect("doRun");

// send the form to the screen
$form -> flush();

// the function witch handles the code after the form
function doRun($data)
{
  // show the birthdate
  echo "Your birthday is ". $data["birthdate"];
}

?>
```

3.1.12 jsDateField

(Since version FH3-v1.1)

With this function you can create a date field with a jsCalendar so that the user can easily select a date.

The usage is exactly the same as a normal DateField, only this one comes with a js calendar!

The arguments are:
- $title: Title of the field

- $name: Name of the field

- $validator: This can either be the name of your own validation function or the constant name of a predefined validator function. For more information about validators see Validators.
It is also possible to use a method as a validation function. In this case you should pass an array where the first item is the object and the second item the name of the method.

(**NOTE:** the date is automaticly checked for a correct day-month-year combination)

- $required: When the field is required, the user has to select something. If the field is not required, the user can select empty values.

- $mask: The mask which will be used to lay out the date fields. If no mask is given, the default mask is loaded which is set in the config file with the var FH_DATEFIELD_DEFAULT_DISPLAY.

NOTE: The jsDateField will only work when d, m and y are used in the mask! If they are not all present, the field will be displayed as a normal DateField.

In the mask the following placeholders are replaced:

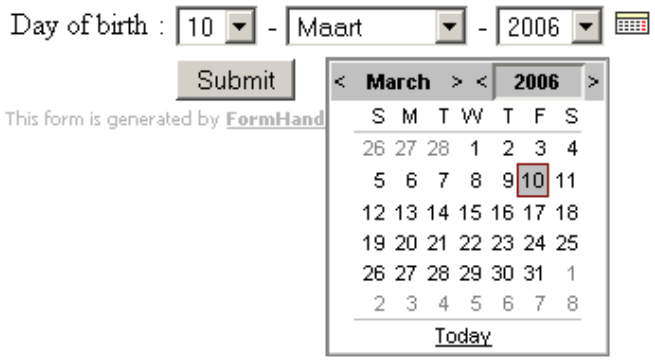
- **d** = selectfield day
- **m** = selectfield month
- **y** = selectfield year
- **D** = textfield day
- **M** = textfield month
- **Y** = textfield year

The date will be changed to the correct format when it's being saved into the database. The date will have the same format as given in the mask when it's returned to the user (in your onSaved or OnCorrect function)

- $interval: The interval between the current year and the years to start/stop. By default, the years are starting at 90 yeas from now. It is also possible to have years in the future. This is done like this: "90:10" (10 years in the future).

- $extra: This can be extra information which will be included in the fields html tag. You can add for example some CSS or javascript.

- bIncludeJS: Should we include javascript?
set to false on a next form on a page.



3.1.13 DateTextField

(Since version FH3 v1.2.12)

This function generates a textfield which will be validated as date

The arguments are:
- $title: The title of the field

- $name: The name of the field

- $validator: This can either be the name of your own validation function or the constant name of a predefined validator function. For more information about validators see Validators.
It is also possible to use a method as a validation function. In this case you should pass an array where the first item is the object and the second item the name of the method.

(NOTE: the date is automaticly checked for a correct combination)

- $mask: The mask which will be used to lay out the date fields. If no mask is given, the default mask is loaded which is set in the config file with the var FH_DATETEXTFIELD_DEFAULT_DISPLAY.

In the mask the following placeholders are replaced:
d = day (2 digits with leading zeros)
D = day
m = month (2 digits with leading zeros)
M = month
y = year (two digits)
Y = year (four digits)

The date will be changed to the correct format when it's being saved into the database. The date will have the same format as given in the mask when it's returned to the user (in your onSaved or OnCorrect function)

- $bParseOtherPresentations: try to parse other presentations of dateformat

- $extra: This can be extra information which will be included in the fields html tag. You can add for example some CSS or javascript.

Example:

```php
<?php
// include the class
include('FH3/class.FormHandler.php');

// make a new form object
$oForm = new FormHandler();

// make the datefield
$oForm -> dateTextField("Birthdate", "birthdate");

// button beneath it
$oForm -> submitButton("Save");

// set the 'commit after form' function
$oForm -> onCorrect("doRun");

// send the form to the screen
$oForm -> flush();

// the function witch handles the code after the form
function doRun($aData)
{
  // show the birthdate
  echo "Your birthday is ". $aData["birthdate"];
}
?>
```

3.1.14 jsDateTextField

(Since version FH3 v1.2.12)

With this function you can create a date text field with a jsCalendar so that the user can easily select a date.

The usage is exactly the same as a normal DateTextField, only this one comes with a js calendar!

The arguments are:
- $title: The title of the field

- $name: The name of the field

- $validater: This can either be the name of your own validation function or the constant name of a predefined validator function. For more information about validators see Validators.
It is also possible to use a method as a validation function. In this case you should pass an array where the first item is the object and the second item the name of the method.

(NOTE: the date is automaticly checked for a correct combination)

- $mask: The mask which will be used to lay out the date fields. If no mask is given, the default mask is loaded which is set in the config file with the var FH_DATETEXTFIELD_DEFAULT_DISPLAY.

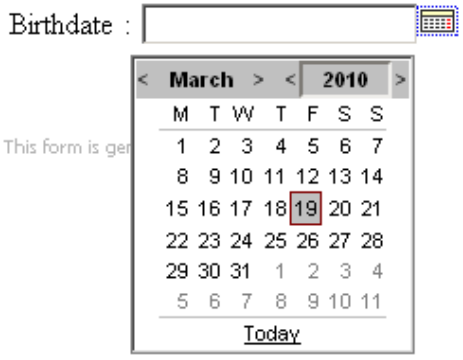
In the mask the following placeholders are replaced:
d = day (2 digits with leading zeros)
D = day
m = month (2 digits with leading zeros)
M = month
y = year (two digits)
Y = year (four digits)

The date will be changed to the correct format when it's being saved into the database. The date will have the same format as given in the mask when it's returned to the user (in your onSaved or OnCorrect function)

- $bParseOtherPresentations: try to parse other presentations of dateformat

- $extra: This can be extra information which will be included in the fields html tag. You can add for example some CSS or javascript.

- $bIncludeJS: Should we include the js file (only needed once on a page)



Example:

```php
<?php
// include the class
include('FH3/class.FormHandler.php');

// make a new form object
$oForm = new FormHandler();

// make the datefield
$oForm -> jsDateTextField("Birthdate", "birthdate");

// button beneath it
$oForm -> submitButton("Save");

// set the 'commit after form' function
$oForm -> onCorrect("doRun");

// send the form to the screen
$oForm -> flush();

// the function witch handles the code after the form
function doRun($aData)
{
  // show the birthdate
  echo "Your birthday is ". $aData["birthdate"];
}
?>
```

3.1.15 TimeField

This function generates a TimeField.

Note: The steps of the minutes field (eg, 10, 20, 30 or 5, 10, 15, etc.) can be changed in the config file!

The arguments are:
- $title: Title of the field

- $name: Name of the field

- $validator: This can either be the name of your own validation function or the constant name of a predefined validator function. For more information about validators see Validators.
It is also possible to use a method as a validation function. In this case you should pass an array where the first item is the object and the second item the name of the method.

- $required: When the field is required, the user has to select something. If the field is not required, the user can select empty values.

- $format: Which time format should be used? Possible values: 12, 24.

- $extra: This can be extra information which will be included in the fields html tag. You can add for example some CSS or javascript.

Example:

```php
<?php

// include the class
include('FH3/class.FormHandler.php');

// make a new formhandler object
$form = new FormHandler();

// a timefield
$form -> timeField("Time", "time");

// button beneath it
$form -> submitButton("Save");

// set the 'commit after form' function
$form -> onCorrect("doRun");

// send the form to the screen
$form -> flush();

// the function witch handles the code after the form
function doRun($data)
{
   echo "Your time: " . $data["time"];
}

?>
```

3.1.16 BrowserField

(Since version FH3 v1.2.2)

Creates a textfield with a browse button which invokes the FCK filemanager to insert a filename

The arguments are:
- $title: Title of the field

- $name: Name of the field

- $path: The path where images can be browsed and uploaded

- $validator: This can either be the name of your own validation function or the constant name of a predefined validator function. For more information about validators see Validators.
It is also possible to use a method as a validation function. In this case you should pass an array where the first item is the object and the second item the name of the method.

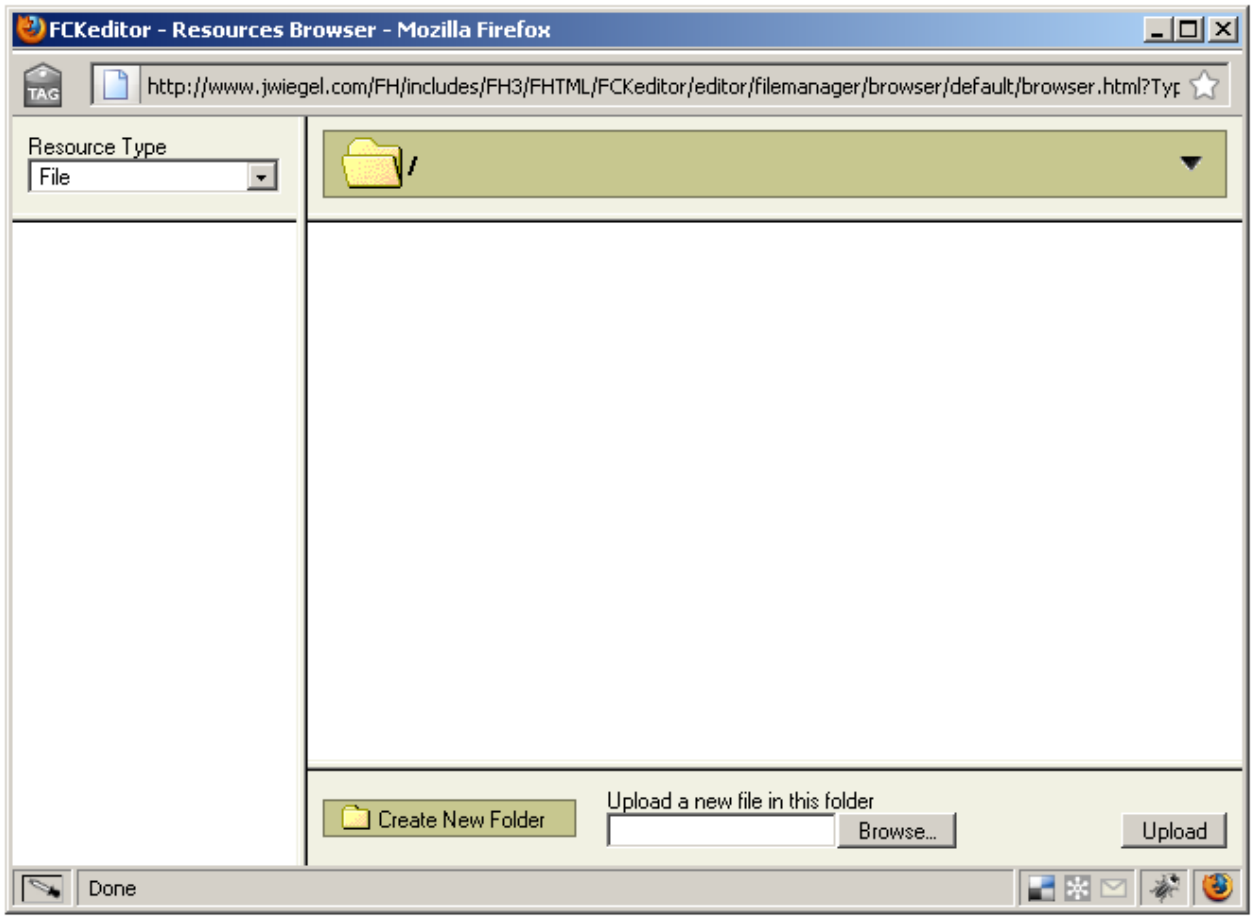
- $size: The size of the field.

- $extra: This can be extra information which will be included in the fields html tag. You can add for example some CSS or javascript.

Image : Bladeren

Submit

This form is generated by FormHandler



Example:

```php
<?php

// include the class
include('FH3/class.FormHandler.php');

// make a new formhandler object
$form = new FormHandler();

// make the browser field
$form->BrowserField('Image','image', "/uploads/Image");

// button beneath it
$form -> submitButton("Save");

// set the 'commit after form' function
$form -> onCorrect("doRun");

// send the form to the screen
$form -> flush();

// the function witch handles the code after the form
function doRun( $data )
{
   // show the data
   return $data["image"];
}

?>
```

3.1.17 ColorPicker

(Since version FH3 v1.2.5)

This function will create a colorpicker on the form.

The arguments are:
- $title: Title of the field

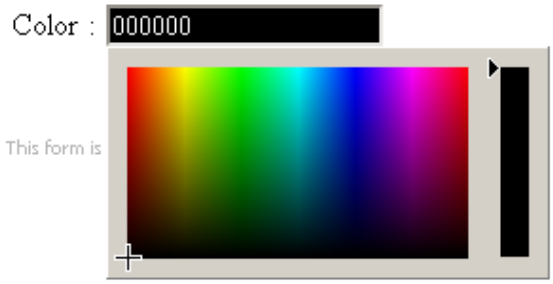
- $name: Name of the field

- $validator: This can either be the name of your own validation function or the constant name of a predefined validator function. For more information about validators see Validators.
It is also possible to use a method as a validation function. In this case you should pass an array where the first item is the object and the second item the name of the method.

- $size: size The size of the field.

- $maxlength: The maximum allowed characters in this field. 0 is interpretated as no limit.

- $extra: This can be extra information which will be included in the fields html tag. You can add for example some CSS or javascript.



Example:

```php
<?php

// include the class
include('FH3/class.FormHandler.php');

// make a new $form object
$form = new FormHandler();

// a ColorPicker
$form->colorpicker("Color", "color");

// submitbutton beneath it
$form->submitButton("Save");

// set the 'commit after form' function
$form->onCorrect("doRun");

// send the form to the screen
$form->flush();

// function to show a message
function doRun($data)
{
   return "You picked this color code". $data["color"];
}
?>
```

3.1.18 TextSelectField

(Since version FH3 v1.2.6)

With this field you have the possibility to present a user a selection to put in the textfield or type there own value

The arguments are:
- $title: Title of the field

- $name: Name of the field

- $aOptions: options for this fields select part

- $validator: This can either be the name of your own validation function or the constant name of a predefined validator function. For more information about validators see Validators.
It is also possible to use a method as a validation function. In this case you should pass an array where the first item is the object and the second item the name of the method.

- $size: size The size of the field.

- $maxlength: The maximum allowed characters in this field. 0 is interpretated as no limit.

- $extra: This can be extra information which will be included in the fields html tag. You can add for example some CSS or javascript.

Example:

```php
<?php

// include the formhandler
include( 'FH3/class.FormHandler.php' );

// create new formhandler object
$oForm = new Formhandler();

// set the options array
$aOptions = array( 'Red', 'Green' );

// new TextSelect field
$oForm->TextSelectField( 'Color', 'color', $aOptions );

// button to submit
$oForm->submitButton();

// what to do when the form is saved
$oForm->onCorrect( 'FH_RUN' );

// show the form
$oForm->Flush( );

// the function which is called when the data is saved
function FH_RUN( $aData )
{
   echo 'You selected or typed value '. $aData['color'];
}
?>
```

3.1.19 CaptchaField

(Since version FH3 v1.2.7)

This function will create a captcha field and takes care of the validation.

The arguments are:
- $title: Title of the field

- $name: name of the field

- $size: The size of the field

- $maxlength: The maximum allowed characters in this field. 0 is interpretated as no limit.

- $extra: This can be extra information which wull be included in the fields html tag. You can add for example some CSS or javascript.

Example:

```php
<?php

// include the class
include('FH3/class.FormHandler.php');

// make a new $oForm object
$oForm = new FormHandler();

// a textfield
$oForm->CaptchaField("Verify the code", "code");

// submitbutton beneath it
$oForm->submitButton("Save");

// set the 'commit after form' function
$oForm->onCorrect("FH_RUN");

// send the form to the screen
$oForm->flush();

// function to show a message
function FH_RUN($aData)
{
   return "You entered the correct code ". $aData["code"];
}
?>
```

3.2 Buttons

3.2.1 Button

Creates a button on the form.

The arguments are:
- $caption: The caption of the button

- $name: Name of the button. A name is generated if none is given.

- $extra: This can be extra information which will be included in the buttons html tag. You can add for example some CSS or javascript.

Example:

```
<?php

// include the class
include('FH3/class.FormHandler.php');

// create new formhandler object
$form = new FormHandler;

// the button
$form -> button("Test", "btnTest", "onclick='alert(this.name)'");

// what to do when the form is correct...
$form -> onCorrect('doRun');

// flush the form
$form -> flush();

// the commit after form function...
function doRun($data)
{
   // do something here...
}
?>
```

3.2.2 SubmitButton

Creates a submitbutton on the form.

The arguments are:
- $caption: The caption of the button. The default caption is "Submit", but this is language dependent.

- $name: Name of the button. A name is generated if none is given.

- $extra: This can be extra information which will be included in the buttons html tag. You can add for example some CSS or javascript.

- $disableOnSubmit: Should the button being disabled after submitting the form? (This prevents double posting)

Example:

```php
<?php

// include the class
include('FH3/class.FormHandler.php');

// create a new formhandler object
$form = new FormHandler();

// a textfield
$form -> textField("Name", "name", FH_STRING);

// submitbutton
$form -> submitButton();

// what to do when the form is correct?
$form -> onCorrect("doRun");

// flush the form
$form -> flush();

// the commit after form function
function doRun($data)
{
   // do something here
   echo "Hello " . $data['name'];
}
?>
```

3.2.3 ImageButton

Create a image button.

The arguments are:
- $image: The image URL which should be the button

- $name: Name of the button. A name is generated if none is given.

- $extra: This can be extra information which will be included in the buttons html tag. You can add for example some CSS or javascript.

Example:

```php
<?php

// include the class
include('FH3/class.FormHandler.php');

// create a new FormHandler object
$form = new FormHandler();

// textfield
$form -> textField("Name", "name", FH_STRING);

// image button!
$form -> imageButton("images/button.gif");

// set the oncorrect function
$form -> onCorrect("doRun");

// flush the form
$form -> flush();

// the commit after form function
function doRun( $data )
{
   return "Hello ".$data['name'];
}

?>
```

3.2.4 ResetButton

Create a resetButton on the form which resets the values of the fields

The arguments are:
- $caption: The caption of the button. The default caption is "Reset", but this is language dependent.

- $name: Name of the button. A name is generated if none is given.

- $extra: This can be extra information which will be included in the buttons html tag. You can add for example some CSS or javascript.

Example:

```php
<?php

// include the class
include('FH3/class.FormHandler.php');

// create new formhandler object
$form = new FormHandler;

// a textfield
$form -> textField("Name", "name", FH_STRING);

// the reset button
$form -> resetButton();

// what to do when the form is correct...
$form -> onCorrect('doRun');

// flush the form
$form -> flush();

// the commit after form function...
function doRun( $data )
{
   return "Hello ". $data['name'] ."\n";
}
?>
```

3.2.5 CancelButton

Creates a cancel button. This is just like a normal button only when you press it you will go to the given page.

The arguments are:
- $caption: The caption of the button. The default caption is "Cancel", but this is language dependent.

- $url: The url to go to when the cancel button is presed. By default, the browser will go to the previous page.

- $name: Name of the button. A name is generated if none is given.

- $extra: This can be extra information which will be included in the buttons html tag. You can add for example some CSS or javascript.

Example:

```php
<?php

// include the formhandler
include("FH3/class.FormHandler.php");

// create a new formhandler object
$form = new FormHandler("myForm");

// textfield
$form -> textField("Name", "name", FH_STRING);

// set a mask for some nice buttons row
$form -> setMask(
  "  <tr>\n".
  "    <td> </td>\n".
  "    <td> </td>\n".
  "    <td>%field% %field%</td>\n".
  "  </tr>\n"
);

// set button 1: a submit button
$form -> submitButton("Save");

// set button 2: the cancelbutton
// go to ../index.php when the button is pressed
$form -> cancelButton("Cancel", "../index.php");

// set the handler
$form -> onCorrect("doRun");

// flush it
$form -> flush();

// the handler...
function doRun( $data )
{
   echo "Hello ". $data["name"];
}

?>
```

3.2.6 BackButton

Creates a button which can be used in a multi-paged form to go 1 page back. Multi-paged forms are made with the function NewPage.

The button works just like a normal Button except that a predifened javascript event is attached to it.

For an example and preview image see the function NewPage.

The arguments are:
- $caption: The caption of the button. The default caption is "Back", but this is language dependent.

- $name: Name of the button. A name is generated if none is given.

- $extra: This can be extra information which will be included in the buttons html tag. You can add for example some CSS or javascript.

3.3 Look & Feel

3.3.1 SetMaxLength

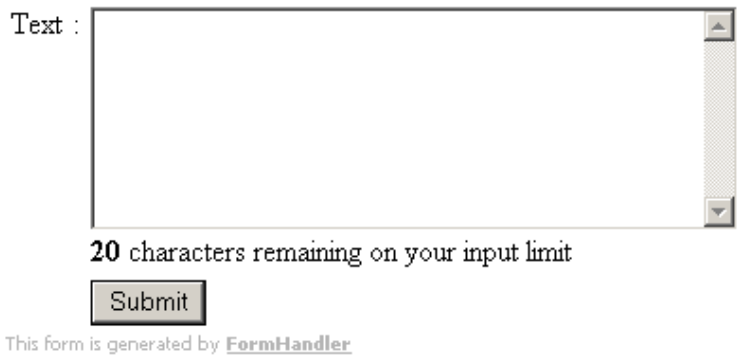
(Since version FH3-v1.1)

With this function you can limit the input of a textarea

The arguments are:
- $field: The name of the textarea which you want to limit.

- $maxlength: The maximum allowed input length of the textarea

- $displayMessage: Should a message being displayed to the user how many characters there are left to use in the textarea?



Example:

```php
<?php

// include the class
include_once('FH3/class.FormHandler.php');

// create a new formhandler object
$form = new FormHandler();

// Limited textarea
$form -> textArea("Text", "myTextArea");
$form -> setMaxLength("myTextArea", 20);

// button to submit the form
$form -> submitButton();

// set the handler
$form -> onCorrect("doRun");

// display the form
$form -> flush();

// the data handler
function doRun( $data )
{
   echo nl2br( $data["myTextArea"] );
}
?>
```

3.3.2 SetHelpText

With this function you can set a help message for a specific field.
The message is displayed with use of the overlib libary.

Set a help text (a hint for the user, description) which is displayed by the field. The help icon will be displayed where %help% is located in the mask which is used.

The arguments are:
- $field: The field where you want to add a help text to.

- $helpText: The description you want to display (Your help text message).

- $helpTitle: The title which is displayed above the help text description. %title% will be replaced with the title of the field.

Example:

```php
<?php

// include the class
include('FH3/class.FormHandler.php');

// create a new formhandler object
$form = new FormHandler();

// just a test field
$form -> textField("Test", "test", FH_STRING);

// set the help message for the field
$form -> setHelpText('test', 'Type here your help message!');

// button to submit the form
$form -> submitButton();

// set the data handler function
$form -> onCorrect("doRun");

// flush it
$form -> flush();

// the data handler function
function doRun( $data )
{
   //just display the entred msg
   echo $data['test'];
}

?>
```

3.3.3 SetHelpIcon

Set the icon which should be used to display the help text.

The arguments are:
- $helpIcon: The icon which you want to use to display the help text messages.

3.3.4 AddHTML

Add some HTML into the form. This HTML will not be inserted into a table-row!

The arguments are:
- $html: The html which you want to include in the form

Example:

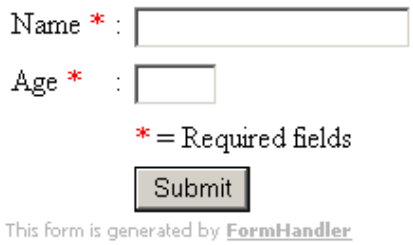
```php
<?php

// include the form
include("FH3/class.FormHandler.php");

// create a new formhandler object
$form = new FormHandler();

// simple field
$form -> textField("Name", "name", FH_STRING);

// addHTML!
$form -> addHTML(
  "  <tr>\n".
  "    <td colspan='3'><hr size='1' /></td>\n".
  "  </tr>\n"
);

// submit button
$form -> submitButton("Save");


// set the handler
$form -> onCorrect("doRun");

// flush it...
$form -> flush();

// the handler
function doRun( $data )
{
    //  do something here with the data...
    echo $data['name'];
}

?>
```

3.3.5 AddLine

This function adds a line (a new row) into the form table. There is one argument: $text. When some text is given, this is inserted into the line.

Note: The look of the line can be changed in the configuration file.

The arguments are:
- $text: The text you want to display.



Example:

```php
<?php

// include the class
include("FH3/class.FormHandler.php");

// create new formhandler object
$form = new FormHandler();

// star for required fields
$star = ' <font color="red">*</font>';

// some fields
$form -> textField("Name".$star, "name", FH_STRING, 20, 50);
$form -> textField("Age".$star, "age", FH_INTEGER, 3, 2);

// add a line that every field with a red * is required
$form -> addLine($star ."  = Required fields");

// button to submit the form
$form -> submitButton();

// set the data handler
$form -> onCorrect("doRun");

// display the form
$form -> flush();

// the data handler
function doRun( $data )
{
   echo
   "Hello ". $data["name"].",\n".
   "You are ". $data["age"] ." years old.\n";
}
?>
```

3.3.6 BorderStart

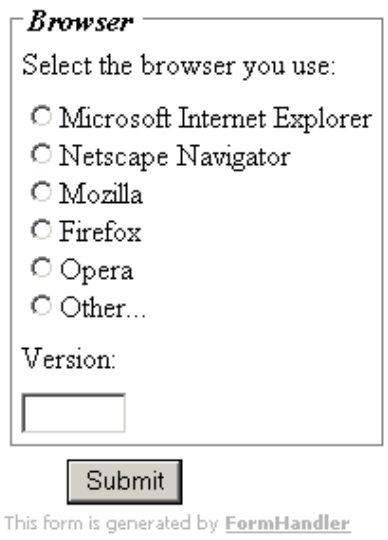
Sets a border arround the upcomming fields untill borderStop() is called. (Creates a fieldset.)

Note: The look of the fieldset can be changes in the configuration file (FH_FIELDSET_MASK)!

The arguments are:
- $caption: The caption of the fieldset.

- $name: The name of the fieldset. This name will also be used as id of the fieldset (in the HTML tag!), so you can use it in javascript. When no name is given, a name will be generated.

- $extra: This can be extra information which will be included in the fieldset's html tag. You can add for example some CSS or javascript.

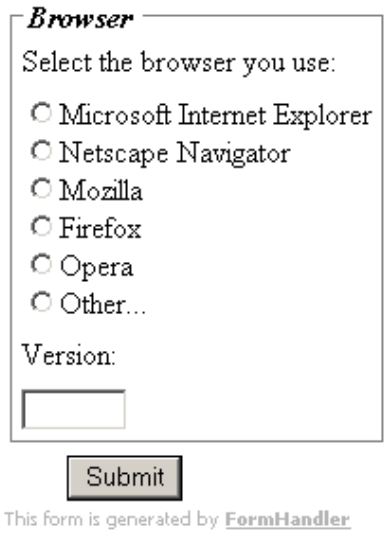


Example:

```php
<?php

// include the class
include("FH3/class.FormHandler.php");

// some options used in the form
$browsers = array(
  "IE"  => "Microsoft Internet Explorer",
  "NN"  => "Netscape Navigator",
  "MOZ" => "Mozilla",
  "FF"  => "Firefox",
  "OP"  => "Opera",
  "-1"  => "Other..."
);

// create new formhandler object
$form = new FormHandler();

// start a fieldset!
$form -> borderStart("Browser");

// set a mask for the upcoming field
$form -> setMask(
  "  <tr><td>%title%:</td></tr>\n".
  "  <tr><td>%field% %error%</td></tr>\n",
  1 # repeat it once (so for the upcoming 2 fields!!)
);

// browsers to select from
$form -> radioButton("Select the browser you use", "browsers", $browsers);
// which version of the browser?
$form -> textField("Version", "version", FH_FLOAT, 5, 5);

// stop the border
$form -> borderStop();

// button to submit the form
$form -> submitButton();

// set the data handler
$form -> onCorrect("doRun");

// display the form
$form -> flush();

// the data handler
function dorun( $data )
{
   // do something here...
}

?>
```

3.3.7 BorderStop

This function ends a border started with the function borderStart(). No arguments are expected.



Example:

See BorderStart for an example.

3.3.8 UseTable

When you use a mask/template which not exists of table rows, there is no need for adding a <table> tag around the data. You can disable (or enable) it with this function.

The arguments are:
- $setTable: Should a <table> tag been put around the fields? This is by default enabled.

Example:

For an example see example 2 at the function SetMask.

3.3.9 SetMask

With this function you can change to look of your form. Default, the form is set in a table. FormHandler sets each field in a seperate row. The default mask (can be edited in the config file!):

```
<tr>
<td valign='top'>%title%</td>
<td valign='top'>%seperator%</td>
<td valign='top'>%field% %help% %error%</td>
</tr>
```

As shown above, you have to set some special "words" which are replaced with the real value:

- **%title%** - Is replaced with the title of the field (the first argument of all fields)
- **%seperator%** - Is replaced with a ":". When no title is given, no seperator is placed.
- **%field%**- Is replaced with the field OR BUTTON!
- **%error%** - Is replaced with a possible error message when the field was invalid. When there is no error message, it will just be removed.
- **%name%** - Is replaced with the name of the field or button
- **%value%** - Is replaced with the value of the field
- **%help%** - Is replaced with a help icon. See for more information at setHelpText
- **%error_style%** - element with this "word" will get the error class added to any existing classes

When using a template file it's also possible to use

- **%header%** - is replaced with the start of the form ie <form> tag
- **%footer%** - is replaced with the end of the form ie </form> tag

The mask is repeated untill all the fields are processed. However, you can change this. If wanted, you could set 2 or more fields in one row and use this "layout" untill all fields are processed. It is even possible to fill one mask with all the %field% signs, so that you can use templates. How to do this is explained below.

Note: When the mask is not completly filled but all the fields are processed, the remaining %field% tags and so are removed from your mask!

The arguments are:
- $mask: The mask which you want to use for the oncomming fields. When no mask is given, the default mask is used (set in the config file with the var FH_DEFAULT_ROW_MASK).

- $repeat: Should the mask be repeated? When set to true (default), the mask will be repeated unlimited. When a integer is used, the integer will be countdown and the mask will be displayed as long as the number is greater then 0. After this the mask will be set back to the default mask.

Example:

```
<?php

// include the class
include("FH3/class.FormHandler.php");

// create new formhandler object
$form = new FormHandler();

// set another type of mask
$form -> setMask(
  "   <tr><td>%title% %seperator%</td></tr>\n".
  "   <tr><td>%field% %error%</td></tr>\n",
  true  # repeat this mask!
);

// some fields
$form -> textField("Name", "name", FH_STRING);
$form -> textField("Age", "age", FH_INTEGER, 3, 2);
$form -> selectField("Gender", "gender", array('M', 'F'), null, false);

// button to submit the form
$form -> submitButton();

// data handling
$form -> onCorrect("doRun");

// display the form
$form -> flush();

// data handler
function doRun( $data )
{
    // do something here!
    echo "Data recieved!";
}
?>
```

The result:

Some HTML code of the result (example of the mask!):

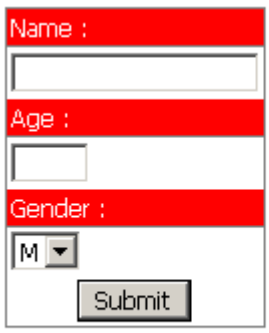
```
<!--
  This form is automaticly being generated by FormHandler v3.
  See for more info: http://www.formhandler.net
  This credit MUST stay intact for use
-->
<form name='FormHandler' method='post' action='/engineering/test.php'>
<input type="hidden" name="FormHandler_submit" id="FormHandler_submit" value="1" />
<table border='0' cellspacing='0' cellpadding='3'>
  <tr><td>Name :</td></tr>
  <tr><td><input type="text" name="name" id="name" value="" size="20" /> </td></tr>
  <tr><td>Age :</td></tr>
  <tr><td><input type="text" name="age" id="age" value="" size="3" /> </td></tr>
  <tr><td>Gender :</td></tr>
  <tr><td><select name="gender" id="gender" size="1">
```

Example 2: Using a template

Since FH3 v1.2.2 It is also possibnle to use a php file as a template the file will be parsed!

```php
<?php

// include the class
include("FH3/class.FormHandler.php");

// create new formhandler object
$form =& new FormHandler();

// set the template we are using
$form -> setMask(
  'templates/form.tpl',
  false # dont repeat this mask
);

// Let formhandler don't use the <table> tag
$form -> useTable( false );

// some fields
$form -> textField("Name", "name", FH_STRING);
$form -> textField("Age", "age", FH_INTEGER, 3, 2);
$form -> selectField("Gender", "gender", array('M', 'F'), null, false);

// button to submit the form
$form -> submitButton();

// data handling
$form -> onCorrect("doRun");

// display the form
$form -> flush();

// data handler
function doRun( $data )
{
    // do something here!
    echo "Data recieved!";
}
?>
```

The result:



This form is generated by FormHandler

The template used above contains html code:

```
<style>
.hdr {
  background-color: red;
  color: white;
  border-bottom: 1px solid gray;
  font-family: tahoma;
  font-size: 12px;
}
.tbl {
  border: 1px solid gray;
}
select, input {
  font-family: tahoma;
  font-size: 12px;
}


</style>

<table border="0" cellspacing="0" cellpadding="2" class="tbl">
  <tr><td class="hdr">%title% %seperator%</td></tr>
  <tr><td>%field% %error%</td></tr>
  <tr><td class="hdr">%title% %seperator%</td></tr>
  <tr><td>%field% %error%</td></tr>
  <tr><td class="hdr">%title% %seperator%</td></tr>
  <tr><td>%field% %error%</td></tr>
  <tr><td colspan="2"><center>%field%</center></td></tr>
</table>
<br />
```

3.3.10 SetErrorMessage

(Since version FH3-v1.1)

With this function you can set a custom error message for a specific field

The arguments are:
- $field: The field where you want to set a special error message for.

- $message: The error message you want to set for this field

- $useStyle: Should the error style set around the error message?

Example:

```
<?php
// include the class
include('FH3/class.FormHandler.php');

// create a new formhandler object
$form = new FormHandler();

// a textfield + custom error message!!!
$form -> textField("First Name", "fname", FH_STRING);
$form -> setErrorMessage( "fname", "You have to enter a first name!");

// button to submit the form
$form -> submitButton();

// what to do when the form is correct...
$form -> onCorrect('doRun');

// display the form
$form -> flush();

// do something...
function doRun( $data )
{
   echo "Hello ". $data["fname"];
}
?>
```

3.3.11 SetAutoComplete

With this function you can set some options for a specific textfield for auto completion.

The arguments are:
- $field: The textfield where you want to set some autocomplete options for.

- $options: List of options used for the auto complete.

Example:

```php
<?php

// include the class
include('FH3/class.FormHandler.php');

// create a new formhandler object
$form = new FormHandler();

// the auto complete items
$colors = array ( "red", "orange", "yellow", "green", "blue", "indigo", "violet", "brown", "rood" );
// the textfield used for auto completion
$form -> textField("Type a color", "color", FH_STRING);

// set the auto completion for the field Color
$form -> setAutoComplete("color", $colors);

// button to submit the form
$form -> submitButton();

// set the handler
$form -> onCorrect('doRun');

// display the form
$form -> flush();

// the data handler
function doRun( $data )
{
   echo "<pre>";
   print_r( $data );
   echo "</pre>";
}
?>
```

3.3.12 SetAutoCompleteAfter

(Since version FH3 v1.2.2)

With this function you can set some options for a specific textfield for auto completion after a specified character

The arguments are:
- $field: The textfield where you want to set some autocomplete options for.

- $after: The chracter after which we will start completion.

- $options: List of options used for the auto complete.

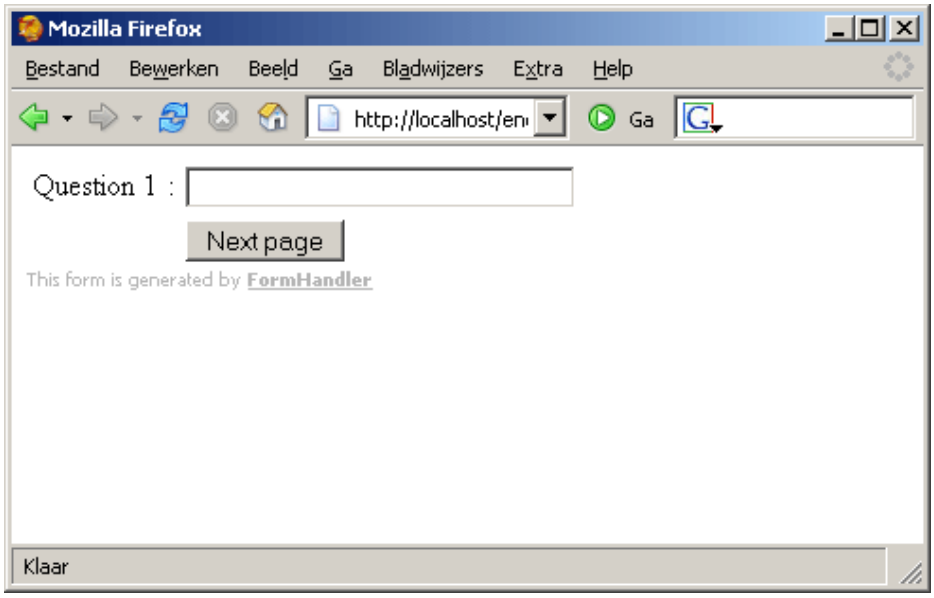
Example:

```
<?php

// include the class
include('FH3/class.FormHandler.php');

// create a new formhandler object
$form = new FormHandler();

// the auto complete items
$providers = array ( "hotmail.com", "live.com", "php-globe.nl", "freeler.nl" );
// the textfield used for auto completion after
$form -> textField("Type your email", "email", FH_STRING);

// set the auto completion for the field Color
$form -> setAutoCompleteAfter("email", "@", $providers);

// button to submit the form
$form -> submitButton();

// set the handler
$form -> onCorrect('doRun');

// display the form
$form -> flush();

// the data handler
function doRun( $data )
{
   echo "<pre>";
   print_r( $data );
   echo "</pre>";
}
?>
```

3.3.13 NewPage *

Note: This function does not work with edit forms!!

This function will split your form up in multiple pages. Very easy to generate wizard-like forms.



Example:

```php
<?php

// include the class
include("FH3/class.FormHandler.php");

// create a new form
$form = new FormHandler();

//first page...
$form -> textField("Question 1", "q1", FH_STRING, 30, 50);
$form -> submitButton("Next page");

// second page
$form -> newPage();
$form -> textArea("Question 2", "q2", FH_TEXT);
$form -> submitButton("Next Page");

// third and last page
$form -> newPage();
$form -> textField("Question 3", "q3", FH_STRING);
$form -> submitButton("Submit");

// set the handler function
$form -> onCorrect("doRun");

// flush the form..
$form -> flush();

// data handler
function doRun( $data )
{
   echo "Data submitted:";
   echo "<pre>\n";
   print_r( $data );
   echo "</pre>\n";
}

?>
```

3.3.14 SetTabIndex

With this function you can set the tabindexes of the field.
This function has one argument: $tabs.

You can set the tabindexes in two ways. You can give a string or an array as argument.
When you give a string as argument, the field names have to be comma seperated and in the right order.
When you give a array as argument, it has to contain the names of the fields (or buttons). The array keys are used as tab index.

Fields with negative tabindexes are not accessable with the tab key. Fields without tabindexes are accessable with the tab key after all the other fields.

The arguments are:
- $tabs: Array or comma seperated string with the field names. When an array is given the array index will set as tabindex.

Example:

```php
<?php

// include the class
include("FH3/class.FormHandler.php");

// create a new form
$form = new FormHandler();

// some fields + button
$form -> textField("Field 1", "fld1");
$form -> textField("Field 2", "fld2");
$form -> textField("Field 3", "fld3");
$form -> submitButton("Submit", "submitBtn");

// the tabs!
$tabs = array(
  3 => "fld1",
  1 => "fld2",
  2 => "fld3",
  4 => "submitBtn"
);

// set the tabs
$form -> setTabIndex($tabs);

/* // this is also correct!
* $form -> setTabIndex( "fld2, fld3, fld1, submitBtn" );
*/

// set the handler function
$form -> onCorrect("doRun");

// flush the form..
$form -> flush();

// data handler
function doRun( $data )
{
   // do something here...
}
?>
```

In the example above, a indexed array is used. You can also use an array like this:

```php
<?php

// the tabs!
$tabs = array(
  "fld2",
  "fld3",
  "fld1",
  "submitBtn"
);

?>
```

3.3.15 SetLanguage

Set the language used for displaying messages and other. By default, the language used by the visitors browser wil be set.

In the directory FH3/language you will find all the languages which are supported. To set a language, just give the name of the file without the ".php" extension (so "it.php" bekomes "it").

The arguments are:
- $language: The language you want to set. When no language is given, the language is set defined by the users browser (auto detect).

Example:

```php
<?php

// include the class
include('FH3/class.FormHandler.php');

// make a new $form object
$form = new FormHandler();

// set the language to dutch
$form -> setLanguage( 'nl' );

// a textfield  + submit button
$form -> textField    ("Your name", "name", FH_STRING);
$form -> submitButton ("Save");

// set the 'commit after form' function
$form -> onCorrect("doRun");

// send the form to the screen
$form -> flush();

// function to show a message
function doRun($data)
{
   return "Hello ". $data["name"];
}
?>
```

3.3.16 CatchErrors *

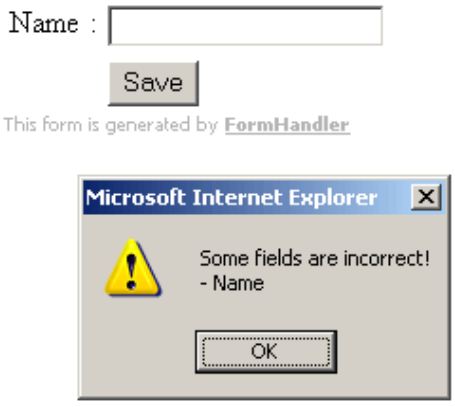
Get the error messages of the invalid fields in the form.

The error messages are returned in array format where the array keys are the names of the fields. This method has one argument: display the error messages in the form or not? Set display to false (default), and the error messages will not be displayed in the form (so you have to handle that yourself!). If set to true, the error messages will be displayed in the form.

KNOWN ERROR. Fetching the title with getTitle like in the example below does not work correctly when this code is still in the flush method:

```
<?php

// remove all vars to free memory
$vars = get_object_vars($this);
foreach( $vars as $name => $value )
{
   // remove all vars except these..
   if( !in_array($name, array( '_cache', 'edit', 'insert', '_posted', '_name' ) ) )
   {
      unset( $this->{$name} );
   }
}
?>
```

The arguments are:
- $display: Should there still be error messages displayed behind the invalid fields?



Example:

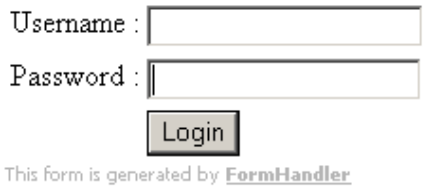
```
<?php

// include the class
include("FH3/class.FormHandler.php");

// create new formhandler object
$form = new FormHandler();

// textfield
$form -> textField("Name", "name", FH_STRING);

// submitbutton
$form -> submitButton("Save");

// get the errors of invalid fields
$errors = $form->catchErrors();

// oncorrect function...
$form -> onCorrect('doRun');

// flush
$form -> flush();

/** handle your own errors! **/

// any errors?
if( sizeof($errors) > 0 )
{
   // create a JS message
   $msg = "Some fields are incorrect!\\n";

   foreach($errors as $field => $error)
   {
      $msg .= "- ". $form -> getTitle( $field )."\\n";
   }
   echo
   "<script language='javascript'>\n".
   "alert('".$msg."');\n".
   "</script>\n";
}

function doRun($data)
{
   echo "Hi ". $data['name'];
}
?>
```

3.3.17 SetFocus

Set the focus to a specific field (So that the cursor is placed in a field).
Default, the focus will be set to the first field to the form. When the form is submitted but shown again (because some fields are not valid), the first of the invalid fields will get the focus.

This method has one argument: the name of the field which should get the focus.

The arguments are:
- $field: The field which should get the focus when the form is displayed



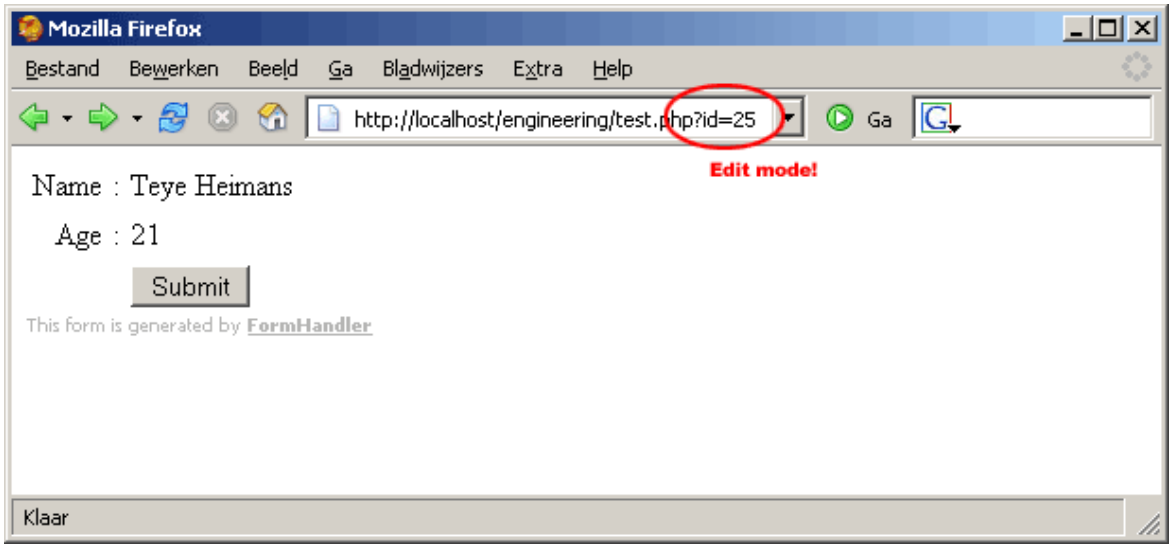
Example:

```php
<?php

// include the class
include("FH3/class.FormHandler.php");

// create new formhandler object
$form = new FormHandler();

// a textfield & passfield
$form -> textField("Username", "username", FH_STRING);
$form -> passField("Password", "password", FH_PASSWORD);

// set the focus to the password
$form -> setFocus("password");

// submit button below it..
$form -> submitbutton("Login");

// set the onCorrect function
$form -> oncorrect("doLogin");

// flush the form
$form -> flush();

// the commit after form function
function doLogin($data)
{
   if($data['username'] == 'admin' &&
      $data['password'] == 'passw')
   {
      // login here...
   }
   else
   {
      echo "Error, invalid username or password!";
      return false;
   }
}

?>
```

3.3.18 EnableViewMode

This function enables the view mode for all fields. All the fields are removed from the from and their values are displayed instead.

The arguments are:
- $mode: The mode. Should all fields set the "View Mode"?



Example:

```php
<?php

// include the class
include('FH3/class.dbFormHandler.php');

// create new dbFormHandler object
$form = new dbFormHandler();

// set the database info
$form -> dbInfo( 'myDatabase', 'myTable' );
$form -> dbConnect( 'localhost', 'myUsername', 'myPassword' );

// on edit mode, just display the data
if( $form -> edit )
{
   $form -> enableViewMode();
}

// the fields
$form -> textField( 'Name', 'Name', FH_STRING );
$form -> textField( 'Age', 'Age', FH_DIGIT );

// button to submit the form
$form -> submitButton();

// what to do when the form is saved
$form -> onSaved('showMessage');

// display the form
$form -> flush();

// the function which is runned when the form is saved
function showMessage( $id, $data )
{
   echo
   "Hello ". $data['Name'] ."\n".
   "Your data is saved with id ". $id."\n";
}

?>
```

3.3.19 IsViewMode

This function checkes if viewMode is enabled or not *for the whole form*. You can enable or disable view mode with the function enableViewMode.

Note: To check if a specific field is set to viewMode you should use the function isFieldViewMode.

Example:

```php
<?php

// include FormHandler
include ('FH3/class.dbFormHandler.php' );

// create new dbformhandler object
$form = new dbFormHandler();

// set the database options
$form -> dbInfo( 'test', 'test' );
$form -> dbConnect( 'localhost','root', '');

// enable view mode for the whole form when it's an edit form!
if( $form -> edit )
{
   $form -> enableViewMode();
}

// field
$form -> textField("Your name", "name" );

// submitbutton... only show when the form is not set to view mode!!!
if( !$form -> isViewMode() )
{
   $form -> submitButton();
}

// what to do when the form is correct
$form -> onCorrect( 'doRun' );

// show the form
$form -> flush();

// the function which is runned when the form is correct
function doRun( $data )
{
   // show the submitted data
   echo "<pre>";
   print_r( $data );
   echo "</pre>";

}

?>
```

3.3.20 SetFieldViewMode

(Since version FH3-v1.2)

With this function you can set a specific field to view mode. When a field is set to view mode, only the value is displayed.

The arguments are:
- $field: The field which you want to set to "ViewMode"

- $mode: Do you want to set the field to "ViewMode"?

Example:

```
<?php

// include the formhandler
include( 'FH3/class.FormHandler.php' );

// create new FormHandler object
$form = new FormHandler();

// add some fields
$form -> textField("Name", "name", FH_STRING );

// username
$form -> textField("Username", "username", FH_STRING );

// view mode for the username in edit mode
if( $form -> edit )
{
   $form -> setFieldViewMode( "username" );
}

// button to submit the form
$form -> submitButton();

// what to do when the form is correct
$form -> onCorrect( "doRun" );

// display the form
$form -> flush();

// the function which is called when the form is correct
function doRun( $data )
{
   echo "Hello ". $data['name'];
}

?>
```

3.3.21 IsFieldViewMode

(Since version FH3-v1.2)

This function will check if a specific field is set to view mode.

Note: This function will only return true if the field is set to view mode with the function setFieldViewMode.

The arguments are:
- $field: The field which you want to get the "ViewMode" status from

3.3.22 SetTableSettings

(Since version FH3-v1.2)

With this function you can change the look of the table where the fields are displayed in.

Note: If you don't want FormHandler to set a <table> tag arround the content, please use the function useTable to disable it.

The arguments are:
- $width: The width of the table where the fields are set in. The default value of the table is set in the config file with the var FH_DEFAULT_TABLE_WIDTH.

- $cellspacing: The space between the cells in the table. The default value is set in the configuration file with the var FH_DEFAULT_TABLE_CELLSPACING.

- $cellpadding: The space between the cell border and the cell contents. The default value is set in the configuration file with the var FH_DEFAULT_TABLE_CELLPADDING.

- $border: The thickness of the border of the table. The default value is set in the configuration file with the var FH_DEFAULT_TABLE_BORDER.

- $extra: This can be extra information which will be included in the table's html tag. You can add for example some CSS or javascript.

Example:

```php
<?php

// include the class
include("FH3/class.FormHandler.php");

// make a new $form object
$form = new FormHandler;

// set the table settings
$form -> setTableSettings(400, 2, 2, 0, ' style="border: 1px solid green"' );

// a textfield
$form -> textField("Your name", "name", FH_STRING );

// button beneath it
$form -> submitButton();

// set the 'commit after form' function
$form -> onCorrect( "doRun" );

// send the form to the screen
$form-> flush();

// function to show a message
function doRun($data)
{
   return "Hello ". $data["name"];
}

?>
```

3.4 Data handling

3.4.1 GetAsArray

(Since version FH3-v1.2)

This function returns the value of a date field as an array. The array will have three items:

Array (
0 => year,
1 => month,
2 => day
)

See example below.

The arguments are:
- $datefield: The datefield where you want the values from

Example:

```php
<?php

// include the class
include('FH3/class.FormHandler.php');

// new form object
$form = new FormHandler();

// datefield
$form -> dateField( 'Date', 'date' );

// only when the form is posted..
if( $form -> isPosted() )
{
   // get the value from the field
   list( $year, $month, $day ) = $form -> getAsArray( 'date' );

   echo
   "Day: ". $day ."\n".
   "Month: ". $month ."\n".
   "Year: ". $year ."\n";
}

// submitbutton
$form -> submitButton();

// which function to run when the form is correct
$form -> onCorrect( 'doRun' );

// display the form
$form -> flush();

// the function which is called when the form is correct
function doRun( $data )
{
   // do something here..
   echo "Selected date: ". $data['date'] ."\n";
}

?>
```

3.4.2 Value

Return the value of the field which fieldname is given to this function.

When the database option is used, it's also possible to get a value from the database (also when the field is not in the form!).

The arguments are:
- $field: The field where you want to get the value from

Example:

Simple example of retrieving a value from a field and using it.

```
<?php

// include the class
include("FH3/class.FormHandler.php");

// create new formhandler object
$form = new FormHandler();

// passfield
$form -> passField("Password", "password", FH_STRING);

// get the value from the field
$value = $form -> value("password");

// save the password MD5 encrypted
// SO OVERWRITE THE CURRENT VALUE!
$form -> addValue("password", md5( $value )
);

// submitbutton
$form -> submitButton("Save");

// set the onCorrect function
$form -> onCorrect("doRun");

// flush
$form -> flush();

// commit after form function
function doRun( $data )
{
   echo "MD5 encrypted password: ".$data['password'];
}

?>
```

3.4.3 GetValue

This function is an alias for the function Value.

The arguments are:
- $field: The field where you want to get the value from

Example:

For an example see the function Value.

3.4.4 SetValue

This function works on all fields! Needs to be placed before the field in order to keep the altered value in a multipage form!

In fields which can have multiple values (like a listfield), you can select multiple items in two ways:

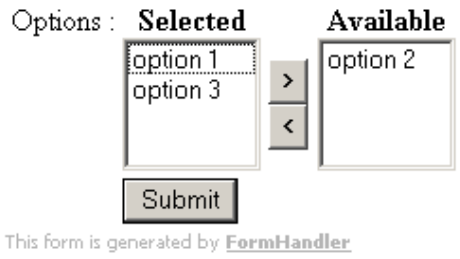
1. Give an array
2. Give the items seperated with a comma (,)

See example 2 how to do this.

The arguments are:
- $field: The field where you want to set a value in

- $value: The value you want to set

- $overwriteCurrentValue: Should a posted value or a value from the database also been overwritten?



Example:

```php
<?php

// include the class
include("FH3/class.FormHandler.php");

// make a new formhandler object
$form = new FormHandler();

// set a default value
$form -> setValue("name", "Enter your name...");

// textfield
$form -> textField("Name", "name", FH_STRING);

// button
$form -> submitButton();

// set the 'commit after form' function
$form -> onCorrect("doRun");

// flush the form
$form -> flush();

// the 'commit after form' function
function doRun( $data )
{
   return "Hello ". $data["name"];
}

?>
```

The result is:

Example 2: Setting more then 1 value

```php
<?php

// include the class
include("FH3/class.FormHandler.php");

// options to use in the listfield
$options = array(
  1 => "option 1",
  2 => "option 2",
  3 => "option 3"
);

// make a new formhandler object
$form = new FormHandler();

// listfield
$form -> listField("Options", "options", $options, FH_NOT_EMPTY, true);

// set a default value
// (We have to set the index of the array,
// because we have set the option useArrayKeyAsValue to true.
// If we had not done that, we had to set
// the real "value" of the options array)
$form -> setValue( "options", "1, 3" );

// This was also correct!!
// $form -> setValue( "options", array(1, 3) );

// button
$form -> submitButton();

// set the 'commit after form' function
$form -> onCorrect("doRun");

// flush the form
$form -> flush();

// the 'commit after form' function
function doRun( $data )
{
   global $options;

   $msg = "Selected: \n";

   // show the selected options
   foreach($data['options'] as $id)
   {
      $msg .= " - ".$options[$id]." \n";
   }

   return $msg;
}

?>
```

3.4.5 AddValue

With this funciton you can add a value to the form results (like a hidden field, only without one).

This is very handy is you want to add some data into the database with not have to be shown in the form (like the date of insert).

You can also overwrite a value of a existing field!

The arguments are:
- $field: The name of the field in the table where you want to add some value to

- $value: The value you would like to add

- $sqlFunction: Should the value being interpretaded as a SQL function (Like for example NOW() in MySQL)?

Example:

```php
<?php

// include the class
include("FH3/class.FormHandler.php");

// create new formhandler object
$form = new FormHandler();

// passfield
$form -> passField("Password", "password", FH_STRING);

// get the value of the field
$value = $form->value("password");

// save the password MD5 encrypted
// so OVERWRITE the current value!!
$form->addValue(
  "password",
  "MD5('". mysql_escape_string( $value ) ."')",
  true  # the value should be interpretated as a SQL statement
);

// submitbutton
$form -> submitButton("Save");

// set the onCorrect function
$form -> onCorrect("doRun");

// flush
$form -> flush();

// commit after form function
function doRun($data)
{
    echo "MD5 encrypted password: ".$data['password'];
}

?>
```

3.4.6 OnCorrect

Set the function which has to be called when the form is correct. Your function will get 2 arguments from FormHandler:

1. An array of the data which is saved. The keys of this array represents the field names of the form.
2. A reverence to the FormHandler object. This can be used to make some changes, like check a field and set it to invalid

Note: The second argument which is passed to your oncorrect function is added on 01-04-2006. Before then, only argument 1 was passed trough to your function.

You can also use a method which has to be called when the form is correct. You can do this like shown in example 2.

It is also possible to validate some fields in your oncorrect function. This is shown in example 3.

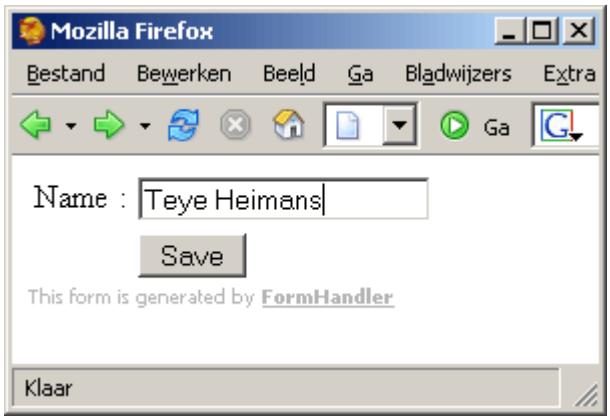
When you return *false* in the onCorrect function, the form is shown again. When returning *true* or *null*, the form will not be displayed again. When returning a *string*, the string will be displayed.

The arguments are:
- $validator: The name of the function which should be called when the form is correct. It is also possible to use a method. In this case you should pass an array where the first item is the object and the second item the name of the method.

Example:

```php
<?php

// include the class
include("FH3/class.FormHandler.php");

// create new formhandler object
$form = new FormHandler("myForm");

// simple textfield + submitbutton
$form -> textField    ("Name", "name", FH_STRING);
$form -> submitbutton ("Save");

// set the commit after correct function
$form -> onCorrect    ("doRun");

// flush the form
$form -> flush();

// the oncorrect function
// make sure you add a & before the second argument!!!
function doRun( $data, &$form )
{
   echo "Hello! Your name is " . $data["name"];
}

?>
```

The result:



Example 2: Calling a method

```php
<?php

// include the class
include("FH3/class.FormHandler.php");

/**** Simple example class!! *******/
class Example
{
   // this is the method we are going to call
   // when the form is correct!
   function doRun( $data, &$form )
   {
      echo "Hello! Your name is " . $data["name"];
   }
}


// create a new example object
// for usage in the onCorrect call!
$example = new Example();

// create new formhandler object
$form = new FormHandler("myForm");

// simple textfield + submitbutton
$form -> textField    ( "Name", "name", FH_STRING );
$form -> submitbutton ( "Save" );

// set the method which we should call when the form is saved
$form->onCorrect(
  // note: the & caracter is needed!
  array(&$example, "doRun")
);

// flush the form
$form -> flush();

?>
```

The result is the same as the example above.

Example 3: Validating

Possible since FH3 v1.2.1 (01-04-2006)

```php
<?php

// include the class
include("FH3/class.FormHandler.php");

// create new formhandler object
$form = new FormHandler("myForm");

// simple textfield + submitbutton
$form -> textField    ("Name", "name", FH_STRING);
$form -> submitbutton ("Save");

// set the commit after correct function
$form -> onCorrect    ("doRun");

// flush the form
$form -> flush();

// the oncorrect function
function doRun( $data, &$form )
{
   // is the name shorter then 3 characters ?
   if( strlen( $data['name'] ) < 3 )
   {
      // set an error message for the name field
      $form -> setError(
        'name',
        'Your name has to be at least 3 characters!'
      );

      // display the form again
      return false;
   }
   else
   {
      echo "Hello! Your name is " . $data["name"];
   }
}

?>
```

The result:


Example - Mozilla Firefox
Bestand Bewerken Beeld Ga Bladwijzers Extra Help
Ga
Name :
Save
This form is generated by FormHandler
Klaar

3.4.8 SetError

(Since version FH3-v1.2)

With this function it is possible to set an error message for a field (the field's value will be interpretated as invalid!)

Note: If you just want to set a custom error message, please use SetErrorMessage!

The arguments are:
- $field: The field where you want to set the error for

- $error: The error message you want to set for the field

Example:

```php
<?php

// include the class
include("FH3/class.FormHandler.php");

// create new formhandler object
$form = new FormHandler("myForm");

// simple textfield + submitbutton
$form -> textField    ("Name", "name", FH_STRING);
$form -> submitbutton ("Save");

// set the commit after correct function
$form -> onCorrect    ("doRun");

// flush the form
$form -> flush();

// the oncorrect function
function doRun( $data, &$form )
{
   // is the name shorter then 3 characters ?
   if( strlen( $data['name'] ) < 3 )
   {
      // set an error message for the name field
      $form -> setError(
        'name',
        'Your name has to be at least 3 characters!'
      );

      // display the form again
      return false;
   }
   else
   {
      echo "Hello! Your name is " . $data["name"];
   }
}

?>
```

The result:


Example - Mozilla Firefox
Bestand Bewerken Beeld Ga Bladwijzers Extra Help
Ga
Name :
Save
This form is generated by FormHandler
Klaar

3.5 Database Functions

3.5.1 dbFormHandler

(Since version FH3-v1.2)

With this function you can create a new FormHandler object with database supporting enabled. For performance reasons it is recommended to use FormHandler() if you dont use one of the database functions!!

The arguments are:
- $name: The name of the form. When none is given, a unique name will be generated.

- $action: The action of the form. By default this is the same URL as the one which was called to display the form. When you set this to another URL formhandler cannot check the form! Make sure it's pointing to itsself!

- $extra: This can be extra information which will be included in the form's html tag. You can add for example some CSS or javascript.

Example:

```php
<?php
/**
* Your Database Connection (In This Example MySQL)
*/
$connection = mysql_connect( "localhost", "username", "password" );

// connection made ?
if( $connection )
{
   // select the database
   if( !mysql_select_db( "database", $connection ) )
   {
      // could not select database!
      die('Could not select the database! Error: ' . mysql_error() );
   }
}
// could not connect to the database
else
{
   die('Could not make a connection to the database! Error: ' . mysql_error() );
}

// include the formhandler
include('FH3/class.dbFormHandler.php');

// create a new form
$form = new dbFormHandler();

// set the database info
$form -> setConnectionResource( $connection, "myTable", "mysql" );

// fields..
$form -> textField('Test', 'myField', FH_STRING );

// here comes the rest of the form
// .....

// set the data handler
// (NOTE the onSaved, this is different then onCorrect!)
$form -> onSaved("doSomething");

// display the form
$form -> flush();

// the data handler...
// NOTE the two arguments!!!!
function doSomething( $id, $data )
{
   // do something here...
}

?>
```

3.5.2 dbSelectField

With this function you can populate a selectfield with values which are loaded from a table.

The arguments are:
- $title: Title of the field

- $name: Name of the field

- $table: The table where the values of this field should be fetched from

- $fields: String or array with the field(s) which are retrieved from the table and put into the select field.

- $extraSQL: Some extra SQL which will be included right achter the "FROM <table>" statement. With this you can order the items, join tables, etc.

- $validator: This can either be the name of your own validation function or the constant name of a predefined validator function. For more information about validators see Validators.
It is also possible to use a method as a validation function. In this case you should pass an array where the first item is the object and the second item the name of the method.

- $multiple: Should it be possible to select more then one option? Default false.

- $size: The number of items which should be shown. Default 1, but when $multiple is set to true the size will be 4.

- $extra: This can be extra information which will be included in the fields html tag. You can add for example some CSS or javascript.

- $mergeArray: The options off the database are merged with the options in this array. The optinos in $mergeArray will come first.

Example:

```php
<?php

// include the formhandler
include( 'FH3/class.dbFormHandler.php' );

// create new db formhandler object
$form = new dbFormHandler();

// set the database info (create a new connection)
$form -> dbInfo( 'database', 'table', 'mysql' );
$form -> dbConnect( 'localhost', 'username', 'password');

// new selectfield
$form -> dbSelectField(
  'Options from a table',
  'saveInField',
  'loadFromTable',
  array( 'keyField', 'valueField' ),
  ' ORDER BY `valueField`',
  FH_NOT_EMPTY
);

// button to submit
$form -> submitButton();

// what to do when the form is saved
$form -> onSaved( 'doRun' );

// show the form
$form -> flush();

// the function which is called when the data is saved
function doRun( $id, $data )
{
   echo "Your selected value '". $data['saveInField'] ."' is saved!";
}

?>
```

3.5.3 dbListField

The arguments are:
- $title: Title of the field

- $name: Name of the field

- $table: The name of the table where the values should be loaded from

- $fields: String or array with the field(s) which are retrieved from the table and put into the select field

- $extraSQL: Some extra SQL which will be included right achter the "FROM <table>" statement. With this you can order the items, join tables, etc.

- $validator: This can either be the name of your own validation function or the constant name of a predefined validator function. For more information about validators see Validators.
It is also possible to use a method as a validation function. In this case you should pass an array where the first item is the object and the second item the name of the method.

- $onTitle: The title which will be displayed above the field where the items are in which will be saved. By default this is "Selected" but is language depented.

- $offTitle: The title which will be displayed above the field where the items are displayed where the user can select from. By default this is "Available" but is language depented.

- $size: The number of items which are displayed (the height of the field)

- $extra: This can be extra information which will be included in the fields html tag. You can add for example some CSS or javascript.

- $verticalMode: indicates whether the two selectfields should be stacked horizontally or vertically.

3.5.4 dbCheckBox

(Since version FH3 v1.2.4)

With this function you can populate a checkbox with values which are loaded from a table.

The arguments are:
- $title: Title of the field

- $name: Name of the field

- $table: The table where the values of this field should be fetched from

- $fields: String or array with the field(s) which are retrieved from the table and put into the select field.

- $extraSQL: Some extra SQL which will be included right achter the "FROM <table>" statement. With this you can order the items, join tables, etc.

- $validator: This can either be the name of your own validation function or the constant name of a predefined validator function. For more information about validators see Validators.
It is also possible to use a method as a validation function. In this case you should pass an array where the first item is the object and the second item the name of the method.

- $extra: This can be extra information which will be included in the fields html tag. You can add for example some CSS or javascript.

- $mask: With this argument you can position the field(s). In the given mask a the placeholder %field% will be replaced with the field. When the mask is full and there are still fields, the mask will be repeated. This works the same as SetMask.
When no mask is given the mask will be used which is set in the config file in the var FH_DEFAULT_GLUE_MASK.

Example:

```php
<?php

// include the formhandler
include( 'FH3/class.dbFormHandler.php' );

// create new db formhandler object
$form = new dbFormHandler();

// set the database info (create a new connection)
$form -> dbInfo( 'database', 'table', 'mysql' );
$form -> dbConnect( 'localhost', 'username', 'password');

// new selectfield
$form -> dbCheckBox(
  'Options from a table',
  'saveInField',
  'loadFromTable',
  array( 'keyField', 'valueField' ),
  ' ORDER BY `valueField`',
  FH_NOT_EMPTY
);

// button to submit
$form -> submitButton();

// what to do when the form is saved
$form -> onSaved( 'doRun' );

// show the form
$form -> flush();

// the function which is called when the data is saved
function doRun( $id, $data )
{
   echo "Your selected value '". $data['saveInField'] ."' is saved!";
}

?>
```

3.5.5 dbRadioButton

(Since version FH3 v1.2.4)

With this function you can populate a radiobutton with values which are loaded from a table.

The arguments are:
- $title: Title of the field

- $name: Name of the field

- $table: The table where the values of this field should be fetched from

- $fields: String or array with the field(s) which are retrieved from the table and put into the select field.

- $extraSQL: Some extra SQL which will be included right achter the "FROM <table>" statement. With this you can order the items, join tables, etc.

- $validator: This can either be the name of your own validation function or the constant name of a predefined validator function. For more information about validators see Validators.
It is also possible to use a method as a validation function. In this case you should pass an array where the first item is the object and the second item the name of the method.

- $extra: This can be extra information which will be included in the fields html tag. You can add for example some CSS or javascript.

- $mask: With this argument you can position the field(s). In the given mask a the placeholder %field% will be replaced with the field. When the mask is full and there are still fields, the mask will be repeated. This works the same as SetMask.
When no mask is given the mask will be used which is set in the config file in the var FH_DEFAULT_GLUE_MASK.

Example:

```php
<?php

// include the formhandler
include( 'FH3/class.dbFormHandler.php' );

// create new db formhandler object
$form = new dbFormHandler();

// set the database info (create a new connection)
$form -> dbInfo( 'database', 'table', 'mysql' );
$form -> dbConnect( 'localhost', 'username', 'password');

// new selectfield
$form -> dbRadioButton(
  'Options from a table',
  'saveInField',
  'loadFromTable',
  array( 'keyField', 'valueField' ),
  ' ORDER BY `valueField`',
  FH_NOT_EMPTY
);

// button to submit
$form -> submitButton();

// what to do when the form is saved
$form -> onSaved( 'doRun' );

// show the form
$form -> flush();

// the function which is called when the data is saved
function doRun( $id, $data )
{
   echo "Your selected value '". $data['saveInField'] ."' is saved!";
}

?>
```

3.5.6 dbInfo

This function sets the database info to make a new connection to a database. After setting this data you have to connect to
the database using the method dbConnect();

Note: You can only use these functions if you make use of the dbFormHandler class! So, include *class.dbFormHandler.php* and create a new FormHandler object with the constructor *dbFormHandler()*.

Note: If you want to use an already opened connection, please use setConnectionResource().

The arguments are:
- $db: The name of the database where you want to connect to

- $table: The name of the table where the values should loaded from and stored in.

- $type: The type of database you are connecting to. The default is set in the configuration file with the var FH_DEFAULT_DB_TYPE.

Example:

```php
<?php

// include the class
include('FH3/class.dbFormHandler.php');

// create new formhandler object
$form = new dbFormHandler();

// Set the database info and connect!
$form->dbInfo( "myDB", "myTable" );
$form->dbConnect( "localhost", "username", "password" );

// some fields + button
$form->textField("Name", "name", FH_STRING, 20, 50);
$form->textField("Age", "age", FH_INTEGER, 3, 3);
$form->selectField("Gender", "gender", array('M', 'F'), null, false);

$form->submitButton("Save");

// data handling
// MAKE SURE YOU USE ONSAVED!
$form->onSaved("doRun");


// display the page
$form->flush();

// the data handler
function doRun( $id, $data )
{
   echo
   "Hello ". $data['name'] ."\n".
   "Your data is saved!\n";
}
?>
```

3.5.7 dbConnect

Create a new connection to the database after setting the database data with the function dbInfo().

Note: You can only use these functions if you make use of the dbFormHandler class! So, include *class.dbFormHandler.php* and create a new FormHandler object with the constructor *dbFormHandler()*.

Note: If you want to use an already opened connection, please use setConnectionResource().

The arguments are:
- $host: The host where the database is located. This could also be the server name for some types of databases. When no host is given, the host saved with the var FH_DEFAULT_DB_HOST from the configuration file is used.

- $username: The username which should be used loggin in into the database.

- $password: The password which should be used loggin in into the database.

Example:

```php
<?php

// include the class
include('FH3/class.dbFormHandler.php');

// create new formhandler object
$form =& new dbFormHandler();

// Set the database info and connect! (Create a new connection)
$form->dbInfo( "myDB", "myTable" );
$form->dbConnect( "localhost", "username", "password" );

// some fields + button
$form->textField("Name", "name", FH_STRING, 20, 50);
$form->textField("Age", "age", FH_INTEGER, 3, 3);
$form->selectField("Gender", "gender", array('M', 'F'), null, false);

$form->submitButton("Save");

// data handling
// MAKE SURE YOU USE ONSAVED!
$form->onSaved("doRun");


// display the page
$form->flush();

// the data handler
function doRun( $id, $data )
{
   echo
   "Hello ". $data['name'] ."\n".
   "Your data is saved!\n";
}
?>
```

3.5.8 SetConnectionResource

Set the connection resource which should be used for the database connection.

This function is implemented since v1.0 on 17 oct 2005.
The arguments $conn and $table are switched since 26 oct 2006. However, the old way still works for backwards compatibility.

Note: You can only use these functions if you make use of the dbFormHandler class! So, include *class.dbFormHandler.php* and create a new FormHandler object with the constructor *dbFormHandler()*.

Note: If you want to create a new connection, please use dbInfo() and dbConnect().

The arguments are:
- $conn: The connection resource used for the database.

- $table: The name of the table where the values should loaded from and stored in.

- $type: The type of database you are connecting to. The default is set in the configuration file with the var FH_DEFAULT_DB_TYPE.

Example:

```php
<?php
/**
* Your Database Connection (In This Example MySQL)
*/
$connection = mysql_connect( "localhost", "username", "password" );

// connection made ?
if( $connection )
{
   // select the database
   if( !mysql_select_db( "database", $connection ) )
   {
      // could not select database!
      die('Could not select the database! Error: ' . mysql_error() );
   }
}
// could not connect to the database
else
{
   die('Could not make a connection to the database! Error: ' . mysql_error() );
}

// include the formhandler
include('FH3/class.dbFormHandler.php');

// create a new form
$form = new dbFormHandler();

// set the database info
$form -> setConnectionResource( $connection, "myTable", "mysql" );

// fields..
$form -> textField('Test', 'myField', FH_STRING );

// here comes the rest of the form
// ............

// set the data handler
// (NOTE the onSaved, this is different then onCorrect!)
$form -> onSaved("doSomething");

// display the form
$form -> flush();

// the data handler...
// NOTE the two arguments!!!!
function doSomething( $id, $data )
{
   // do something here...
}

?>
```

3.5.9 OnSaved

Set the function which has to be called when the form is saved. Your function will get 2 arguments from FormHandler:

1. The ID of the saved record.
2. An array of the data which is saved. The keys of this array represents the field names of the form.

You can also use a method which has to be called when the form is saved. You can do this like shown in example 2.

The arguments are:
- $callback: The name of the function which should be called when the form is correct and saved into the database.
It is also possible to use a method. In this case you should pass an array where the first item is the object and the second item the name of the method.

Example:

```
<?php

// .... your form here..

// set the commit after save function
$form->onSaved("doRun");

// flush the form
$form->flush();

// the onsaved function
function doRun( $id, $data )
{
   echo "The record is saved with id: ".$id." <br />\n";
   echo "Data of the form: <pre>\n";
   print_r( $data );
   echo "</pre>\n";
}
?>
```

Example 2: Calling a method

```php
<?php

// ......

// this is just a example object!
$obj = new YourClass();

// set the method which we should call when the form is saved
$form->onSaved(
  // note: the & caracter is needed!
  array( &$obj, "methodName" )
);

// flush the form
$form->flush();

?>
```

3.5.10 dbTextSelectField

(Since version FH3 v1.2.6)

With this field you have the possibility to present a user a selection from a table in your database to put in the textfield or type there own value

The arguments are:
- $title: Title of the field

- $name: Name of the field

- $table: The table where the values of this field should be fetched from

- $field: String the field which isretrieved from the table and put into the select part of this textfield.

- $extraSQL: Some extra SQL which will be included right achter the "FROM <table>" statement. With this you can order the items, join tables, etc.

- $validator: This can either be the name of your own validation function or the constant name of a predefined validator function. For more information about validators see Validators.
It is also possible to use a method as a validation function. In this case you should pass an array where the first item is the object and the second item the name of the method.

- $extra: This can be extra information which will be included in the fields html tag. You can add for example some CSS or javascript.

Example:

```php
<?php

// include the formhandler
include( 'FH3/class.dbFormHandler.php' );

// create new db formhandler object
$oForm = new dbFormHandler();

// set the database info (create a new connection)
$oForm -> dbInfo( 'database', 'table', 'mysql' );
$oForm -> dbConnect( 'localhost', 'username', 'password');

// new TextSelectfield
$oForm -> dbTextSelectField(
  'Options from a table',
  'saveInField',
  'loadFromTable',
  array( 'keyField', 'valueField' ),
  ' ORDER BY `valueField`',
  FH_NOT_EMPTY
);

// button to submit
$oForm -> submitButton();

// what to do when the form is saved
$oForm -> onSaved( 'FH_RUN' );

// show the form
$oForm -> flush();

// the function which is called when the data is saved
function FH_RUN( $iID, $aData )
{
    echo "Your type or selected value '". $aData['saveInField'] ."' is saved!";
}

?>
```

3.5.11 getDBValue

returns the value of the field in the database

The arguments are:
- sField: Name of the field to get the value from

3.5.12 setEditName

(Since version FH3 v1.2.11)

Set the name of the variable which we are watching in the URL whether the form is an edit form.

This function should be used before dbInfo and dbConnect functions.

If not used the setting of the config.inc.php will be used.

The arguments are:
- $sName: The name of the $_GET variable we should watch to see it's an edit form or not

3.5.13 addJunctionTable

(Since version FH3 v1.2.11)

Add a table and the fields for a junction table
Makes it possible to save data in a junction table instead of comma-separated values.

The arguments are:
- $sTable: The name of the junction table

- $sDestinationField: Foreign key to the table

- $sSourceField: Foreign key to the id of this form

Example:

```
<?
$oForm = new dbFormHandler();
$oForm->dbInfo( DB_NAME, 'table', DB_TYPE );
$oForm->dbConnect( DB_HOST, DB_USER, DB_PASS );

$oForm->textField( 'Name', 'name' );

// get items from an other table as checkbox
$oForm->dbCheckBox( 'Select items', 'item_id', 'tbl_items', array( 'id', 'item' ) );

// add the junction table
$oForm->addJunctionTable( 'tbl_table_has_item', 'item_id', 'table_id' );

$oForm->onSaved( 'FH_SAVE' );
$oForm->submitButton(  );
$oForm->flush( );
?>
```

3.6 Other functions

3.6.1 GetFileInfo

(Since version FH3-v1.2)

With this function you can retrieve the data of an uploaded file retrieved from the $_FILES array.

Please note that this function will not work after calling flush(). It will work in the onCorrect and onSaved functions.

The arguments are:
- $uploadfield: The name of the uploadfield where the file is uploaded.

Example:

```
<?php

// include the formhandler
include("FH3/class.FormHandler.php");

// create a new instance of FormHandler
$form = new FormHandler();

// the uploadfield
$form -> uploadField("file", "file");

// is there a file uploaded?
if( $form -> IsUploaded( "file" ) )
{
   // get the data as in the $_FILES array
   $data  = $form -> GetFileInfo("file");

   // show the data
   echo '<pre>';
   print_r( $data );
   echo '</pre>';
}

// button to submit the form
$form -> submitbutton();

// show the form
$form -> flush();


?>
```

The result will be:

```
Array
(
[name] => newlogo.gif
[type] => image/gif
[tmp_name] => C:\WINDOWS\TEMP\php6.tmp
[error] => 0
[size] => 5902
)
```

3.6.2 IsUploaded

The arguments are:
- $uploadfield: The name of the uploadfield where you want to check if a file has been uploaded in

3.6.3 GetPage *

Returns the page number of the last submitted page the form (when getPage is called)

Better us getLastSubmittedPage()

NOTE: This method is still buggy!

3.6.4 getLastSubmittedPage *

(Since version FH3 v1.2.9)

Returns the page number of the last submitted page the form (when getPage is called)

3.6.5 getCurrentPage *

Returns the page number of the current page of the form (used when newPage is used!)

3.6.6 LinkSelectFields

With this function you can link certan selectfields. So you can load specific values in selectField B which are linked to the selected value of field A.

NOTE: To avoid a known bug, make sure you call this function just before flush() (after all fields!)

NOTE: Don't use this in a multipaged form, it wil give javascript errors

With this function you can make 2 ore more(!) linked selectFields. After an item is selected in the first selectfield, your script is requested with the selected item. Now you have to generate a list of values for the child selectfield. FormHandler put's this values in the selectfield and let the user select an item.

In your script, you will get 3 arguments trough the **$_POST** array:

- **linkselect:** Always true
- **filter:** The selected item in the parent field. You should use this value as a filter for the new values for the child field
- **field:** The field which values you should now load. (So not the fieldname of the parent field!!!)

In your script you can set the new options for the field with the static function setDynamicOptions:
FormHandler::setDynamicOptions($options, $useArrayKeyAsValue);
The first argument ($options) has to be an array of items which should be set. The second argument represents if we have to use the array key's as the values for the field (default). If not, set this argument to *false*.

The arguments are:
- $filename: The name of the file where the field's values are loaded in

- $fields: The name of the first selectfield

- ...: More names of the selectfields...

Example:

click here and click here

```
<?

//Todo.. database option example

?>
```

3.6.7 SetDynamicOptions *

With this function you can set the options which should be loaded into a dynamic selectfield used with LinkSelectFields().

See LinkSelectFields for more information.

The arguments are:
- $options: The options which you want to set

- $useArrayKeyAsValue: Should the key of the options array be used as the value for the field? If not (false) the array value is used.

3.6.8 GetTitle

This function returns the title of the given field name. This function can be usefull when you use the function catchErrors().

The arguments are:
- $field: The fieldname where you want to have the title from

3.6.9 GetLanguage

Get's the language which is used.

3.6.10 FieldExists

Check if the given field exist in the form or not. This could be useful when you create forms dynamicly.

NOTE: This function can only check in the fields which are allready set!

The arguments are:
- $field: The field which you want to check if it exists

Example:

This is the WRONG way!

```
<?php
echo $form->fieldExists( "test" ); // false!!!
$form->textField("Test", "test");
?>
```

This is the CORRECT way!

```
<?php
$form->textField("Test", "test");
echo $form->fieldExists( "test" ); // true!!!
?>
```

3.6.11 GetFormName

Returns the name of the form (used in the <form> tag)

3.6.12 GetJavascriptCode

The arguments are:
- $header: Should the javascript be returned which is determined for the header? Otherwise thejavascript code will be returned which has to be beneath the form.

3.6.13 ResizeImage

NOTE: If only the MaxWidth is set, the MaxHeight will be set to the same value as MaxWidth!

NOTE: The uploadfield creates the dir if it does not exists. The ResizeImage function does NOT! So make sure that the dir exists where the resized image is put!

The arguments are:
- $field: The name of the uploadfield where the image is uploaded

- $saveAs: How the new image should be saved. When nothing is given, the original will be overwritten.

If you want to save a copy of the original image, use something like this: "dir/to/save/"

IF you want to save it with another filename, use something like this: "dir/to/save/image"
The file wil now be saved as "image.<ext>", where <ext> is the extension of the file.

- $maxWidth: The maximum width of the image. If none is given, the default is loaded which is set in the config file with the var FH_DEFAULT_RESIZE_WIDTH.

- $maxHeight: The maximum height of the image. If none is given, this value will be the same as $maxWidth.

- $quality: The quality used for the resized image. When no quality is given, 80% is used.

- $constrainProportions: Keep the proportions when the image is resized?

Example:

```
<?php

// include the class
include('FH3/class.FormHandler.php');

// get the current dir working in
$curDir = $_SERVER['DOCUMENT_ROOT'] . dirname( $_SERVER['PHP_SELF'] );

// upload config
$config = array(
  'path'   => $curDir .'/uploads/',
  'type'   => 'jpg jpeg png gif',
  'exists' => 'rename'
);

// create new formhandler object
$form = new FormHandler('myForm');

// uploadfield
$form->uploadField('Image', 'image', $config);

// save the resized image as...
// MAKE SURE THAT THIS DIR EXISTS!
$saveAs = $curDir .'/uploads/thumbs/'.$form->value('image');

// resize the image
$form->resizeImage( 'image', $saveAs, 200 );

// submit button...
$form->submitButton();

// commit after form function
$form->onCorrect('doRun');

// flush the form
$form->flush();

// the commit after form function
function doRun($data)
{
   global $curDir;

   $dir = $curDir .'/uploads/';

   // display the images
   echo
   "<img src='". $dir . "thumbs/" . $data['image'] . "' border='0' />\n".
   "<img src='". $dir . $data['image'] ."' border='0' />\n";
}

?>
```

3.6.14 MergeImage

Please note that merging an image **with transparancy** only works with some png images. See this topic for more information.

The arguments are:
- $field: The name of the uploadfield where the image is uploaded in.

- $merge: The image which should be used for merging.

- $align: The horizontal alignment of the merge image. Possible values are:
"left"
"center" or "middle"
"right"
Or a percentage between 0 and 100, where '0%' is left and '100%' is right.

- $valign: The vertal alignment where the merge image will be located on the original. Possible values are:
"top"
"middle" or "center"
"bottom"
Or you can give a percentage from 0 to 100, where "0%" is top and 100% is "bottom".

- $transparantColor: The red, green and blue values of a color which should be transparant on the merge image. This only works on PNG24 files!

Example:

```php
<?php

// include the class
include("FH3/class.FormHandler.php");

// create a new formHandler object
$form = new FormHandler("myForm");

// the upload config
$uploadCfg = array(
  "type"   => "jpg jpeg png",
  "name"   => "", // <-- keep the original name
  "exists" => "rename",
  "path"   => "images/uploads"
);

// uploadfield
$form->uploadField("Image", "image", $uploadCfg );

// merge the image with the stamp
// and make all black in the stamp transparant
$form->mergeImage("image", "images/stamp.png", "right", "bottom", array(0,0,0) );

// submit button to submit the form
$form->submitButton();

// set the onCorrect handler
$form->oncorrect("doRun");

// flush the form
$form->flush();

// the oncorrect handler
function dorun( $data )
{
    // display the uploaded image
    echo "<img src='images/uploads/". $data['image']."' alt='' />";
}

?>
```

3.6.15 CheckPassword

With this function you can compare the values of two password fields. This function is build because its a common check made with to password fields (check if they are the same).

When the form is a insert form, the password fields are both required and need to be the same. When the form is an edit form, the password fields can be left empty by the visitor and the original passfield will be kept.

The arguments are:
- $field1: The name of the first passfield.

- $field2: The name of the second passfield

- $setEditMsg: Should a message beeing displayed in an edit form that when leaving the fields blank the current passwords will be kept?

Example:

```php
<?php

// include the class
include("FH3/class.FormHandler.php");

// create a new formhandler object
$form = new FormHandler("myForm");

// create 2 passfields
$form->PassField("Password", "password", FH_PASSWORD);
$form->PassField("Confirm password", "re_password");

// check if both values are the same
// if not, the form will automatically load the form again
// with a error message
$form->checkPassword("password", "re_password");

// set the commit after form function
$form->onCorrect("doRun");

// flush it...
$form->flush();

// display the password
function doRun($data)
{
   echo "Your chosen password: ". $data['password'];
}

?>
```

3.6.16 IsPosted

This function returns true when the form is posted.

3.6.17 IsCorrect

This function walks al fields **which are set at that point** and checks if they are valid. It returns true if the are all valid and false if one or more fields are invalid.

Example:

```
<?php

// include the class
include("FH3/class.FormHandler.php");

// create new formhandler object
$form = new FormHandler();

/* WRONG ! No fields set! */
if( $form->isCorrect() )
{
  // ...
}

// normal textfields
$form->textField("Name", "name", FH_STRING, 20, 50);
$form->textField("Age", "age", FH_INTEGER, 2, 5);

/* CORRECT way of use! There is a form to check! */

// check of the form is posted and if the field(s) are correct
if( $form->isPosted() && !$form->isCorrect() )
{
   // clear the fields
   $form->setValue("name", "", true);
   $form->setValue("age", "", true);
}

// button to submit the form
$form->submitButton();

// set the handler
$form->onCorrect("doRun");

// display the form
$form->flush();

// the data handler
function doRun( $data )
{
   echo "Hello ". $data["name"];
}

?>
```

3.6.18 Flush

The arguments are:
- $return: Should the HTML of the form beeing returned or should it be printed to the screen. By default it's printed.

3.6.19 FormHandler

The arguments are:
- $name: The name of the form. When none is given, a unique name will be generated.

- $action: The action of the form. By default this is the same URL as the one which was called to display the form. When you set this to another URL formhandler cannot check the form! Make sure it's pointing to itsself!

- $extra: This can be extra information which will be included in the form's html tag. You can add for example some CSS or javascript.

Table of contents