1 - Installation & Configuration

1.1 - 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! :-)

1.2 - 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 ....
?>
```

2 - Usage

2.1 - Introduction (About FH)

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!

NOTE: *If you are reading this in the PDF version of the manual, please note that the square-brackets in the examples are not displayed!!! Be aware of that!*

Why is FormHandler Built?

Let me introduce myself first. I am Teye Heimans, 21 years old and my proffesion is building webapplications. At least 40% of those applications consists of forms. To write all these forms is an irritating and time-taking job. That's why I made the FormHandler.
In the beginning it was a simple class which could generate some fields. After sharing it with some friends I decided to put it on the internet, and here it is. Version 1 is downloaded over 5000 times and version 3 is currently in the make. (Version 2 was never launched, it was to complex).

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
- The size FormHandler is only +/- 256 kb (server side)! (without the FCKeditor files)
- 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!

Available languages:
- Dutch

- English
- French
- Spanish
- Czech
- Polish
- Frisian
- German
- Serbian
- Brazil-portuguese
- Italian
- Chinese

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...

2.2 - Validating fields

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

- **Use your own validator**

- **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

// 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;
}

// Our own validation function!!!!
function myValidator( $value ) {
    // check the value
    if( strLen( $value ) == 0) {
        return "You have to enter your name!";
    } 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 (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

// 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;
}

?>
```

The result after submitting an empty field:

2.3 - How to start

Okay, so you have downloaded FH and uploaded it to your server. What now? First of all you need to realize that FormHandler is a tool that will allow you to build forms on a easy way. So you can't just request one of FH's files in your browser and tadaaa, your form is there! You still need to make your own forms!

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

You have to make a script where you are going to use FH. In this example I use "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 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 or when you are using the database option with onSaved.
Finally you have to flush the form so that it will be send to the visitors browser.

```
<?php

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

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

// some fields.. (see manual for examples e.g.)
$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 ". $data.", you are ".$data ." 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! Youre done!

If you still cant make FH work, use the forum or email me!

2.4 - How to use the database option

Since version RC2 FormHandler supports multiple databases. The databases which are supported at the moment are:

- **mysql** - MySQL

- **postgresql** - PostgreSQL

- **access** - Microsoft Access (Windows only)

- **mssql** - Microsoft SQL Server

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.

How does the database option work?

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 database/table it should use. 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 to find out what arguments are passed trough.

Example:

```
<?php

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

// 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...
function doSomething( $id, $data ) {
    // do something here...
```

```
}

?>
```

After passing all the needed data you have to create the form, only this time 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?! ;-)

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: Field 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`)
)  TYPE=MyISAM;
```

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

```
<?php

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

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

// 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!

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

3 - Functions

3.1 - Fields

3.1.1 - CheckBox

This function generates one ore more CheckBoxes.

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

- **Name:** The name of the field

- **Value:** The value which is used for the field.
If the value is a string it will be used as value (No label is used!).
If the value is an array the array contents is used for both field label and value. If you set the useArrayKeyAsValue option at **true**, the array index will be used as field value!

- **Validator:** The function which have to be used to check the value of the field

- **UseArrayKeyAsValue:** If an array is given as value you can determine if the key has to be the value of the field or not. If not (false) the array-value is used.

- **Extra:** Here you can give some CSS or Javascript for the field.

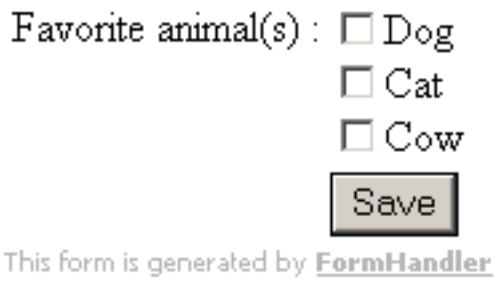
- **Glue:** When using an array as value, you can give here the "glue" which comes between the checkboxes.

Here is an example how to use the field:

```
<?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):<br />\n ";

   foreach($data as $animal) {
       echo " - $animal<br />\n";
   }
}

?>
```

The result:



3.1.2 - DateField

This function generates a DateField.

The arguments are:

- **Title:** The title of the field

- **Name:** The name of the field

- **Validator:** The function witch have to check the value of the date (*NOTE*: the date is automaticly checked for a correct day-month-year combination)

- **Required:** When the field is required, the user doesnt have the option to select nothing.

- **Display:** The format of how the fields are displayed in the form.You can use these caracters:
d - day
m - month
y - year

- **Interval:** The interval between the current year and the years to start/stop.Default the years are beginning at 90 yeas from the current. It is also possible to have years in the future. This is done like this: "90:10" (10 years in the future).

- **Extra:** Posibility to add some CSS or JavaScript to the field tag.

See also the config vars
- FH_DATEFIELD_DEFAULT_DISPLAY,
- FH_DATEFIELD_SET_CUR_DATE,
- FH_DATEFIELD_DEFAULT_DATE_INTERVAL,
- FH_DATEFIELD_DEFAULT_REQUIRED

Example:

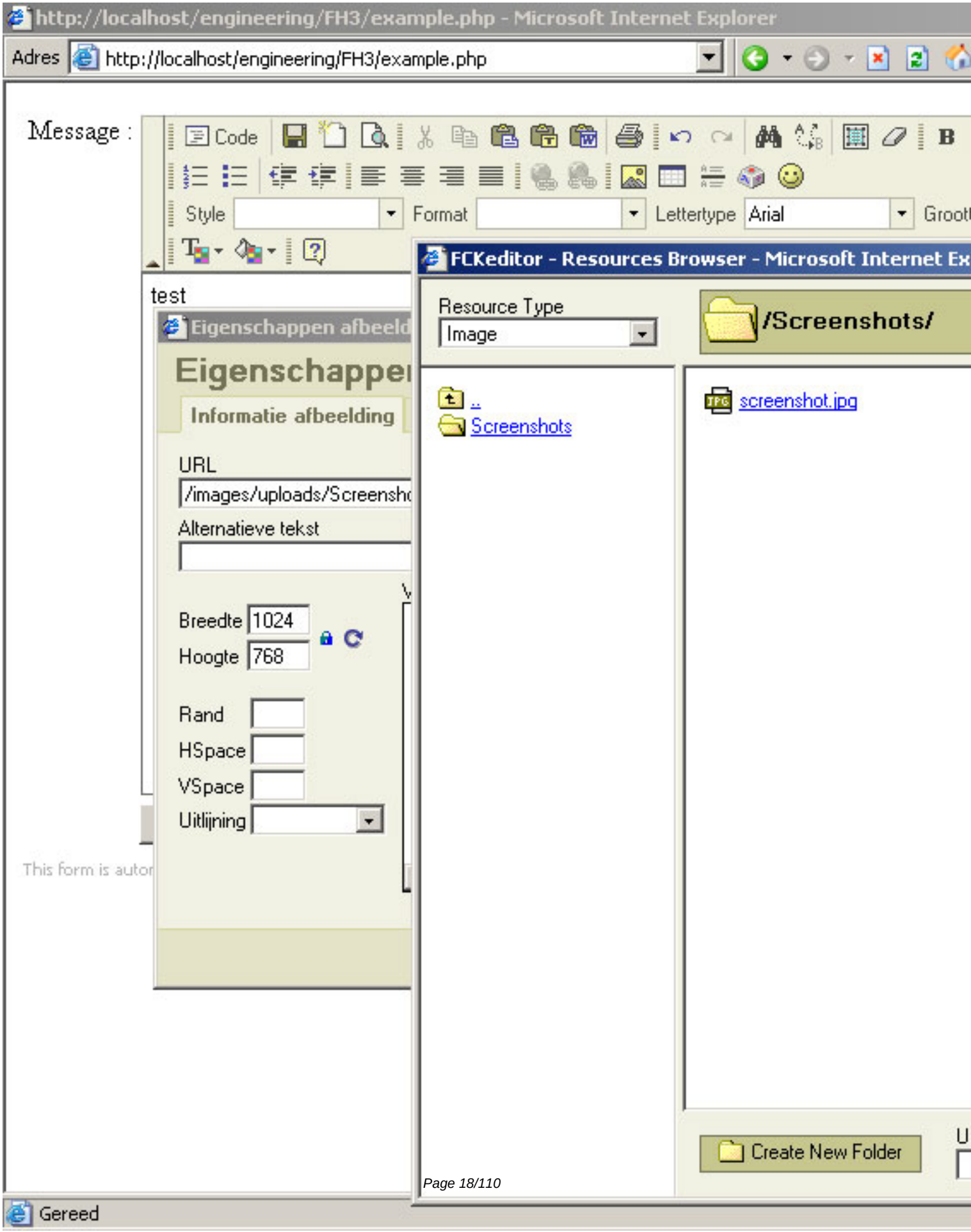
```
<?php

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

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

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

// buttons 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;
}

?>
```

The result is:

3.1.3 - 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.

The arguments are:

- **Title:** The title of the field

- **Name:** The name of the field

- **Validator:** The name of the function to check the editor

- **Path:** When uploading is enabled, you have to set this path where the images are uploaded

- **Toolbar:** The name of the toolbar used for the Editor

- **Skin:** The skin to use for the editor. Possible options are: 'default', 'office2003' and 'silver'.

- **Width:** The width of the editor in pixels

- **Height:** The height of the editor in pixels

The path which is used for browsing on the server 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/
```

NOTE 1: 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/');

NOTE 2: 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.

NOTE 3: Since version RC2 the argument skin is available. Also is the argument path moved 1 place forward, before toolbar.

The path has to be relative to the web root.

An example code:

```
<?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;
}

?>
```

3.1.4 - 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!

The arguments are:

- **Name:** The name of the hidden field.

- **Value:** The value of the Hidden Field.
The value will only be set the first time the form is displayed. After the form is submitted, the submitted value will be used. Also, when the form is an edit form, the value loaded from a database will be used (if there is one).

- **Validator:** The name of the function which FormHandler should use to check if the value is valid.

- **Extra:** Extra code (like CSS or JS) to include with the tag of the field.

NOTE: This function is only usefull if you want to change the value of the hidden field with javascript or something like that.

Example:

```
<?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 .",\n".
    " your prefered language is " . $data;
}
?>
```

3.1.5 - ListField

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

The arguments are:

- **Title:** The title of the field

- **Name:** The name of the field

- **Options:** The options used by the field. If the parameter useArrayKeyAsValue is set, the array index will be used as value for the field.

- **Validator:** The name of the function which has to check the value of the field

- **UseArrayKeyAsValue:** If the array index (key) has to be used as value for the field.

- **OnTitle:** The title above the "on" field

- **OffTitle:** The title above the "off" field

- **Size:** The size (height) of the field. Default 4.

- **Extra:** Option to add some CSS / JavaScript to the field's tag

When you double click on the buttons < or > all items are moved to the other field.

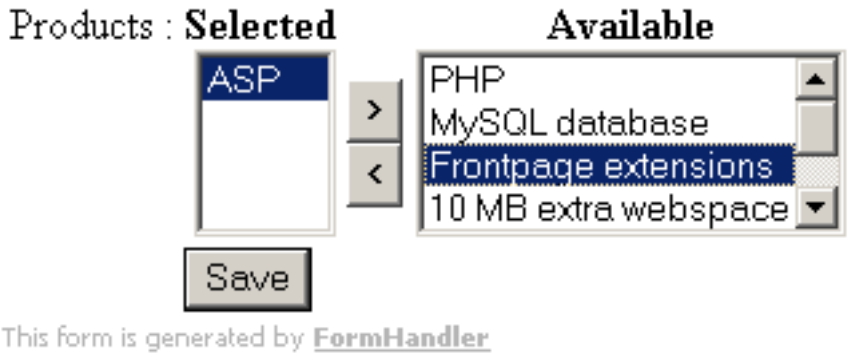
NOTE: This function returnes an array as value. When the value has to be saved into a database the values are comma seperated inserted (like: option1, option2, option3, etc..).

Example:

```
<?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:<br />\n";

    foreach($data as $id) {
        $message .= " - ".$values." <br />\n";
    }

    return $message;
}

?>
```

The result is:



3.1.6 - PassField

Create a PassField on the form.

The arguments are:

- **Title:** The title of the field

- **Name:** The name of the field

- **Validator:** The name of the function which check's the value of the field

- **Size:** The size of the field (default 20)

- **MaxLength:** The maximum input length (default no limit)

- **Extra:** Possibility to add some CSS / JavaScript to the field's tag

Note: You can check 2 passfields with the function **checkPassword**.
Passfields will never be filled with the value, so the password is never vissible in the source code of the page.

Example:

```
<?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;
}
```

```
?>
```

The result is:

3.1.7 - RadioButton

This function generates a group of radiobuttons on the form.

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

- **Name:** The name of the field

- **Options:** The options used by the field. If the parameter *useArrayKeyAsValue* is set, the array index will be used as value for the field.

- **Validator:** The function which have to be used to check the value of the field

- **UseArrayKeyAsValue:** Determine if the keys of the given options-array has to be the value of the field or not. If not (false) the array-value is used.

- **Extra:** Here you can give some CSS or Javascript for the field.

- **Glue:** The "glue" which comes between the radiobuttons.

NOTE: Since 11-07-2005 it is also possible to use php code in the glue. Just start the string with "**php:**". You can use 2 variabele in the php string with are also used by FH, that is **$counter** and **$glue**. You have to save the new glue which should be used in the var **$glue**. **$counter** will be initialized with value 0 so that you can use it as counter in your glue.

Here is a special glue which is correct and can be used:

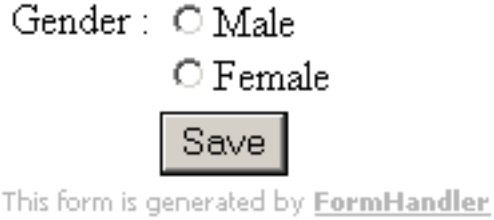
```
<?php
// this string can passed to the glue-argument
$php = 'php:$glue = ++$counter % 2 == 0 ? "</td></tr><tr><td>" : "</td><td>";';

?>
```

Here is an example how to use the field:

```
<?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) {
     case "m":
       echo "boy.";
       break;
     case "f":
       echo "girl.";
       break;
     default:
       echo "shemale!! Whaaaaa...";
       break;
    }
}

?>
```

The result is:



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

In the example above that sould be:

```
<?php

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

?>
```

3.1.8 - SelectField

This function generates a SelectField.

The arguments are:

- **Title:** The title of the field

- **Name:** The name of the field

- **Options:** The options of the field. This has to be an array! 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:** The name of the function which checks the value of the field.

- **UseArrayKeyAsValue:** true, if the indexes (key's) of the options array has to be used as value for the field.

- **Multiple:** Should it be possible to select multiple options? *(This argument is implemented since RC2)*

- **Size:** The size of the field. When multiple is enabled this the default value is 4, otherwise it will be 1.

- **Extra:** Possibility to add some CSS / JavaScript to the field's tag.

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

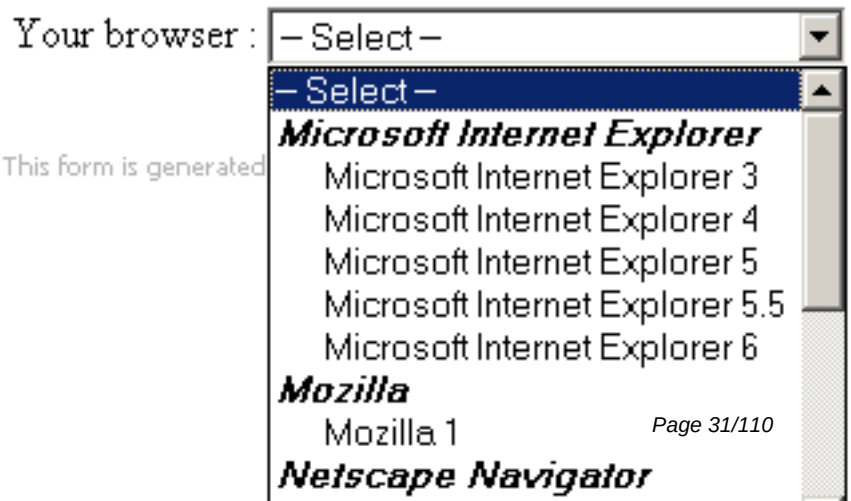
NOTE: when "multiple" is enabled FH will return an array as value (otherwise a string). When the value is saved into a database it will be comma seperated.

Example:

```
<?php

// the options
$browsers = array(
    "0"             => "-- 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;
}

?>
```

The result is:



3.1.9 - TextArea

This function generates a TextArea.

The arguments are:

- **Title:** The title of the field.

- **Name:** The name of the field.

- **Callback:** The name of the function which check's the value of the field.

- **Cols:** The numer of cols of the field.

- **Rows:** The number of rows of the field.

- **Extra:** Possibility to add some CSS / JavaScript to the field's tag.

Example:

```
<?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);
}

?>
```

The result is:

3.1.10 - TextField

Creates a TextField on the form.

The arguments are:

- **Title:** The title of the field

- **Name:** The name of the field

- **Validator:** The name of the function which check's the value of the field

- **Size:** The size of the field (default 20)

- **MaxLength:** The maximum input length (default no limit)

- **Extra:** Possibility to add some CSS / JavaScript to the field's tag

Example:

```
<?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;
}
?>
```

The result is:

3.1.11 - TimeField

This function generates a TimeField.

The arguments are:

- **Title:** The title of the field

- **Name:** The name of the field

- **Validator:** The function witch have to check the value of the time

- **Required:** When the field is required, the user doesnt have the option to select nothing.

- **Display:** The time format, eg. 12 or 24.

- **Extra:** Posibility to add some CSS or JavaScript to the field tag.

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

Example:

```
<?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;
}
```

```
?>
```

The result is:

3.1.12 - UploadField

This function generates a UploadField.

The arguments are:

- **Title:** The title of the field

- **Name:** The name of the field

- **Config:** The configuration options:
path - Directory name where the file must be uploaded (default: "./uploads").
type - The types which are permitted seperated by a comma(like: jpg, gif, etc..)
mime - The allowed mime types.
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:** The name of the function wich checks the upload. This overwrites the default upload check!

- **Extra:** Possibility to add some CSS / JavaScript to the field's tag

- **AlertOverwrite**: If a message have to shown if there's allready a value for this field (in an editform). Default true

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

Mime types

By default, FormHandler checks the mime type of the uploaded file. 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
// 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")
  )
);

?>
```

NOTE: The extension of the file is automaticly checked by javascript (if enabled in the config file), so that the visitor don't have to wait until his file is uploaded!

See also: ResizeImage and MergeImage.

Example: config array

```
<?php
```

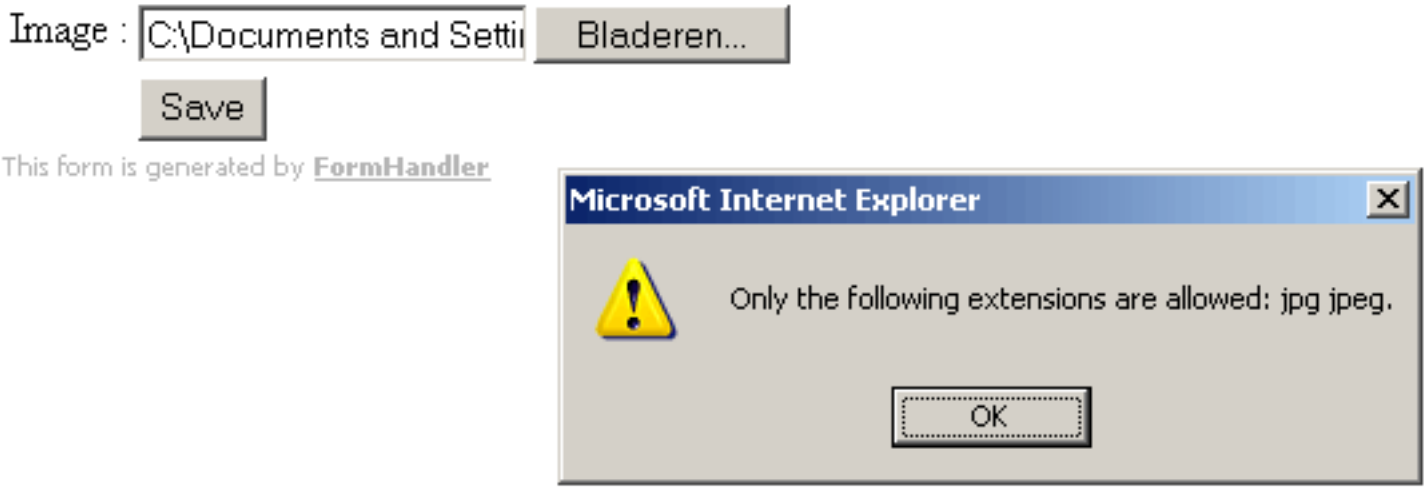
```
$config = array (
  "path"       => "./uploads",
  "type"       => "jpg jpeg png gif",
  "mime"       => array(
    "jpg" => "image/jpeg image/jpg",
    "png" => "image/png",
    "gif" => "image/gif"
  ),
  "size"       => 1000,
  "name"       => "test",
  "width"      => 800,
  "height"     => 600,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

Example:

```
<?php

// 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"       => $_SERVER.dirname($_SERVER).'/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) {

    return
    "The image is saved: <br />\n".
    "<img src='uploads/images/". $data ."'>\n";
}

?>
```

The result:



3.1.13 - dbSelectField

This function generates a SelectField with values retrieved from a table.

This function only works when you make use of the database option!!

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

- **Name:** The name of the field

- **Table:**The table to load the data from

- **Fields:** String or array whith 2 items with the columns which should be used as value for the field. When an array is given, the first column will be used as value for the field. The second column is used as "label".

- **ExtraSQL:** Here you can define extra SQL code what comes behind the " SELECT * FROM table" section. You can use ORDERS, WHERE clauses etc.

- **Validator:** The name of the function which checks the value of the field.

- **Multiple:** Should it be possible to select multiple options? Default false.

- **Size:** The size of the field. Default 1.

- **Extra:** Possibility to add some CSS / JavaScript to the field's tag.

- **MergeArray:** Default value(s) for the field. If you want to ad a value, for example an empty option in front of the list: "array(0 => '--Select--')".

NOTE: FormHandler will add an empty option before the other options. If you want to make the field required you can use the validator FH_NOT_EMPTY.

NOTE: The argument **extraSQL** was added on 22-07-2005! Watch out! NOT BACKWARDS COMPATIBLE!

Example:

```
<?php

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

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

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

```
// the field (select the groups which begin with an A)
$form->dbSelectField(
  "Select your favorie group",
  "favoriteGroup",
  "tblGroup", array("groupId", "groupName"),
  " WHERE groupName LIKE 'A%' ",
  FH_NOT_EMPTY
);

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

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

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

// the function witch handles the code after the form
function doRun( $id, $data) {
    echo "Your data is saved with id ".$id;
}

?>
```

3.1.14 - jsDateField

This function is Beta!

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!

NOTE: This function is implemented in v1.0 since 25-07-2005!

images/manual/jsdatefield.gif

3.2 - Buttons

3.2.1 - Button

Creates a button on the form.

The arguments are:

- **Caption:** The caption of the button

- **Name:** The name of the button. If none is given, a automatic generated name is used.

- **Extra:** Some extra code that should be included in the tag of the button (like CSS or JS)

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...
}
?>
```

The result:

This form is generated by FormHandler

3.2.2 - SubmitButton

Creates a submitbutton on the form.

The arguments are:

- **Caption:** The caption of the button, default submit

- **Name:** The name of the button. If no name is given, a name will be generated and assigned.

- **Extra:** Some extra CSS or Javascript to include with the Buttons HTML tag.

- **DisableBtnOnSubmit:** Disable the button when it is clicked? (Prevents double submitting, default true)

Example:

```
<?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;
}
?>
```

The result is:

3.2.3 - ResetButton

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

The arguments are:

- **Caption:** The caption of the buttonm. Default "Reset"

- **Name:** The name of the button. If none is given, a automatic generated name is used.

- **Extra:** Some extra code that should be included in the tag of the button (like CSS or JS)

Example:

```
<?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."<br />\n";
}
?>
```

The result is:

3.2.4 - 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, default Cancel

- **URL:** The url to go to when the button is pressed

- **Name:** The name of the button. When no name is given, a name will be generated and assigned.

- **Extra:** Extra html tags, CSS or javascript to include with the buttons tag.

Example:

```
<?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;
}

?>
```

The result is:

3.2.5 - ImageButton

Create a image button.

The arguments are:

- **Image:** The path to the image (like "images/button.gif")

- **Name:** The name of the button. If none is given a name wil be generated an assigned.

- **Extra:** Extra CSS, JS or other data to include in the HTML tag of the button

- **DisableButtonOnSubmit:** Disable the button when submitting the form? (Avoiding double submit)

Example:

```
<?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;
}

?>
```

The result:

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.

3.3 - Look & Feel

3.3.1 - AddHTML

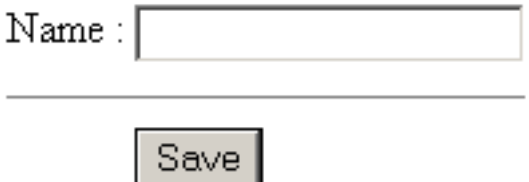
Add some HTML into the form. This HTML will **not** be inserted into a table-row!

Only one argument is given: the HTML to insert into the form.

Example:

```
<?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;
}

?>
```

The result is:



3.3.2 - AddLine

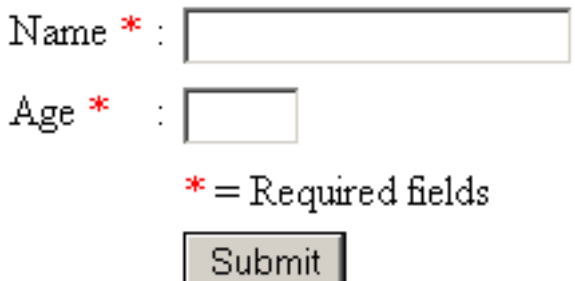
This function adds a line (a new row) into the form table. There is one argument: $text. When some text is given, this is inserted into the line.

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

Example:

```
<?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.",\n".
    "You are ". $data ." years old.\n";
}
?>
```

The result is:



3.3.3 - BorderStop

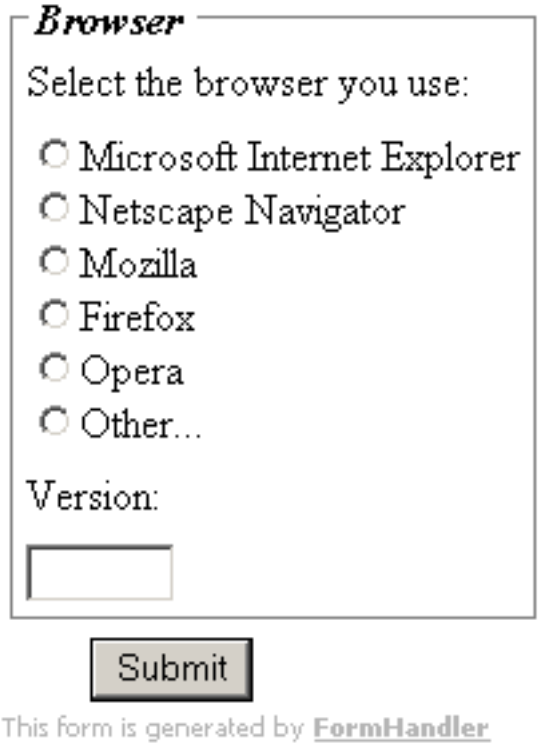
This function ends a border started with the function borderStart(). No arguments are expected.

Example:

```
<?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...
}

?
```

The result:



3.3.4 - BorderStart

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

The arguments are:
- **Caption:** The caption of the fieldset. Leave blank for no caption

- **Name:** The id of the fieldset (can be used to call the fieldset in Javascript)

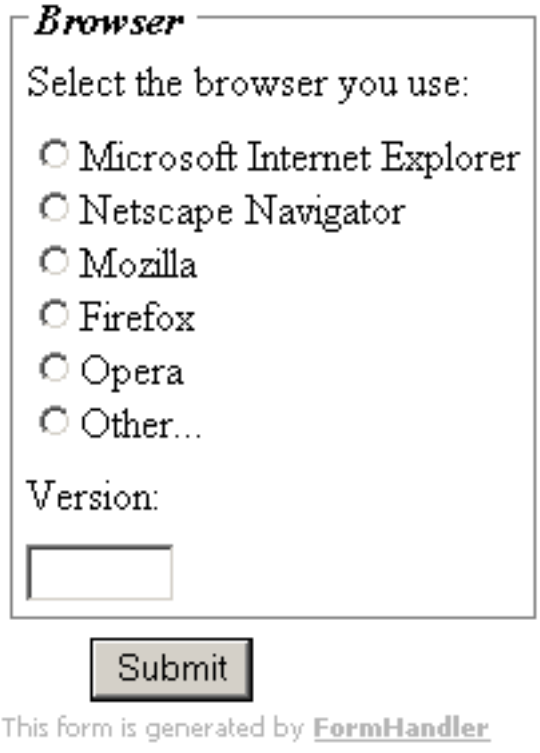
NOTE: The look of the fieldset can be changes in the configuration file (FH_FIELDSET_MASK)!

Example:

```
<?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...
}

?>
```

The result:



3.3.5 - 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%**new - Is replaced with a help icon. See for more information at setHelpText

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.

The arguments of this function:
- **Mask:** The new mask which should be used. *This can also be the path to a file which contains the mask!*

- **Repeat:** When the mask is full and there are still some fields to process, repeat this mask or jump back to the default? This can also be a number! Example: when you set repeat to "2", the mask will be repeated 2 times before the default mask is used again!

- **SetTable:** FormHandler sets automaticly a <table> tag arround the form, because using tables is the most common way to change the layout of a form.
However, if you are using templates or dont want to use a table, set this argument to false and you are free to use whatever you want.

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!

Example 1: Using a different row mask and repeat it for the whole form

```
<?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 saved!";
}
?>
```

The result:

Name :

Age :

Gender :

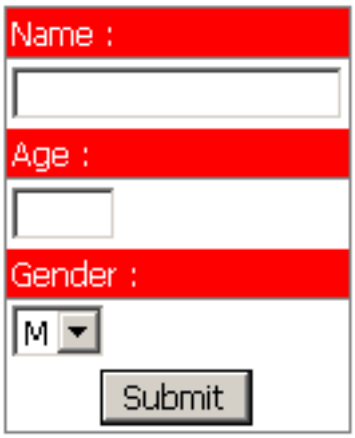
Some HTML code of the result (example of the mask!):

```
<!--
  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="
<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><td>Age :</td></tr>
  <tr><td><input type="text" name="age" id="age" value="" size="3" /> </td></t
  <tr><td>Gender :</td></tr>
  <tr><td><select name="gender" id="gender" size="1">
```

Example 2: Using a template

```
<?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
  false  # FH does not have to set the <table> tag!
);

// 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 saved!";
}
?>
```

The result:



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 />
```

3.3.6 - SetLanguage

Set the language used for displating messages eg. Default the language set by the visitors browser wil be used.

On this page you can find a list of available languages.

Example: setting the language to dutch

```
<?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;
}
?>
```

The result after **submitting an invalid form**

Screenshot of the frysk language:

3.3.7 - 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.

Example:

```
<?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 == 'admin' &&
       $data == 'passw')
    {
         // login here...
    } else {
         return "Error, invalid username or password!";
    }
}

?>
```

The result:

3.3.8 - 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.

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;
}
?>
```

The result is:

3.3.9 - NewPage

NOTE: This function is **beta** and works not with edit forms!!

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

NOTE: this function works in version RC2 made on 4-05-2005 or newer.

Example:

```
<?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";
}

?>
```

The result is:
images/manual/newpage.gif

3.3.10 - 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.

Example:

```
<?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(
  1 => "fld2",
  2 => "fld3",
  3 => "fld1",
  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

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

?>
```

3.3.11 - SetErrorMessage

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

The arguments are: - **Field -** The field to set the message for

- **Message -** The message to display when the value of the field is incorrect.

- **UseDefaultStyle -** Should the default style beeing put arround the error message? (Default true)

Note: This function is added in FH3 v1.0 on 5 juli 05.

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;
}
?>
```

3.3.12 - SetAutoComplete

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

The arguments are: - **Field:** The field which should have the auto complete function

- **Options:** An array with the options for the aut completion.

Example:

```
<?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>";
}
?>
```

The result is:
images/manual/autocomplete.gif

3.3.13 - SetHelpText

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

The arguments are: - **Field:** The name of the field

- **HelpText:** The text for the help message

Example:

```
<?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;
}

?>
```

The result is:
images/manual/sethelptext.gif

3.3.14 - SetHelpIcon

Set an other icon for the help function.

3.3.15 - EnableViewMode

This function enables the view mode. The fields are removed and the values are displayed instead.

3.3.16 - SetMaxLength

With this function you can limit the input of a textarea.

The arguments are:
- **Field** - The name of the textarea field. NOTE! This field must exists!

- **Lenght** - The allowed lenght of the field

- **SetMessage** - Does formhandler has to display a message?

NOTE: This function is added on 25-07-2005 in v1.0.

Example:

```
<?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 );
}
?>
```

The result is:
images/manual/setmaxlength.gif

3.4 - Data handling

3.4.1 - 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" you would like to use to name your data

- **Value:** The value with you want to insert

- **SqlFunction:** If the value of the field is a SQL function (like NOW()). If you set this to true, FormHandler will not set quotes arround it so that it will be interpreted as a SQL function

Example:

```
<?php

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

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

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

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

// 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;
}

?>
```

The result is:

3.4.2 - SetValue

Set a default value for a field.

The arguments are:

- **Field:** The name of the field where you want to set a value of

- **Value:** The value you want to place into the field

- **OverwriteCurrentValue:** If set to true, the new value will overwrite the current value (like loaded from the database or the value which is posted)

This function works on all fields! Also, it does not matter if you call this function before or after the declaration of a field.

Fields which have multiple values, like radiobuttons, listfields and checkboxes (if an array is given), you can set multiple values. You can do this to seperate the values you want to set with a comma (","). See example 2 how to do this.

Example:

```
<?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;
}

?>
```

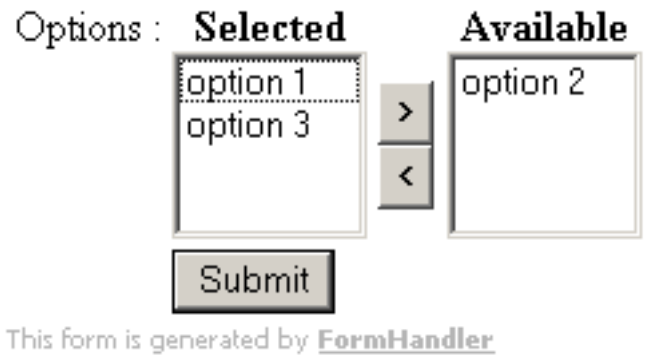
The result is:

Example 2: Setting more then 1 value

```
<?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");

// 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;

    $msg = "Selected: <br />\n";
    foreach($data as $id) {
        $msg .= " - ".$options." <br />\n";
    }

    return $msg;
}

?>
```

The result:



3.4.3 - Value

Return the value of the field which fieldname is given.

This method has one argument: the name of the field which value you want to get.

Example:

```
<?php

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

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

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

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

?>
```

The result is:

3.4.4 - OnCorrect

Set the function which has to be called when the form is correct. Your function will get 1 arguments from the formhandler: 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 correct. You can do this like shown in example 2.

The return value
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.

Example:

```
<?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) {
    echo "Hello! You name is " . $data;
}

?>
```

The result **after** submitting the form:

Hello! You name is Teye Heimans

Example 2: Calling a method

```
<?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) {
        echo "Hello! You name is " . $data;
    }
}


// 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 **after** submitting the form:

Hello! You name is Teye Heimans

3.4.5 - MailForm

NOTE!! THIS FUNCTION WILL NOT BE SUPPORTED ANYMORE IN VERSION 1.2! IT's RECOMMENDED TO NOT USE THIS FUNCTION! USE *THIS METHOD* INSTEAD!

With this function you can send the form data to a email adres. It is possible to send it to more then 1 e-mail adres.

The arguments are: - **To**: The e-mail address of the receiver

- **Subject**: The subject of the message

- **Headers**: Additional headers to send with the mail

Uploaded files will be attached. Resized files or merged files will only be attached if they are replacing the original uploaded file.

Example:

```
<?php

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

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

// here come the fields...
$form->textField('Name', 'name');
$form->uploadField('File', 'file');

// submit button
$form->submitButton('Send');

// send the data to your e-mail address
$form->mailForm( 'your@email.com' );
// you can do this multiple times:
// $form->mailForm( 'someone@else.com' );

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

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

// the 'commit-after-form' function
```

```
function doRun( $data ) {
    echo "Your data has been sent!";
}

?>
```

3.5 - Database

3.5.1 - DBInfo

This function sets the database info to make use of a database. After setting this data you have to connect to the database using the method DBConnect();

The argumens are:

- **DB-Name:** The name of the database which we are using.

- **Table:** The table to write and read the data from. The fields (colums) which are named the same as in the form are updates/inserted automatically.

- **dbType:** The type of the database your are using. Supported are: 'mysql', 'mssql', 'access' and 'postgresql'. By default the type which is set in the config file will be used.

- **PrKey1:** The name of the (first) primary key field (the id-field). When none is given, FormHandler tries to retreive it itself.

- **PrKey2:** The name of the second primary key field (the id-field)
Etc... The rest of the primary keys of the table goes on the same way

Note: This function is changed since version RC2. The first argument *host* is now located in the function dbConnect.

Example:

```
<?php

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

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

// 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 ."\n".
    "Your data is saved!\n";
}

?>
```

3.5.2 - DBConnect

Connect to the database after setting the database data with the function **dbInfo()**.
Currently only MySQL database is supported.

The arguments are:
- **Host:** The host where the database is located. Default is 'localhost' -
- **Username:** The username needed to connect with the database

- **Password:** The password needed to connect with the database

Note: The first argument host is new since version RC2. Before version RC2 the argument host was located in the function dbInfo.

Example:

```
<?php

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

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

// 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 ."\n".
    "Your data is saved!\n";
}
```

```
?>
```

3.5.3 - OnSaved

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

Example:

```
<?php

// ....

// 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

// ......

// 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();


?>
```

3.5.4 - setConnectionResource

Set the connection resource which should be used for the database connection.

The arguments are:
- **Conn:** The connection resource which should be used for the database executions.

- **Table:** The table to write and read the data from. The fields (colums) which are named the same as in the form are updates/inserted automatically.

- **dbType:** The type of the database your are using. Supported are: 'mysql', 'mssql', 'access' and 'postgresql'. By default the type which is set in the config file will be used.

NOTE: This function is implemented since v1.0 on 17 oct 2005.

NOTE: The arguments $conn and $table are switched since 26 oct 2006. However, the old way still works for backwards compatibility.

Example:

```
<?php

// Your connection!!!
$conn = mysql_connect("localhost", "user", "pass") or die(mysql_error());
mysql_select_db( "myDatabase" );


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

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

// Set the connection resource!
$form->setConnectionResource($conn, "myTable", "mysql");

// 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 ."\n".
    "Your data is saved!\n";
}

?>
```

3.6 - Other

3.6.1 - FormHandler

Create a new object of the FormHandler.

The arguments are:

- **Name:** The name of the form. If a name is not given, a default name will be assigned. When the name of the form is allready taken by another formhandler object, it will be added with a number (eg. myForm becomes myForm1)

- **Action:** The action of the form (the URL to go to when the form is submitted). When no action is specified the same URL will be the same like it was when requesting the page.

- **Extra:** Extra tag information you want to include with the <form> tag (like css or javascript).

NOTE: If the action is set to another PHP page FormHandler loses the possibility to validate the form!! So, be carefull with the action!

Example:

```
<?php

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

// create a new Form with the name "MyForm"
$form =& new FormHandler( "MyForm" );

// .... etc ....
?>
```

The result (when looking into the source code of the form):

```
<!--
  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='MyForm' method='post' action='/engineering/test.php'>
<input type="hidden" name="MyForm_submit" id="MyForm_submit" value="1"
<table>
```

3.6.2 - 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).

The arguments are:
- **PassField 1:** The name of the first password field

- **PassField 2:** The name of the second password field

- **SetEditMessage:** On edit, should FormHandler print a message that the currect password will stay if the fields are kept empty.

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.

Example:

```
<?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;
}

?>
```

3.6.3 - MergeImage

Merge an uploaded image with another image.

The arguments are:
- **Field:** the name of the uploadfield where the image is uploaded. (This can also be the path to an existing image! Like: "images/original.jpg", so you dont *have to use* the uploadfield!)

- **Merge:** The merge-image (the image which is pasted on top of the original). Example: "images/stamp.png"

- **Align:** The horizontal position of the merged image on the original. These values can be used: left, center, right.

- **Valign:** The vertical position of the merged image on the original. These values can be used: top, middle, bottom.

- **TransparantColor:** If you want to make a certan color transparant on the merge-image, you can set the RGB value here. If you want to make the black color transparant, use: array(0,0,0). (This works only with PNG stamps!!)

Example:

```
<?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."' alt='' />";
}

?>
```

Example of merged picture:

3.6.4 - ResizeImage

Resize an uploaded image.

The arguments are:

- **Field:** The uploadfield where the image is uploaded. *This can also be the path to an existing image!*

- **SaveAs:** How the new image has to be saved. If none is given, the original will be overwritten. If no extension is given, the extension of the original file will be put behind it.

- **MaxWidth:** The max width of the resized file. (default 80 px)

- **MaxHeight:** The max height of the resized file. (default 80 px)

- **Quality:** The quality of the resized file. (default 80 %)

NOTE: If only the MaxWidth is set, the MaxHeight will be set to the same value as MaxWidth!

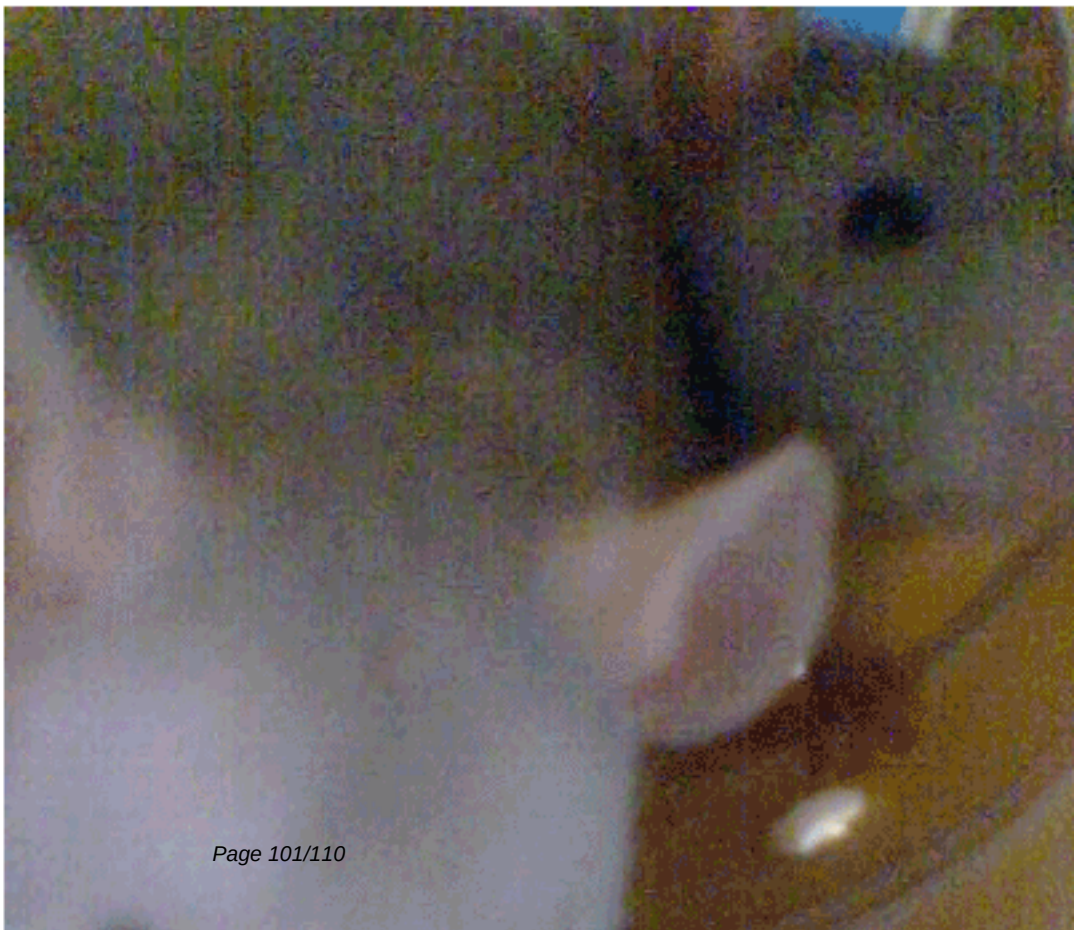
NOTE: The uploadfield creates the dir if it does not exists. The ResizeImage function does ***NOT!*** So make sure that the dir exists where the resized image is put!

Example:

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

// get the current dir working in
$curDir = $_SERVER.dirname($_SERVER);

// 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) {
    $dir = dirname($_SERVER)."/uploads/";

    // display the images
    echo
    "<img src='".$dir."thumbs/".$data."' border='0' /><br />\n".
    "<img src='".$dir.$data."' border='0' /><br />\n";
}

?>
```

The result **after submitting**:



3.6.5 - Flush

Prints or returns the generated form dependent on the argument given.

Example:

```
<?php

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

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

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

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

// FLUSH!
$form->flush(); // print it

// If you want to save the data in an var you can return
// the form like this:
// $html = $form->flush(true);

// commit after form function
function dorun($data) {
    return "Hello ".$data;
}

?>
```

3.6.6 - IsPosted

Check if the form is posted. Returns true if the form is posted, or false if not.

Example:

```
<?php

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

// is the form posted
if( $form->isPosted() ) {
    // do something here...
}

// etc....

?>
```

3.6.7 - 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.

NOTE! This function is added in version 3.0 RC1 on 08-02-2005 / 19:15

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;
}
```

```
?>
```

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().

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();

// 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";
}

?>
```

The result:

3.6.9 - FieldExists

Check if the given field exist in the form or not. This could be usefull when you create forms dynamicly.

NOTE: This function can only check in the fields which are allready set!

```
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!!!

?>
```

3.6.10 - GetJavascriptCode

This function returns the needed javascript code for the form. If this function is not called before the flush(), the javascript code will be echo'd just before the <form> tag.
It is recommended to set the javascript code in the <head> section of the page.

3.6.11 - LinkSelectFields

Note: This function is beta!

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.

The arguments are:
- **Filename:** The file which will return the values for the linked selectfield. How this works is descriped below.

- **SelectField1:** The name of first selectfield (the parent)

- **SelectField2:** The name of the second selectfield (the child) which values are linked to the value selected in the parent field.

- **...:** Possible more names of selectfields.

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*.

Example: click here

Example 2: click here