EXT: Universal Formhandler

Extension Key: formhandler
Language: en

The content of this document is related to TYPO3
- a GNU/GPL CMS/Framework available from www.typo3.org

Table of Contents

Introduction

This extension was initially developed by Reinhard Führicht for Typoheads (www.typoheads.at). After the initial upload to Forge, the development team got enriched by Fabien Udriot and Johannes Feustel. If you want to participate, report bugs or demand additional features please use the possibilities of Forge (forge.typo3.org).

If you like this extension and want to encourage development of new features, feel free to write an email to office(at) typoheads.at to receive details about donation process.

What does it do?

When you reach the limits of the standard mailform functionality of TYPO3 (especially when you want the form to look special) you can use this extension to use your conventional HTML-forms (as HTML-snippets) with full mailform functionality.

This extension is considered as the follow-up to th_mailformplus, including nearly all the features of th_mailformplus and many more new features having a completely new architecture and code. The result is a very flexible approach to form handling.

The new architecture and TypoScript configuration makes it possible to easily control the processing of the form by adding an infinite number of validators and interceptors to filter form input. After successful form submission you can add special interceptors to sanitize input values for final processing in the so called "finishers". Finishers are used to save data into DB, send emails, redirect to another page, etc. Have a look at the available classes described in this manual. If you miss a feature, you can easily write your own class and plug it into MailformPlus MVC processing. After having read the manual and looked into the code documentation, you will see how your own classes can be integrated into MailformPlus MVC.

Please visit the extension's section on forge to post feature requests or report bugs.

http://forge.typo3.org/projects/show/extension-formhandler

Note This extension requires PHP 5.2 or higher and the extension "gimmefive" from Jochen Rau.

Features

- HTML-Template based
- Server side error checks
- Send submitted values as e-mail
- Redirect to a page after successful submission
- Store submitted data in DB
- Captcha support (captcha, sr_freecap, jm_recaptcha, mathguard, wt_calculating_captcha)
- Multipage forms
- HTML mails, plain text mails, multipart mails (HTML+plain)
- File upload handling
- Multipage forms with conditions (branches)
- Frontend listing of submitted data
- Frontend multilanguage support
- Backend module to manage and export submitted data (PDF, CSV)
- ...

Goals

This extension's architecture is designed to be as modular as possible making it easy to perform changes and add new features. As every form needs different configuration than another form, there are many different "blocks" which can be stuck together fulfilling almost all needs.

Comparison to th_mailformplus

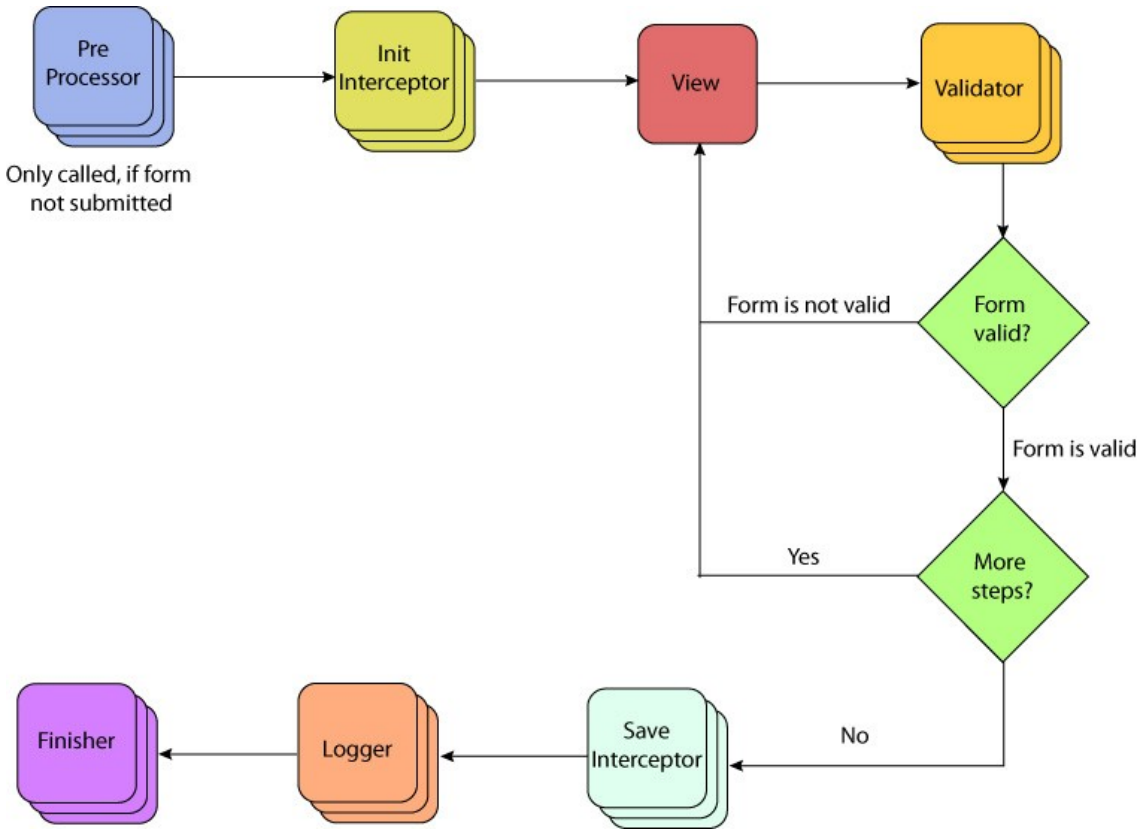
Formhandler is considered a follow-up to th_mailformplus. th_mailformplus has started with a very basic feature set and has grown to a huge extension over the years. The code got kind of unmaintainable and it was time to change the architecture to be quicker spotting and fixing bugs and to offer a whole new experience of flexibility. For a guide on how to upgrade TypoScript configuration from th_mailformplus to Formhandler have a look at the section "How to upgrade from th_mailformplus to Formhandler".

First of all Formhandler has a more up-to-date architecture, which allows a more modular approach having different classes for different tasks. Another advantage of this architecture is a smaller code base making updates and adding of additional features much less time consuming.

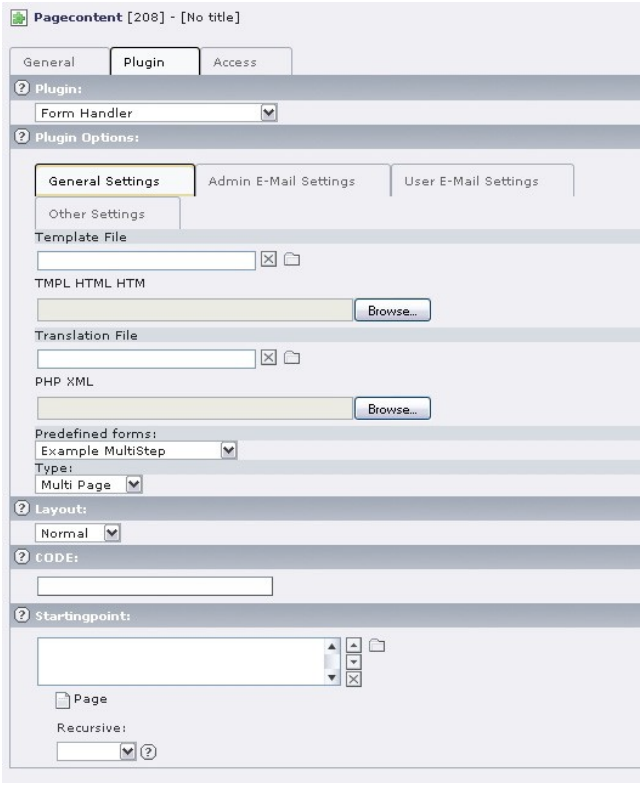
Compared to th_mailformplus Formhandler offers much more flexibility. The modular approach allows the user to add an infinite number of either predefined or self written classes to processing (PreProcessors, Interceptors, Validators, Finishers). When you are not satisfied with the features Formhandler has to offer, you can easily write your own class and add it to the rendering process without changing a single line in the PHP code of Formhandler.

Have a look at the section "How does this extension work?" and the TypoScript configuration options. For a more practical way of explanation, have a look at the "How to ..." sections at the end of the manual or the folder "Examples" in the extension directory.

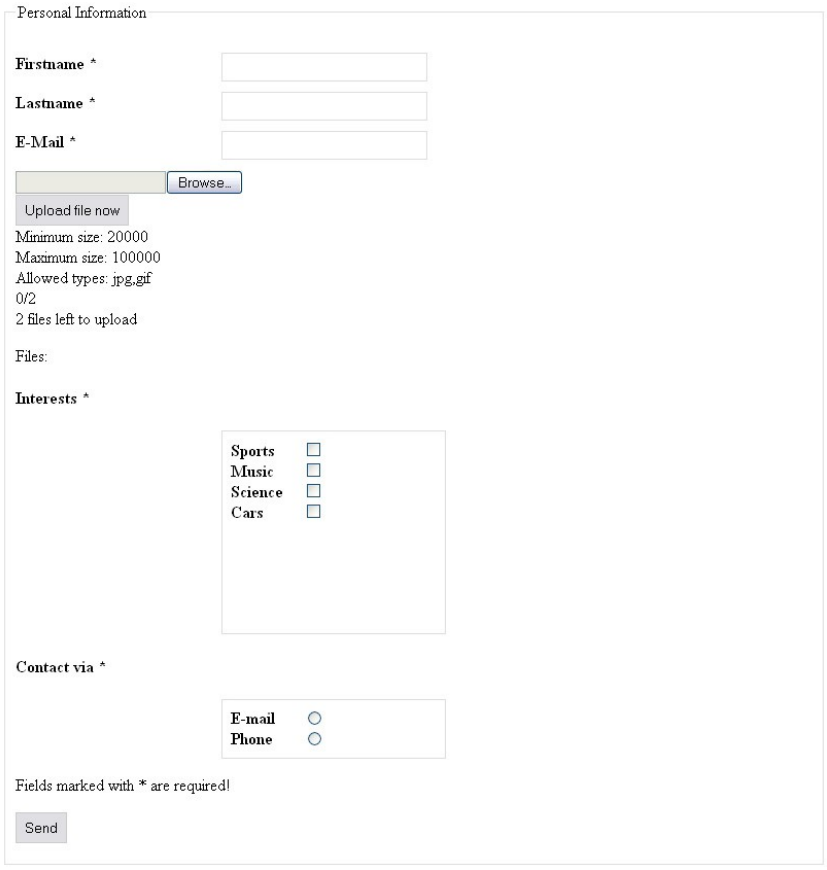
Screenshots



This is how Formhandler works. Quite simple, isn't it? For more information about the rendering process have a look at the section “How does this extension work?”.



The Plugin view of Formhandler. The user can choose a template file and a translation file. Furthermore the user can enter settings for E-mail submission and redirection.



A form generated with Formhandler having file uploads handling and error checks.

Users manual

Create template

To create a form using Formhandler you will first have to write a HTML template file containing the code of your form. You will have to follow certain rules, so that Formhandler will be able to detect the suitable subparts for each situation. For an overview of how to create a form template have a look at the sections "How to set up a simple form" and "Available markers to use in subparts" or check the demo templates located at EXT:formhandler/Examples/

Create translation file

In order to use error messages for your fields and to be able to use multilanguage labels, you have to write a translation file. We decided to make translation files kind of obligatory because it's just nicer. So if you want your form to be fully functional, we recommend to use a translation file. A little tutorial on how to use those translations is in section "How to set up multilanguage forms".

Create a Plugin Record

Create a new page and create a content element. Choose type "General plugin". In the section "Plugin" now choose "Formhandler" as plugin type. Now you are able to select your template file and translation file as well as setting options for e-mail sending, basic "required" error checks and a redirect page after successful submission of the form. When creating a new record, the default predefined form is selected. To add your own predefined form, you should add a new TypoScript template.

Enter your options and save.

Create a TypoScript setup

If you want to use the whole power of Formhandler you should use TypoScript to configure your form. Many different examples are available at EXT:formhandler/Examples/

Tips and tricks: It is sometimes more convenient to edit the TypoScript in an external file. This is particularly true for big typoscript chunk. To do this, insert in a suitable typoscript template (can be the root or a local TS template):

```
<INCLUDE_TYPOSCRIPT: source="FILE:fileadmin/templates/mailform/setup.ts">
```

Then, you could add this line to your user profile / group to avoid clearing the cache all the time.

```
admPanel.override.tsdebug.forceTemplateParsing = 1
```

Include static

Don't forget to include the Formhandler static setup in your root template!

FAQ

– Feel free to contribute on http://forge.typo3.org/projects/extension-formhandler/issues

Administration

Not much to say here. Have a look at the sections "Available subparts to use in the template", "Available markers to use in subparts", the TypoScript reference and the available error checks and you will be fine.

For better understanding the rendering process of this extension will be explained in more detail.

How does this extension work?

The process of rendering a form got simplified to the following simple tasks:

- Preprocessing:

  Do some work before the form is shown, when the page is loaded for the first time (no submission took place). This can be used to load default values for fields or other tasks.

- Intercepting:

  Before the form is shown, there may be a need to intercept and do something with the previously entered data. Formhandler uses this for doing XSS checks. Formhandler knows two types of interceptors, InitInterceptors and SaveInterceptors. InitInterceptors are called everytime before the form is shown, SaveInterceptors are called when the form got submitted successfully right before the Finishers are called.

- Validating:

  After form submission there has to be a possibility to validate the input.

- Finishing

  After successful form submission there may be several tasks to perform, such as saving the input data to database or sending emails.

Any of these mentioned tasks can take more than one class that are called seperately.

A little example:

```
finishers {
  1 {
    class = Finisher_DB
    config {
    ...
    }
  }
  2 {
    class = Finisher_Mail
    config {
    ...
    }
  }
  3 {
    class = Finisher_Redirect
    config {
    ...
    }
  }
}
```

As you see, there are several Finishers defined, which perform different tasks. It is kind of a Finisher chain, so that any number of tasks can be performed. The first Finisher will store the submitted data into a defined database table, the second will send emails to users or administrators, the third will redirect to another page. The idea of creating chains of classes is also possible for PreProcessors, Interceptors and Validators.

Configuration

Reference

Important Note:

Formhandler requires to enter class names of components to perform different tasks.

Example:

```
finishers.1.class = Tx_Formhandler_Finisher_Mail
```

To make the usage easier, it is not needed to prefix the classes with Tx_Formhandler_ in TypoScript:

```
finishers.1.class = Finisher_Mail
```

General Options

Property:	Data type:	Description:	Default:
templateFile	String / TS object	Path to template file or template code	
langFile	String / TS object	Path to language file or language code	
stylesheetFile	String	Path to stylesheet file	
formValuesPrefix	String	Enter a prefix of the form values. Formhandler will then only process fields named like prefix[fieldname] in the HTML template, but lets you enter the fieldnames in config without the prefix. Avoid disturbances with other extensions by prefixing your form fields.	
view	String	Classname of the view to be used	Tx_Formhandler_View_Default
controller	String	Classname of the controller to be used	Tx_Formhandler_Controller_Default
formValuesPrefix	String	Prefix of form fields. Use this if you use a prefix for your forms to avoid conflicts with other plugins. Settings this option you will be able to use only the fieldname in all markers and do not need to add prefix. NOTE: It is highly recommended to use this setting! Example: `<input type="text" name="formhandler[email]" value="###value_email###" />` If you do not set formValuesPrefix, you will not be able to use the marker ###value_email###.	
debug	Boolean (0,1)	Toggle debug mode, which will print debug messages to screen	0
storeGP	Boolean (0,1)	If set, stores the GET / POST variables in the session. May be useful for using those values on an other page. Automatically runs Finisher_StoreGP.	0
requiredSign	String	Enter some text which will be used to substitute markers like ###required_[fieldname]###.	*
singleErrorTemplate.totalWrap	wrap	Enter something to be wrapped around the error messages of a single form field.	
singleErrorTemplate.singleWrap	wrap	Enter something to be wrapped around each error message of a single form field.	
singleErrorTemplate.addDefaultMessage	Boolean	Adds the error message found in 'error_[fieldname]' in the language file at any time.	
errorListTemplate.totalWrap	wrap	Enter something to be wrapped around the error message list of all form fields.	
errorListTemplate.singleWrap	wrap	Enter something to be wrapped around the error messages of a single form field in the list of all form fields.	
singleFileMarkerTemplate.totalWrap	wrap	Enter something to be wrapped around the full list of filenames in a file marker for a single form field.	

Property:	Data type:	Description:	Default:
singleFileMarkerTemplate.singleWrap	wrap	Enter something to be wrapped around each filename in a file marker for a single form field.	
singleFileMarkerTemplate.showThumbnails	Boolean (0/1)	Shows thumbnail of uploaded file instead of filename	0
singleFileMarkerTemplate.image	IMAGE	IMAGE object for thumbnail generation. Image.file gets inserted by Formhandler	
totalFilesMarkerTemplate.totalWrap	wrap	Enter something to be wrapped around the full list of filenames in a file marker for all form fields.	
totalFilesMarkerTemplate.singleWrap	wrap	Enter something to be wrapped around each filename in a file marker for all form fields.	
totalFilesMarkerTemplate.showThumbnails	Boolean (0/1)	Shows thumbnail of uploaded file instead of filename	0
files.uploadFolder	String	Path to a custom upload folder	uploads/formhandler/tmp
files.enableAjaxFileRemoval	Boolean (0/1)	Adds a remove link to every filename in ###[fieldname]_uploadedFiles### and ###total_uploadedFiles###. Unfortunately other markers like ###[fieldname]_fileCount### will not be updated at the moment.	
markers.[markername]	mixed	Define your own custom markers and fill them using TypoScript.	

checkBoxFields	String	Comma seperated list of field names which contain checkbox values. This is needed for multipage forms in order to ensure correct submission of these values from step to step. Please enter all checkbox field names in here. Set this option per step.
		
		Example:
		
		`plugin.Tx_Formhandler.settings {`
		
		`  # Checkbox fields in step 1`
		`  1.checkBoxFields = interests, hobbies`
		
		`  # Checkbox fields in step 2`
		`  2.checkBoxFields = work_experience,accounts`
		`}`	

radioButtonFields	String	Comma seperated list of field names which contain radio button values. This is needed for multipage forms in order to ensure correct submission of these values from step to step. Please enter all radio button field names in here. Set this option per step.
		
		Example:
		
		`plugin.Tx_Formhandler.settings {`
		
		`  # Radiobutton fields in step 1`
		`  1.radioButtonFields = contact_via`
		
		`  # Radiobutton fields in step 2`
		`  2. radioButtonFields = receive_gift`
		`}`	

addErrorAnchors	Boolean (0/1)	If you use ###error### and error markers for each field, you can enable this settings to add anchor links to each message in ###error###. The anchors will point to the messages in ###error_[fieldname]###.	
predef	Array	Predefine form settings and make them selectable in plugin record. Have a look at the section "How to use predefined forms"	

isErrorMarker	array	Configure replacements for ###is_error_[fieldname]### markers.
		
		Example
		
		`plugin.Tx_Formhandler.settings {`
		`  isErrorMarker {`
		`  global = Global message if an error occurred (filled into ###is_error###)`
		`  field1 = TEXT`
		`  field1.value = Some message (filled into ###is_error_field1###)`
		`  }`
		`}`	

[plugin.Tx_Formhandler.settings]

Loggers

Property:	Data type:	Description:	Default:

x.class	String	Enter as many loggers as you like. Each entry requires a class name of the logger. Optional you can enter specific configuration for the logger in the confg section.
		
		Example:
		
		`loggers.1 {`
		`  class = Tx_Formhandler_Logger_Default`
		`}`
		`loggers.2 {`
		`  class = F3_MyPackage_Logger`
		`  config {`
		`  table = tx_mypackage_log`
		`  }`
		`}`	

x.config	Array		

[plugin.Tx_Formhandler.settings.loggers]

PreProcessors

Property:	Data type:	Description:	Default:

x.class	String	Enter as many pre processors as you like. Each entry requires a class name of the pre processor. Optional you can enter specific configuration for the pre processor in the confg section. The pre processors are called only the first time the form is shown.
		
		Example:
		
		`preProcessors.1 {`
		`  class = Tx_Formhandler_PreProcessor_Default`
		`}`
		`preProcessors.2 {`
		`  class = F3_MyPackage_ PreProcessor`
		`  config {`
		`  param = value`
		`  }`
		`}`	

x.config	Array		

[plugin.Tx_Formhandler.settings.preProcessors]

PreProcessor_LoadDefaultValues

Prefills form fields with configured values. Config is done for each step and for each form field. If you aim to prefill dynamically a dropdown menu (e.g. from the database), refer to the section "How to fill your own markers via TypoScript". There is a neat example about how to generate the "option" tags.

Example:

```
preProcessors {
    1.class = Tx_Formhandler_PreProcessor_LoadDefaultValues
       1.config {
         1 {
            tile.defaultValue = {$local.mailform.title}
            lastname.defaultValue = Firstname in here
            firstname.defaultValue = TEXT
            firstname.defaultValue.value = Lastname in here
         }
         2 {
            creditcard.defaultValue = Enter your credit card number here
         }
       }
}
```

[
plugin.Tx_Formhandler.settings.preProcessors.x.class = Tx_Formhandler_PreProcessor_LoadDefaultValues
plugin.Tx_Formhandler.settings.preProcessors.x.config.[x].[fieldname].
]

Property:	Data type:	Description:	Default:

defaultValue	String | TypoScript Object	Field will be filled with this default value	-

PreProcessor_ClearTempFiles

Clears temporary uploaded files older than a specified time.

Example:

```
preProcessors {
    1.class = Tx_Formhandler_PreProcessor_ClearTempFiles
    1.config {
      clearTempFilesOlderThan {
        value = 24
        unit = hours
      }
    }
}
```

[
plugin.Tx_Formhandler.settings.preProcessors.x.class = Tx_Formhandler_PreProcessor_ClearTempFiles
plugin.Tx_Formhandler.settings.preProcessors.x.config.[x].[fieldname].
]

Property:	Data type:	Description:	Default
clearTempFilesOlderThan.value	int	Value of the time	-
clearTempFilesOlderThan.unit	string	Unit of the time. Values may be minutes, hours or days	hours

Interceptors

Property:	Data type:	Description:	Default:

x.class	String	Enter as many interceptors as you like. Each entry requires a class name of the interceptor. Optional you can enter specific configuration for the interceptor in the confg section. The init interceptors are called everytime before the form is shown.
		
		Example:
		
		`initInterceptors.1 { class = Tx_Formhandler_Interceptor_RemoveXSS }`
		`initInterceptors.2 { class = F3_MyPackage_ Interceptor config { param = value } }`	

x.config	Array		

[plugin.Tx_Formhandler.settings.initInterceptors]

Property:	Data type:	Description:	Default:
x.class	String	Enter as many interceptors as you like. Each entry requires a class name of	

Property:	Data type:	Description:	Default:

x.config	Array	the interceptor. Optional you can enter specific configuration for the interceptor in the confg section. The save interceptors are called before the finishers are executed.
		
		Example:
		
		`saveInterceptors.1 {`
		`  class = Tx_Formhandler_Interceptor_Default`
		`}`
		`saveInterceptors.2 {`
		`  class = F3_MyPackage_ Interceptor`
		`  config {`
		`  param = value`
		`  }`
		`}`	

[plugin.Tx_Formhandler.settings.saveInterceptors]

Interceptor_RemoveXSS

Removes malicious code from submitted values to prevent XSS. No further configuration required!

This class experiences some problems with special characters such as German Umlauts.

Example:

```
plugin.Tx_Formhandler.settings.initInterceptors.1.class = Tx_Formhandler_Interceptor_RemoveXSS
```

Interceptor_Filtreatment

Removes malicious code from submitted values to prevent XSS. No further configuration required!

In the opinion of the author, this class does a better job than Interceptor_RemoveXSS.

Example:

```
plugin.Tx_Formhandler.settings.initInterceptors.1.class = Tx_Formhandler_Interceptor_Filtreatment
```

Interceptor_IPBlocking

If you do not want to use Captcha for your form, you can control how often the form is allowed to be submitted by a single IP address or in total with this class.

[
plugin.Tx_Formhandler.settings.initInterceptors.x.class =Tx_Formhandler_Interceptor_IPBlocking
plugin.Tx_Formhandler.settings.initInterceptors.x.config.
]

Property:	Data type:	Description:	Default:
redirectPage	int/string	Page ID or URL to redirect to. If a user is blocked from submiting the form, he gets redirected to this page.	-
report	array	If a user gets blocked from submitting the form, a report email is sent to the configured recipient.	-
report.email	string	Comma seperated list of recipients.	
report.subject	string	Subject of the report mail	
report.sender	string	Email address to be set as sender of the report mail.	
report.interval	array	To prevent the recipient of the report mails to get spammed with report mails, you can configure the interval. A report mail will be sent only once during the interval for a single alert.	
report.interval.value	int	Value	

report.interval.unit	string	And unit for the interval. Unit can be seconds,minutes,hours and days	
ip	array	Settings for limiting submission for an IP address	
ip.threshold	int	The amount of submissions for an IP address in the given period of time	
ip.timebase.value	int	Value	
ip.timebase.unit	string	And unit for the mentioned period of time Unit can be seconds,minutes,hours and days	
global	array	Settings for limiting submissions for a form. SAME SETTINGS AS FOR IP.	

Interceptor_ParseValues

Parse formatted values. Currently only floats such as "1.022.000,76 EUR". See unittest file for detailed examples. Attention: x.xxx.xxx gets parsed to xxxxxxx but xx.xx to xx.xx.

[
plugin.Tx_Formhandler.settings.saveInterceptors.x.class =Tx_Formhandler_Interceptor_ParseValues
plugin.Tx_Formhandler.settings.saveInterceptors.x.config.
]

Property:	Data type:	Description:	Default:
parseFloatFields	string	Comma seperated list of fields to parse.	-

Interceptor_AntiSpamFormTime

Spam protection for the form withouth Captcha. It parses the time the user needs to fill out the form. If the time is below a minimum time or over a maximum time, the submission is treated as Spam.

If Spam is detected you can redirect the user to a custom page or use the Subpart ###TEMPLATE_ANTISPAM### to just display something.

IMPORTANT: You will need a form field named "formtime" containing a timestamp in your form. To do this you can use the marker ###TIMESTAMP###.

Example:

```
<input type="hidden" name="formhandler[formtime]" value="###TIMESTAMP###" />
```

[
plugin.Tx_Formhandler.settings.saveInterceptors.x.class =Tx_Formhandler_Interceptor_AntiSpamFormTime
plugin.Tx_Formhandler.settings.saveInterceptors.x.config.
]

Property:	Data type:	Description:	Default:
minTime.value	int	value	-
minTime.unit	string	unit for the minimum time. Unit can be seconds,minutes,hours and days	-
maxTime.value	int	value	-
maxTime.unit	string	unit for the minimum time. Unit can be seconds,minutes,hours and days	-
redirectPage	mixed	Page ID or URL to redirect in case Spam is detected	

Validators

Property:	Data type:	Description:	Default:

x.class	String	Enter as many validators as you like. Each entry requires a class name of the validator. Optional you can enter specific configuration for the validator in the confg section. For detailed information about the default validator settings have a look at the section "Available error checks".
		
		Example:
		
		`validators.1 {`
		`  class = Tx_Formhandler_Validator_Default`
		
		`  /*`
		`  * Some times you maybe want to disable error checks if the user`
		`  * filled out a specific fields or else.`
		`  * Temporary disable error checking for the entered fields by setting`
		`  * this option.`
		`  */`
		`  disableErrorCheckFields = firstname,lastname`
		`  config {`
		`  fieldConf {`
		`  firstname.errorCheck {`
		`  1 = required`
		`  2 = maxLength`
		`  2.value = 20`
		`  }`
		`  email.errorCheck {`
		`  1 = required`
		`  2 = email`
		`  }`
		`  }`
		`  }`
		`}`
		`validators.2 {`
		`  class = F3_MyPackage_ Validator`
		`  config {`
		`  param = value`
		`  }`
		`}`	

x.config	Array		

[plugin.Tx_Formhandler.settings.validators]

Have a look at the section "Available error checks"

Finishers

Property:	Data type:	Description:	Default:
x.class	String	Enter as many finishers as you like. Each entry requires a class name of	

Property:	Data type:	Description:	Default:

x.config	Array	the finisher. Optional you can enter specific configuration for the finisher in the confg section. For more information about available finishers and their settings have a look at the section "Available finishers".
		
		Example:
		
		`finishers.1 {`
		`  class = Tx_Formhandler_Finisher_Mail`
		`  config {`
		`  admin {`
		`  to_email = rf@typoheads.at`
		`  to_name = Reinhard Führicht`
		`  subject = Request`
		`  sender_email = contactform@mysite.com`
		`  replyto_email = email`
		`  replyto_name = name`
		`  }`
		`  user {`
		`  to_email = email`
		`  to_name = name`
		`  subject = ###LLL:user_subject###`
		`  sender_email = contactform@mysite.com`
		`  sender_name = My Site`
		`  replyto_email = contactform@mysite.com`
		`  }`
		`  }`
		`}`
		`finishers.2 {`
		`  class = Tx_Formhandler_Finisher_Default`
		`  config {`
		
		`  }`
		`}`	

[plugin.Tx_Formhandler.settings.finishers]

Finisher_ClearCache

Clear the cache of the current page. It needs no further configuration.

Finisher_GenerateAuthCode

Generates a unique token for a database entry.

This can be users for FE user registration or newsletter registration.

It needs no further configuration.

IMPORTANT: Use this finisher after Finisher_DB and before Finisher_Mail to be able to send the auth code in an email.

Finisher_Redirect

Redirects to specified page after successful form submission.

[
plugin.Tx_Formhandler.settings.finishers.x.class =Tx_Formhandler_Finisher_Redirect
plugin.Tx_Formhandler.settings.finishers.x.config.
]

Property:	Data type:	Description:	Default:
redirectPage	int/string	Page ID or URL to redirect to	-
correctRedirectUrl	Int (1/0)	Replaces '&' with '&' in URL	0

Finisher_StoreUploadedFiles

Moves uploaded files from temporary folder to a new one and renames them if set.

[
plugin.Tx_Formhandler.settings.finishers.x.class =Tx_Formhandler_Finisher_StoreUploadedFiles
plugin.Tx_Formhandler.settings.finishers.x.config.

]

Property:	Data type:	Description:	Default:
finishedUploadFolder	string	Path where to store the files	-
renameScheme	string	Specifiy how the files should be renamed. Example: renameScheme = [pid]_[filename] Possbile options: [pid] = The current page id [md5] = md5 hash over filename [time] = The submission time stamp [filename] = Original Filename [xxx] = Your own marker defined in "schemeMarkers"	-

schemeMarkers	array	Define your own markers to be used in rename scheme.
		
		Example:
		
		renameScheme = [mymarker]_[marker2]_[filename]
		schemeMarkers {
		mymarker = TEXT
		mymarker.value = asdf
		marker2 = fieldValue
		marker2.field = field1
		}
			-

Finisher_Confirmation

Shows an overview of submitted data and links to save the data as PDF or CSV. Also shows a print link.

To configure how the output looks like you have to add a special subpart to your HTML template.

Example:

```
<!-- ###TEMPLATE_CONFIRMATION### begin -->
<table>
  <tr>
    <td>###LLL:firstname###</td>
    <td>###value_firstname###</td>
  </tr>
</table>
<div>
###PRINT_LINK### / ###PDF_LINK### / ###CSV_LINK###
</div>
<!-- ###TEMPLATE_CONFIRMATION### end -->
```

[
plugin.Tx_Formhandler.settings.finishers.x.class = Tx_Formhandler_Finisher_Confirmation
plugin.Tx_Formhandler.settings.finishers.x.config.
]

Property:	Data type:	Description:	Default:
returns	Int (1/0)	MUST be set. Tells the controller that after this finisher, no other finishers can be called!	-
pdf	array	Settings for the PDF generation	-
pdf.class	string	The class to handle PDF generation. Except you write your own class, use: Tx_Formhandler_Generator_PDF or Tx_Formhandler_Generator_TCPDF	Tx_Formhandler_Generator_PDF

pdf.exportFields	string	Comma seperated list of fields to export to PDF. Example: firstname,lastname,email	All submitted fields
pdf.export2file	Int (1/0)	Saves PDF into a temporary file and redirects to it. If not set, the PDF is kept in memory.	0
csv	array	Settings for the CSV generation	
csv.class	string	The class to handle CSV generation. Except you write your own class, use: Tx_Formhandler_Generator_CSV	Tx_Formhandler_Generator_CSV
csv.exportFields	string	Comma seperated list of fields to export to CSV. Example: firstname,lastname,email	

Finisher_DB

Saves submitted values into specified DB table.

[
plugin.Tx_Formhandler.settings.finishers.x.class = Tx_Formhandler_Finisher_DB
plugin.Tx_Formhandler.settings.finishers.x.config.
]

Property:	Data type:	Description:	Default:
table	string	The table name to store the data into	-
key	string	The field with primary key Example: key = uid	uid
updateInsteadOfInsert	int(1/0)	Does not store the values as a new row in the table, but tries to update an existing record. Needs a uid set in GET/POST parameters. Add a hidden field named like the value entered in "key" to your form. Don't forget to add the formValuePrefix!	0

fields	array	Mapping of table fields to submitted values.
		
		Usage:
		
		fields {
		  [db_field].mapping = [form_field]
		}
		
		Example:
		
		fields {
		  header.mapping = name
		}	

fields.[db_field].ifIsEmpty	string	Define a default value if the user did not enter something in the form field	
fields.[db_field].seperator	string	If the field value is an array 8e.g. Checkboxes) you can enter a seperator to save the value imploded into DB.	,

fields.[db_field].special	string	There are some special values available: sub_datetime sub_tstamp ip inserted_uid (explainedbelow) Should not need further explanation. inserted_uid When using this value you can set the according table using special.table The field value will be set with a uid value of an inserted record by another Finisher_DB before the current one. Use this to store data into more than one table. Example in section How to store data into more than one DB table	
fields.[db_field].preProcessing	TS object	Preprocess the value before saving into database Example: `fields { header { preProcessing = USER preProcessing.userFunc = user_helper->appendSomething mapping = firstname } }`	
fields.[db_field].postProcessing	TS object	Postprocess the value before saving into database. Enables you to override processing of Formhandler, if you are not satisfied with it. Example: `fields { header { postProcessing = USER postProcessing.userFunc = user_helper->appendSomething mapping = firstname } }`	

Finisher_DifferentDB

Saves submitted values into specified DB table using a database other than the TYPO3 database.

[
plugin.Tx_Formhandler.settings.finishers.x.class =Tx_Formhandler_Finisher_DifferentDB
plugin.Tx_Formhandler.settings.finishers.x.config.
]

Property:	Data type:	Description:	Default:
host	string	Host of the databse	
port	string	Port to use	
db	string	Name of the databse	
driver	string	Driver to use Example: driver=oci8	

username	string	Username for DB	
password	string	Password for DB user	
table	string	The table name to store the data into	-
key	string	The field with primary key Example: key = uid	uid
updateInsteadOfInsert	int(1/0)	Does not store the values as a new row in the table, but tries to update an existing record. Needs a uid set in GET/POST parameters. Add a hidden field named like the value entered in "key" to your form. Don't forget to add the formValuePrefix!	0
fields		Have a look at the config section of Finisher_DB	

Finisher_Mail

Sends emails to specified addresses using following subparts in the HTML template:

- TEMPLATE_EMAIL_USER_PLAIN
- TEMPLATE_EMAIL_USER_HTML
- TEMPLATE_EMAIL_ADMIN_PLAIN
- TEMPLATE_EMAIL_ADMIN_HTML

```
[
plugin.Tx_Formhandler.settings.finishers.x.class =Tx_Formhandler_Finisher_Mail
plugin.Tx_Formhandler.settings.finishers.x.config.
]
```

Property:	Data type:	Description:	Default:
limitMailstoUser	int	Limit the amount of mails sent to users. If more addresses are specified than this value, emails are sent only to the first x recipients.	unlimited
checkBinaryCrLf	string	Comma seperated list of fields. Converts chr(13) line breaks to 	uid
admin	array	Settings for the mail sent to site admin	-
admin.header	string	Manually set the email header. All options can be strings, TypoScript objects or names of form fields.	-
admin.to_email	string	Comma seperated list of email addresses. Can be addresses or form fields.	-
admin.to_name	string	Comma seperated list of names to the according addresses in to_email. Can be name or form fields.	-
admin.replyto_email	string	Reply to email address	-
admin.replyto_name	string	Name of the according replyto_email	-
admin.cc_email	string	A CC email will be sent to this address.	-

admin.sender_email	string	Email address to be set as sender of the email. Can be addresses or form fields.	-
admin.sender_name	string	Name of the email sender. Can be name or form fields.	-
admin.subject	string	Subject of the email	
admin.return_path	string	Email address to be set as return path.	-
admin.attachment	string	Comma seperated list of form fields or file names to be attached to the email.	-
admin.attachPDF	array	Attach the submitted values as PDF	-
admin.attachPDF.class	string	Class to handle the PDF generation. Normally this will be Tx_Formhandler_Generator_PDF or Tx_Formhandler_Generator_TCPDF	Tx_Formhandler_Generator_PDF
admin.attachPDF.exportFields	string	Comma seperated list of form fields to export to PDF.	All submitted fields
admin.htmlEmailAsAttachment	int(1/0)	Attach the generated HTML email to the outgoing email	0

admin.filePrefix	string|array	Set the prefix of the generated PDF and HTML files attached to email. Use this setting as an array to set different prefixes for the two files.
		
		Example:
		
		admin.filePrefix = myForm
		
		or for different prefixes:
		
		admin.filePrefix {
		html = myFormHTML
		pdf = myFormPDF
		}
			formhandler_

user	array	Settings for the email sent to the user. SAME SETTINGS AS FOR ADMIN	

Frontend Listing

Property:	Data type:	Description:	Default:
view	String	Class name of the view to be used	Tx_Formhandler_View_Listing
templateFile	String	Template file to be used	
pid	Integer	The page ID to select the logged records from.	
table	String	The table the logged records are stored in	
orderby	String	Enter an ORDERBY statement Example: orderby = subheader DESC	
enableDelete	Boolean (0/1)	Enable the deletion of records in Frontend Listing. If set, the marker ###DELETE### will get replaced with a delete link.	0

Property:	Datatype:	Description:	Default:

mapping	Array	Map db fields to fieldnames to use in markers.
		
		Example:
		
		mapping.header=name
		
		Markers in Template:
		
		###value_name###
		
		mapping {
		header=name
		bodytext=subject
		subheader=sub_datetime
		crdate=sub_tstamp
		tstamp=sub_tstamp
		imagecaption=ip
		}	

[plugin.Tx_Formhandler.settings.fe_listing]

Available error checks

The default validator offers a variety of predefined error checks for the most common tasks. Here is a list of the checks and how to configure them:

Check:	Description:	Parameters:	Datatype:
required	Checks if a field is filled out	none	-
email	Checks if a field contains a valid email	none	-
integer	Checks if a field contains a valid integer value	none	-
float	Checks if a field contains a valid float value	none	-
containsNone	Checks if a field contains none of the configured values	words	comma seperated
containsOne	Checks if a field contains at least one of the configured values	words	comma seperated
containsAll	Checks if a field contains all of the configured values	words	comma seperated
equals	Checks if a field equals the configured value	word	string
equalsField	Checks if a field value equals another field value	field	string
notEqualsField	Checks if a field value does not equal another field value	field	string
notDefaultValue	Checks if a field equals a default value set by a pre processor	defaultValue	string/TS object
minValue	Checks if the value of a field is at least the configured value	value	numeric
maxValue	Checks if the value of a field is less than the configured value	value	numeric
minLength	Checks if the value of a field has at least the configured length	value	integer
maxLength	Checks if the value of a field has less than the configured length	value	integer
betweenValue	Checks if the value of a field is between the configured values	minValue maxValue	numeric numeric
betweenLength	Checks if the length of the value of a field is between the configured values	minValue maxValue	integer integer
minItems	Checks if a field contains at least the configured amount of items (e.g. checkboxes)	value	integer
maxItems	Checks if a field contains less than the configured amount of items (e.g. checkboxes)	value	integer
betweenItems	Checks if a field contains values between the configured amount of items (e.g. checkboxes)	minValue maxValue	integer integer

Check:	Description:	Parameters:	Datatype:
isInDBTable	Checks if the value of a field is in a configured field in a configured table	table field	string string
isNotInDBTable	Checks if the value of a field is not in a configured field in a configured table	table field	string string
ereg	Checks a field value using the configured regular expression	value	regexp
eregi	Checks a field value using the configured regular expression, but being case insensitive	value	regexp
pregMatch	Checks a field value using the configured perl regular expression	value	regExp
captcha	Checks if a field value equals the generated captcha string of the extension "captcha"	none	-
srFreecap	Checks if a field value equals the generated captcha string of the extension "sr_freecap"	none	-
jmRecaptcha	Checks if a field value equals the generated captcha string of the extension "jm_recaptcha"		
mathGuard	Checks if a field value equals the result of the generated question of MathGuard		
dateRange	Checks if a field value is between a configured date range	pattern min max	string string string
date	Checks if a field value is a valid date	pattern	string
time	Checks if a field value is a valid time	pattern	string
fileAllowedTypes	Checks if the filetype of an uploaded file is allowed	allowedTypes	comma seperated
fileMaxCount	Checks if the files uploaded from a field are less than the configured value	maxCount	integer
fileMinCount	Checks if the files uploaded from a field are more than or equal the configured value	minCount	integer
fileMinSize	Checks if the size of an uploaded file is at least the configured value (in Byte)	minSize	integer
fileMaxSize	Checks if the size of an uploaded file is at less than the configured value (in Byte)	maxSize	integer
fileRequired	Checks if a file has been uploaded from this field	none	-

Example

```
plugin.Tx_Formhandler.settings.validators.1 {
  class = Tx_Formhandler_Validator_Default
  config {
    fieldConf {
      firstname.errorCheck.1 = required
      firstname.errorCheck.2 = minLength
      firstname.errorCheck.2.value = 4
      firstname.errorCheck.3 = notDefaultValue
      firstname.errorCheck.3.defaultValue = Enter firstname here
      picture.errorCheck.1 = fileRequired
      picture.errorCheck.2 = fileAllowedTypes
      picture.errorCheck.2.allowedTypes = jpg,png,gif,tiff
    }
  }
}
```

Available subparts to use in the template

###TEMPLATE_FORM[1...x]###	Subpart for form template of a form. The number specifies the step. Single step forms will only contain ###TEMPLATE_FORM1###. Example: ###TEMPLATE_FORM1### for first step

###TEMPLATE_FORM[1...x][SUFFIX]###	Subpart for form template of a multi step form with conditions. The number specifies the step and the suffix specifies the current route. Example: ###TEMPLATE_FORM2_routea### ###TEMPLATE_FORM2_routeb###
###FORM_STARTBLOCK###	Use this in multistep forms and put code being the same in all steps into this subpart. You can use the marker ###FORM_STARTBLOCK### in the following steps and it gets replaced with the content of the subpart ###FORM_STARTBLOCK### in step 1.
###FORM_ENDBLOCK###	Use this in multistep forms and put code being the same in all steps into this subpart. You can use the marker ###FORM_ENDBLOCK### in the following steps and it gets replaced with the content of the subpart ###FORM_ENDBLOCK### in step 1.
###TEMPLATE_EMAIL_ADMIN_PLAIN###	This subpart contains the template for the plain text E-mail sent to the admin if configured.
###TEMPLATE_EMAIL_ADMIN_HTML###	This subpart contains the template for the HTML E-mail sent to the admin if configured.
###TEMPLATE_EMAIL_USER_PLAIN###	This subpart contains the template for the plain text E-mail sent to the user if configured.
###TEMPLATE_EMAIL_USER_HTML###	This subpart contains the template for the HTML E-mail sent to the user if configured.
###TEMPLATE_CONFIRMATION###	If you use the confirmation view after successful form submission you can enter your template in this subpart.
###TEMPLATE_PDF###	Customize the look of generated frontend PDF files. Have a look at the section "How to use HTML as PDF template"
###TEMPLATE_ANTISPAM###	Subpart for Interceptor_AntiSpamFormTime.
###ISSET_[FIELDNAME]###	Use this subpart in your E-mail and confirmation templates to not show fields that were not filled out in the form. There is a how to section in this manual.

Available markers to use in subparts

###value_[fieldname]###	GET/POST value of the given inputfield. Example: <inputtype="text"name="email"value="###value_email###"/>
###LLL:[langkey]###	Marker for translated texts. [langkey] is a key in your translation file.
###error_[fieldname]###	This marker gets replaced with the error messages for the field
###is_error_[fieldname]###	Only gets replaced when an error occurred in this fields. Can be filled with translated content or TypoScript objects. Have a look at the tutorial section for more information.
###is_error###	Only gets replaced when an error occurred in the form. Can be filled with translated content or TypoScript objects. Have a look at the tutorial section for more information.
###REL_URL###	Relative URL to current page
###ABS_URL###	Absolute URL to current page
###submit_nextStep###	Use this in multi step forms to add a submit button pointing to the next step. Example: <inputtype"submit"###submit_nextStep###value="###LLL:next###"/>
###submit_prevStep###	Use this in multi step forms to add a submit button pointing to the previous step. Example: <inputtype"submit"###submit_prevStep###value="###LLL:prev###"/>

###submit_reload###	Use this for upload fields in your form. By clicking the button, the user can upload a file directly. Example: <input type"submit" ###submit_reload### value="Upload file now" />
###FEUSER_[property]###	You can access name, email, ... fields from the logged in user (frontend-user) and fill out the name or email address automatically. Attention: the field name has to be written in CAPITAL letters.
###[fieldname]###	GET/POST value of the given input field.
###required_[fieldname]###	Adds required sign, if this field is marked as required in error checks.
###CAPTCHA###	Only works if you have the "captcha" extension installed. This marker will be replaced with a picture showing some only human readable text. The user has to type the text shown on the picture into a input field in your form.
###RECAPTCHA###	Only works if you have the "jm_recaptcha" extension installed. This marker will be replaced with a picture showing some only human readable text. The user has to type the text shown on the picture into a input field in your form.
###MATHGUARD###	This marker gets replaced with a MathGuard security question. The user has to solve the question and enter the result in a form field.
###ERROR###	Cumulated error messages.
###[fieldname]_minSize###	Minimum file size for uploaded files in this field. The minimum size is defined in the error check.
###[fieldname]_maxSize###	Maximum file size for uploaded files in this field. The maximum size is defined in the error check.
###[fieldname]_allowedTypes###	Allowed file types for uploaded files in this field. The types are defined in error check.
###[fieldname]_maxCount###	Maximum count of files uploaded in this field. The amount is defined in error check.
###[fieldname]_fileCount###	Amount of currently uploaded files via this field.
###[fieldname]_remainingCount###	Remaining amount of uploades via this field.
###[fieldname]_uploadedFiles###	Names of files uploaded via this field. Use this in combination with the wrap settings for file markers in TypoScript.
###total_uploadedFiles###	Names of files uploaded in this form. Use this in combination with the wrap settings for file markers in TypoScript.
###selected_[fieldname]_[fieldvalue]###	Only for use with **dropdowns** Example: **dropdown:** <option value="webdesign" ###selected_topic_webdesign###>Webdesign</option>
###checked_[fieldname]_[fieldvalue]###	Only for use with **checkboxes** and **radiobuttons** Example: **checkbox** <input type="checkbox" name="topic" value="webdesign" ###checked_topic_webdesign###>Webdesign **radiobutton** <input type="radio" name="contact_via" value="email" style="border-style:none;" ###checked_contact_via_email###>email
###curStep###	The current step in a multi step form.
###maxStep###	The last step of a multi step form.
###lastStep###	The last step the user was in a multi step form.
###step_bar###	A sample step bar showing the amount of steps and the current step highlighted.
###ADDITIONAL_MULTISTEP###	Adds the submitted values from other steps as hidden fields to preserve the GET/POST values in order not to lose information of current route in conditional multi step form.

###PRINT_LINK###	Only usable in ###TEMPLATE_CONFIRMATION###. Shows a link to print view.
###CSV_LINK###	Only usable in ###TEMPLATE_CONFIRMATION###. Shows a link to export the submitted values as CSV.
###PDF_LINK###	Only usable in ###TEMPLATE_CONFIRMATION###. Shows a link to export the submitted values as PDF.
###LINK2SHOW###	Only usable in Frontend Listing. Shows a link to detail view.
###DELETE###	Only usable in Frontend Listing. Shows a link to delete a record.
###BACK_LINK###	Only usable in Frontend Listing. Shows a link back to list view.
###TIMESTAMP###	The current timestamp. Useful in combination with Interceptor_AntiSpamFormTime.
###auth_code###	This marker will be filled with the unique token generated by Finisher_GenerateAuthCode

Tutorial

How to set up a simple form

1. Create a HTML template file containing your form. Examples can be found in EXT:formhandler/Examples
2. Create a new page
3. Create a content element with type "General Plugin"
4. Choose "Formhandler" in Dropdown "Plugin"
5. Check that predefined form "Default" is selected.
6. Optional: Choose a translation file containing error messages and translations filled into markers ###LLL:[key]###
7. Optional: Enter settings for E-Mail, required fields or a page to redirect to
8. Save and view the page

How to set up an advanced form

1. Create a HTML template file containing your form. Examples can be found in EXT:formhandler/Examples
2. Create a new page
3. Create an extension template and enter a TypoScript setup entering template file, lang file and other settings.
4. Create a content element with type "General Plugin"
5. Choose "Formhandler" in Dropdown "Plugin"
6. Save and view the page

NOTE: Configuring your form using TypoScript gives you much more possibilities.

How to use predefined forms

1. Create a TypoScript setup using the predef setting

   Example:

```
plugin.Tx_Formhandler.settings.predef.contactform {
      # This name appears in the dropdown selector in plugin record
      name = Contact form
      ...
      ...
}
```

2. Create a content element with type "General Plugin"
3. Choose "Formhandler" in Dropdown "Plugin"
4. Now you can choose the entered predefined form from the "Predefined forms" dropdown.
5. Save and view the page.

HINT: they are many examples available at EXT:formhandler/Examples

How to set up a multistep form

1. Create a HTML template file containing your form. Examples can be found in EXT:formhandler/Examples/MultiStep
2. Create a new page
3. Create an extension template and enter a TypoScript setup entering template file, lang file and other settings.

4. Create a content element with type “General Plugin”
5. Choose “Formhandler” in Dropdown “Plugin”
6. Choose “Multi step” from Dropdown “Type”
7. Save and view the page

NOTE: You can overwrite settings made in plugin.Tx_Formhandler.settings for each step using plugin.Tx_Formhandler.settings.[stepnumber]

How to set up a multistep form with conditions

1. Create a HTML template file containing your form. Examples can be found in EXT:formhandler/Examples/MultiStepConditions
2. Create a new page
3. Create an extension template and enter a TypoScript setup entering template file, lang file and other settings.
4. Create a content element with type “General Plugin”
5. Choose “Formhandler” in Dropdown “Plugin”
6. Choose “Multi step” from Dropdown “Type”
7. Add TypoScript conditions and change the settings plugin.Tx_Formhandler.settings.[stepnumber].templateSuffix
8. Save and view the page

NOTE: Using the TypoScript setting 'templateSuffix', you can define different templates for your steps, emails and PDFs. If you have a form with 3 steps and you want to send emails and PDFs according to the chosen route, you can set the templateSuffix option for step 4 too to ensure that the correct template for emails and PDFs is set.

How to set up spam protection

Enable captcha for your form

1. Make sure that the extension "captcha" is installed
2. Put the input field and the required marker into your template:

```
###error_captchafield###
###CAPTCHA###
<input type="text" name="formhandler[captchafield]" />
```

3. Enter error check for this field in TypoScript

```
plugin.Tx_Formhandler.settings.validators.1 {
  class = Tx_Formhandler_Validator_Default
  config {
    fieldConf {
      captchafield.errorCheck.1 = captcha
    }
  }
}
```

Enable sr_freecap for your form

1. Make sure that the extension "sr_freecap" is installed
2. Put the required subpart into your template. You can change the name of the input field and the HTML code

```
<!--###CAPTCHA_INSERT### this subpart is removed if CAPTCHA is not enabled! -->
      <div>
            <label for="freecapfield">###SR_FREECAP_NOTICE###</label>
      <div class="clear"></div>
```

```
        ###SR_FREECAP_CANT_READ###
            <div class="clear"></div>
            <input type="text" size="15" id="freecapfield" name="formhandler[freecapfield]"
title="###SR_FREECAP_NOTICE###" value="">
            ###SR_FREECAP_IMAGE###
      </div>
<!--###CAPTCHA_INSERT###-->
```

3. Enter error check for this field in TypoScript

```
plugin.Tx_Formhandler.settings.validators.1 {
  class = Tx_Formhandler_Validator_Default
  config {
    fieldConf {
      freecapfield.errorCheck.1 = srFreecap
    }
  }
}
```

Enable jm_recaptcha for your form

1. Make sure that the extension "jm_recaptcha" is installed
2. Register an account at http://recaptcha.net/ to get a public and a private key for your domain.
3. Enter the keys received from http://recaptcha.net/ in TypoScript:

```
plugin.tx_jmrecaptcha {
  public_key = xxx
  private_key = xxx
}
```

4. Add marker to template:

```
###error_recaptcha_response_field###
###RECAPTCHA###
```

5. Enter error check for this field in TypoScript

```
plugin.Tx_Formhandler.settings.validators.1 {
  class = Tx_Formhandler_Validator_Default
  config {
    fieldConf {
      recaptcha_response_field.errorCheck.1 = jmRecpatcha
    }
  }
}
```

Enable mathGuard for your form

1. Add marker to template

```
###error_mathguard_answer###
###MATHGUARD###
```

2. Enter error check for this field in TypoScript

```
plugin.Tx_Formhandler.settings.validators.1 {
  class = Tx_Formhandler_Validator_Default
  config {
    fieldConf {
      mathguard_answer.errorCheck.1 = mathGuard
```

```
        }
      }
    }
```

How to set up error checks

1. Add markers like ###error_[fieldname]### to your template
2. Set up a validator in TypoScript

```
plugin.Tx_Formhandler.settings.validators.1 {
  class = Tx_Formhandler_Validator_Default
  config {
    fieldConf {
            field1 {
       errorCheck.1 = required
       errorCheck.2 = maxLength
       errorCheck.2.value = 50
      }
      field2 {
       errorCheck.1 = required
       errorCheck.2 = email
      }
    }
  }
}
```

3. Add entries for error messages to your translation file

```
<label index="error_field1_required">Field1 is required!</label>
<label index="error_field1_maxLength">Field1 has to be shorter than ###value### characters!</label>
<label index="error_field2_required">Field2 is required!</label>
<label index="error_field2_email">Field2 is no valid e-mail!</label>
```

NOTE: The marker ###value### in the error message is filled with the value of the parameter "value" for error check "maxLength". If an error check requires parameters, you can use markers like ###[parametername]### in the translatable message.

Use ###is_error### markers

These markers allow you to display additional content is an error occurred for each form field or globally.

In your HTML template you can add these markers:

```
###is_error###
###is_error_[fieldname]###
```

Formhandler will search for content to replace these markers in the translation file and in the TypoScript setup.

In the translation file enter the values like:

```
<label index="is_error_field1">Error occurred in field1!</label>
```

A possible TypoScript configuration looks like this:

```
plugin.Tx_Formhandler.settings {
  isErrorMarker {
    global = Global message if an error occurred (filled into ###is_error###)
    field1 = TEXT
    field1.value = Some message (filled into ###is_error_field1###)
  }
}
```

How to fill your own markers via TypoScript

Coding 1:

1. Add the marker into your template

```
###myMarker###
```

2. Register the marker in TypoScript. (3 examples available)

```
plugin.Tx_Formhandler.settings.predef.myForm.markers.myMarker_1 = TEXT
plugin.Tx_Formhandler.settings.predef.myForm.markers.myMarker_1.value = My marker content

# getText function
plugin.Tx_Formhandler.settings.predef.myForm.markers.myMarker_2 = TEXT
plugin.Tx_Formhandler.settings.predef.myForm.markers.myMarker_2.data = getIndpEnv:HTTP_HOST

# Constant insertion
plugin.Tx_Formhandler.settings.predef.myForm.markers.myMarker_3 = TEXT
plugin.Tx_Formhandler.settings.predef.myForm.markers.myMarker_3.value = {$local.myConstant}
```

3. The marker will be filled with “My marker content”

NOTE: You can fill your own markers using any TypoScript object like USER or COA. Just try it out.

Coding 2: Prefill a drop down menu dynamically

1. Let's say you have dropdown menu in your HTML template which you may want to generate dynamically.

```
<select id="changeMe" name="changeMe">
  <options value="">###LLL:selectValue###</option>
  ###options_dropdown###
</select>
```

2. Next, you will need to write / adapt the following TypoScript

```
options_dropdown = CONTENT
options_dropdown{
   table = tx_addresses_domain_model_address
   select {
       pidInList = 1
       orderBy = last_name
       selectFields = uid, first_name, last_name
       # possible conditions
       # where = ( tx_mytable.type='b0' OR tx_mytablet.type='b1' )
   }

   renderObj = COA
   renderObj {

       #value
       10.wrap = <option value="|"
       10 = TEXT
       10.field = uid

       #selected
       12.noTrimWrap = | ###selected_mytable_|###>|
       12 = TEXT
       12.field = uid

       #label
       13 = TEXT
       13.wrap = |</option>
       13.field = first_name
   }
}
```

How to hide unfilled form field values in templates

In your HTML-Template you are able to wrap the field output in e-mails and subpart ###TEMPLATE_CONFIRMATION### with a subpart:

```
<!-- ###ISSET_fax### -->
  ###value_fax###<br/>
<!-- ###ISSET_fax### -->
```

If the user filled out the field "fax" in the form, the text will get shown in the e-mails, otherwise it will not be added.

You have the possibility to add some conditions:

```
<!-- ###ISSET_fax&&phone### -->
  ###value_fax###<br/>
<!-- ###ISSET_fax&&phone### -->
<!-- ###ISSET_fax&&!phone### -->
  ###fax###<br/>
<!-- ###ISSET_fax&&!phone### -->
<!-- ###ISSET_fax||phone### -->
  ###fax###<br/>
<!-- ###ISSET_fax||phone### -->
```

As you see, you can use the operators (&&,||,!) to do some simple checks.

NOTE: The case of the fieldname in ISSET subpart and marker MUST match.

good:

```
<!-- ###ISSET_EMAIL### -->
###value_EMAIL###
<!-- ###ISSET_EMAIL### -->
<!-- ###ISSET_email### -->
###value _email###
<!-- ###ISSET_email### -->
```

bad:

```
<!-- ###ISSET_EMAIL### -->
###value _email###
<!-- ###ISSET_EMAIL### -->
<!-- ###ISSET_email### -->
###value _EMAIL###
<!-- ###ISSET_email### -->
```

How to set up multi language forms

In the template file:

Specify markers for the multi language texts.

example:

```
<!-- ###TEMPLATE_FORM### begin -->
<form name="form" method="post" action="###REL_URL###">
<input type="hidden" name="submitted" value="1">

<table>
<tr>
<td>###LLL:username###</td>
<td><input type="text" name="username" value="###value_username###"></td>
</tr>
<tr>
<td>###LLL:password###</td>
<td><input type="password" name="password" value="###value_password###"></td>
</tr>
<tr>
<td colspan=”2”><input type="submit" name="submit" value="###LLL:submit###"></td>
</tr>
</table>
</form>
<!-- ###TEMPLATE_FORM### end -->
```

Your custom language file:

```
<?xml version="1.0" encoding="utf-8" standalone="yes" ?>
<T3locallang>
      <meta type="array">
            <description>Language labels for my form</description>
      </meta>
      <data type="array">
            <languageKey index="default" type="array">
                  <label index="username">Username:</label>
                  <label index="password">Password:</label>
                  <label index="submit">Login</label>
            </languageKey>
            <languageKey index="de" type="array">
                  <label index="username">Benutzername:</label>
                  <label index="password">Passwort:</label>
                  <label index="submit">Anmelden</label>
            </languageKey>
      </data>
</T3locallang>
```

TypoScript:

You can specifiy the path to your translation file in any setting "langFile" associated with "formhandler".

Example:

```
plugin.Tx_Formhandler.settings.langFile = fileadmin/lang/myLangFile.xml
```

How to load data from DB to use in Finisher_DB

It may be useful to extend the set of data submitted in the form before inserting them in the database. You may want to fetch additional data from the database, the session, the GET / POST parameters or from something else. This can be done quite easily since the key "mapping" can understand TypoScript.

Let's have a look at the following lines. They are 4 examples, field_demo_1, field_demo_2, field_demo_3, field_demo_4.

```
2.class = Tx_Formhandler_Finisher_DB
2.config {
    table = tt_content
    key = uid
    fields {

        # Retrieves data from the GET / POST parameters
        field_demo_1 {
            mapping = TEXT
            mapping.data = GPvar:formhandler|firstname
        }

        # Retrieves data from the database
        field_demo_2 {
            mapping = TEXT
            mapping {
                dataWrap = DB:tx_cuso_courses:{GPvar:formhandler|course_uid}:title
                insertData = 1
                wrap3 = {|}
            }
        }

        # Retrieves data from the session
        field_demo_3 {
            mapping = TEXT
            mapping.data = TSFE:fe_user|user|last_name
        }

        # Some more cool stuff
        field_demo_4 {
            mapping = COA
            mapping {
                10 = TEXT
                10 {
                    data = GPvar:formhandler|firstname
                }
                20 = TEXT
                20 {
                    data = GPvar:formhandler|lastname
                    noTrimWrap = | |: |
                }
                30 = TEXT
                30 {
```

```
                    dataWrap = DB:tx_cuso_courses:{GPvar:formhandler|course_uid}:title
                    insertData = 1
                    wrap3 = {|}
                }
            }
        }
    }
}
```

How to store data into more than one DB table

To store data into different tables you have to chain different Finisher_DBs. You can use the setting special to access the uid of a previously inserted record.

```
finishers {
    1.class = Finisher_DB
    1.config {
      table = fe_users
      fields {
            username.mapping = lastname
     }
    }
    2.class = Finisher_DB
    2.config {
      table = tt_address
      fields {
            pid.special = inserted_uid
            pid.special.table = fe_users
            email.mapping = email
            first_name.mapping = firstname
            last_name.mapping = lastname
     }
    }
}
```

The first Finisher_DB stores data into the table fe_users. The second one create a record in the child table tt_address. The pid field of this record is set to the uid of the inserted record in fe_users.

How to use HTML as PDF template

This only works if Generator_TCPDF is used.

```
<!-- ###TEMPLATE_PDF### -->
<table>
      <!-- ###pid###, ###submission_date### and ###ip### are predefined markers to use in this subpart
-->
      <tr>
            <td>###LLL:pid###</td>
            <td>###pid###</td>
      </tr>
      <tr>
            <td>###LLL:ip###</td>
            <td>###ip###</td>
      </tr>
      <tr>
            <td>###LLL:submission_date###</td>
            <td>###submission_date###</td>
      </tr>
      <tr>
            <td>###LLL:[fieldname]###</td>
            <td>###value_[fieldname]###</td>
      </tr>
</table>
<!-- ###TEMPLATE_PDF### -->
```

How to use your own controller

This is for advanced users only!

If you want to write your own controller make sure it inherits from Tx_Formhandler_AbstractController.

You can set your controller using TypoScript:

```
tt_content.list.20.formhandler_pi1.controller = Tx_MyExt_MyController
```

NOTE: Your custom controller file (in this case: Tx_MyExt_MyController.php) has to be located in a directory 'Classes' in your extension. Otherwise it will not be found by the component manager!
(EXT:myext/Classes/Tx_MyExt_MyController.php)

How to upgrade from th_mailformplus to Formhandler

There are many forms out there which use th_mailformplus. If you ask yourself, what would happen if you did an update to Formhandler, here is a short explanation on how to change the TypoScript of th_mailformplus to work with MailformplusMVC. This is not a complete tutorial, it only shows the basic differences. Many of the TypoScript settings are named equally, but some also have changed, were removed or didn't exist in th_mailformplus.

A TypoScript setup for th_mailformplus may look like this:

```
plugin.tx_thmailformplus_pi1 {
	langFile = typo3conf/ext/th_mailformplus/example_form/singlepage_forms/improved_demo_lang.php

	default {
		email_to = admin@host.com
		email_sendtouser = email
		email_subject_user = Your contact request
		email_sender = noreply@host.com
	}

	fieldConf {
		name {
			errorCheck = required
		}
		email {
			errorCheck = required,email
		}
		text {
			errorCheck = required
		}
	}
}
```

In this configuration a translation file is included, error checks are defined and settings for email sending to an admin and the user are made.

The same configuration with Formhandler will look this way:

```
plugin.Tx_Formhandler.settings {
	langFile = typo3conf/ext/th_mailformplus/example_form/singlepage_forms/improved_demo_lang.php

	validators.1.class = Validator_Default
	validators.1.config {
		name {
			errorCheck.1 = required
		}
		email {
			errorCheck.1 = required
			errorCheck.2 = email
		}
		text {
			errorCheck = required
		}
	}

	finishers.1.class = FinisherMail
	finishers.1.config {
		admin {
```

```
                to_email = admin@host.com
                sender_email = noreply@host.com
            }
            user {
                to_email = email
                sender_email = noreply@host.com
                subject = Your contact request
            }
        }
}
```

As you see, the configuration looks similar, but Formhandler uses blocks like “validators” or “finishers” to make it possible to group any number of classes. So that it is possible to let “finishers.1” be a class to send emails and “finishers.2” be a class to store data into database.

When you compare the TypoScript options of th_mailformplus with Formhandler you will find out that they are quite similar and you should have no problems with writing configuration for Formhandler if you knew th_mailformplus before.

Known problems

- When using AJAX remove links for uploaded files the file markers such as ###[fieldname]_fileCount###do not get updated because the request only refreshes the content of the marker ###[fieldname]_uploadedFiles###.A possible solution would be to reload the whole form using the AJAX request.
- Some problems may occur in multistep handling, when using the submit_reload feature. We are aware of this problem and will solve it in the future.

To-Do list

- Add a 1-2-3 wizard to generate form templates, translation files and TypoScript setup code.
- Think about integration of "extBase" architecture and usage of Fluid.

ChangeLog

– 1.0.0: beta release.