EXT: Universal Formhandler

Extension Key: formhandler
Language: en

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.

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 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

 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.

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”.

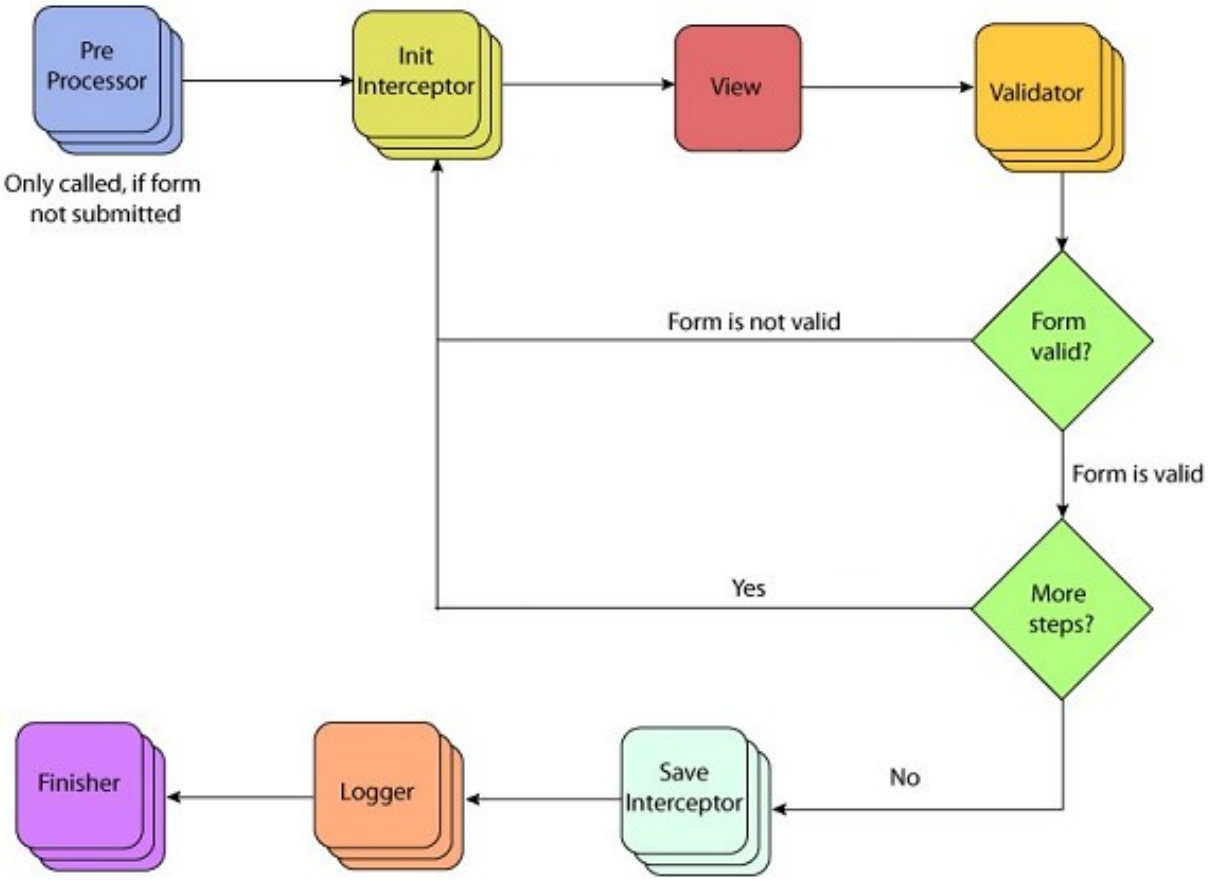
First of all Formhandler has a more up-to-date architecture, which allows a more modular approach having different classes for different tasks. Another advantage of this architecture is a smaller code base making updates and adding of additional features much less time consuming.

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.

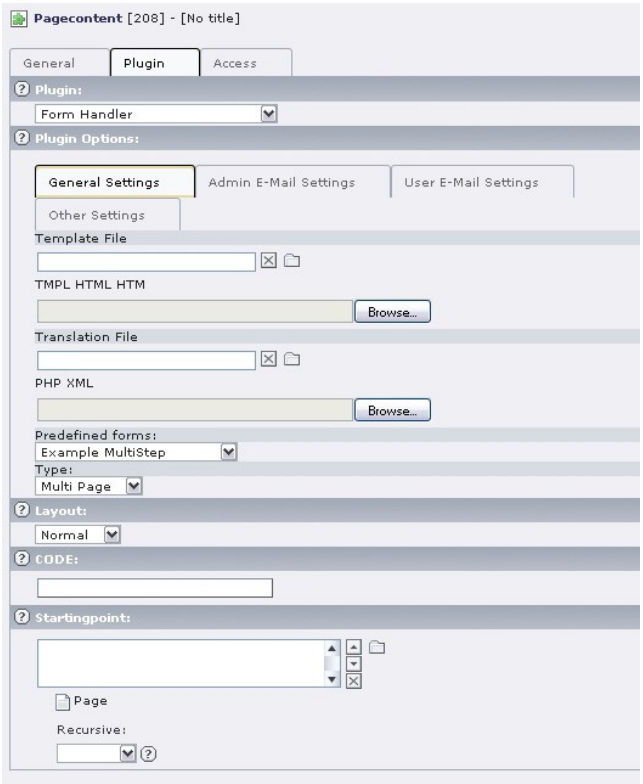
 Have a look at the section “How does this extension work?” and the TypoScript configuration options. For a more practical way of explanation, have a look at the “How to ...” sections at the end of the manual or the folder “Examples” in the extension directory.

Screenshots

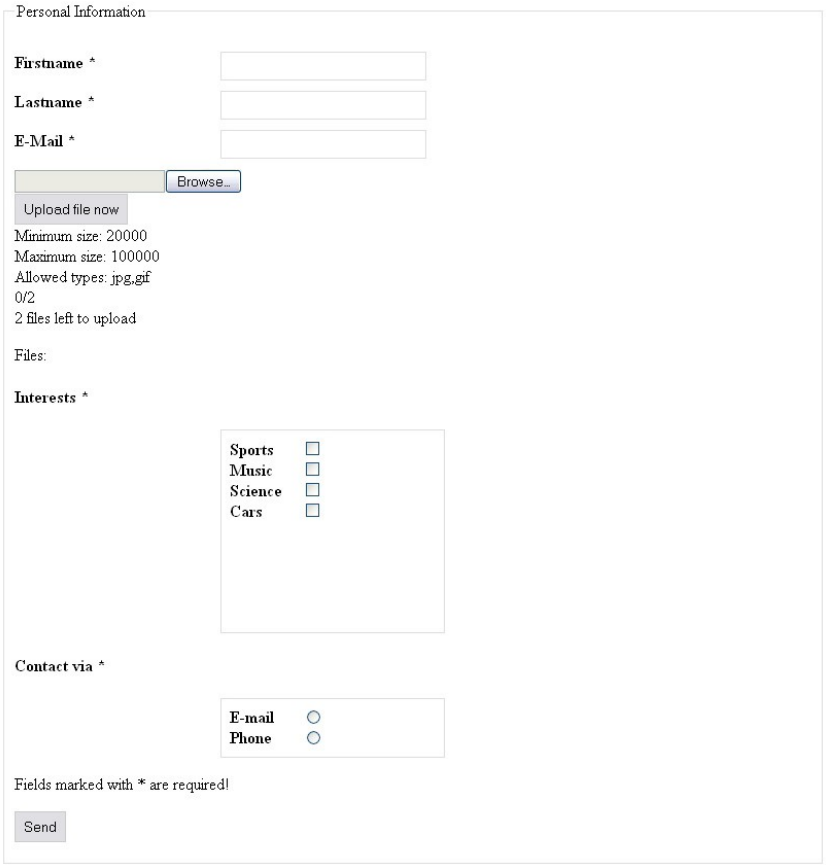


This is how Formhandler works. Quite simple, isn't it? For more information about the rendering process have a look at the section “How does this extension work?”.

 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.



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
```

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	

masterTemplateFile	String/TS object/Array	Path to master template file. See section How to use a master template for details.
		
		Example:
		
		`#single master template`
		`masterTemplateFile =`
		`fileadmin/templates/formmaster.html`
		
		`#use a cObject`
		`masterTemplateFile = TEXT`
		`masterTemplateFile {`
		`  value = fileadmin/mymastertemplate.html`
		`}`
		
		`#multiple master templates`
		`masterTemplateFile {`
		`  1 = fileadmin/templates/formmaster.html`
		`  2 = fileadmin/templates/master_for_contactform.html`
		`}`
		
		`#use cObjects for multiple master templates`
		`masterTemplateFile {`
		`  1 = TEXT`
		`  1.value = fileadmin/templates/formmaster.html`
		`  2 = fileadmin/templates/master_for_contactform.html`
		`}`	

Property:	Data type:	Description:	Default:

langFile	String/TS object/Array	Path to language files
		
		Example:
		
		`#single lang file`
		`langFile = fileadmin/lang/shared.xml`
		
		`#use a cObject`
		`langFile = TEXT`
		`langFile.value = fileadmin/lang/shared.xml`
		
		`#multiple lang files`
		`langFile {`
		`  1 = fileadmin/lang/shared.xml`
		`  2 = fileadmin/lang/forms/mailformxy.xml`
		`}`
		
		`#use cObjects for multiple master templates`
		`langFile {`
		`  1 = TEXT`
		`  1.value = fileadmin/lang/shared.xml`
		`  2 = fileadmin/lang/forms/mailformxy.xml`
		`}`	

additionalIncludePaths	Array	The component manager will search in paths enteres here for classes.
		
		Example:
		
		`plugin.Tx_Formhandler.settings {`
		`  additionalIncludePaths {`
		`  1 = fileadmin/templates/scripts`
		`  }`
		`}`
		
		If you use “predef” for your form, you can also use it for “additionalIncludePaths”:
		
		`plugin.Tx_Formhandler.settings.predef.example {`
		`  additionalIncludePaths {`
		`  1 = fileadmin/templates/scripts`
		`  }`
		`}`
		
		NOTE: additionalIncludePaths on global level and additionalIncludePaths on predef level will be merged. So you can define include paths for all your forms and special ones for single forms.
		
		Use this if you create your own components and don't want to create an extension for that. (See section How to use your own components for details)	

cssFile	String/Array	Path to stylesheet file(s).
		
		Example:
		
		`cssFile = fileadmin/css/forms.css`
		
		`cssFile {`
		`  1 = fileadmin/css/forms.css`
		`  2 = fileadmin/css/contactform.css`
		`}`	

jsFile	String/Array	Path to JavaScript file(s).
		
		Example:
		
		`jsFile = fileadmin/js/jquery.js`
		
		`jsFile {`
		`  1 = fileadmin/js/jquery.js`
		`  2 = fileadmin/js/jquery_plugin.js`
		`}`	

Property:	Data type:	Description:	Default:
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/cObj	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.	
singleFileMarkerTemplate.singleWrap	wrap	Enter something to be wrapped around each filename in a file marker for a single form field.	
singleFileMarkerTemplate.showThumbnails	Boolean/integer (0/1/2)	Shows thumbnail of uploaded file instead of filename. Set to 2 to display thumbnail and 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/integer (0/1/2)	Shows thumbnail of uploaded file instead of filename. Set to 2 to display thumbnail and filename.	0
files.uploadFolder	String	Path to a custom upload folder	uploads/formhandler/tmp

Property:	Data type:	Description:	Default:
files.enableAjaxFileRemoval	Boolean (0/1)	You need to install the extension xajax in order to use this feature! 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.	
files.enableFileRemoval	Boolean (0/1)	Enables file removal without having xajax installed. It will just display an “X” or anything entered in “files.customRemovalText” next to a file name of an uploaded file, so the user can remove it.	
files.customRemovalText	String/cObj	Enter a custom text shown as the text of the remove link. Used in combination with “enableAjaxFileRemoval”.	X
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”	

Property:	Data type:	Description:	Default:

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###)`
		`  default = class="error" (filled into ###is_error_[fieldname]###`
		`  field1 = TEXT`
		`  field1.value = Some message (filled into ###is_error_field1###)`
		`  }`
		`}`	

fillValueMarkersBeforeLangMarkers	Boolean (0/1)	If set to 1, the value markers (###value_[field]###) are replaced before the language markers (###LLL:[field]###).	0

arrayValueSeparator	String/cObj	If using f.e. A checkbox array named "cb", the marker ###value_cb### will contain a comma separated list of the selected values. Use this option to specifiy your own separator.
		
		Example:
		
		`plugin.Tx_Formhandler.settings {`
		`  arrayValueSeparator = |`
		`}`
			,

[plugin.Tx_Formhandler .settings]

General Options for any component

Any component and complete component arrays can be disabled via TypoScript.

Examples

```
plugin.Tx_Formhandler.settings {
  validators.1.class = Validator_Default
  validators.1.disable = 1
  validators.1.config.fieldConf {
    company.errorCheck.1 = required
    name.errorCheck.1 = required
  }

  finishers {
    disable = 1
    1.class = Finisher_Mail
    1.config {
      admin {
        to_email = rf@typoheads.at
        subject = TEXT
        subject.data = LLL:fileadmin/templates/ext/formhandler/lang.xml:email_admin_subject
        sender_email = email@example.tld
        attachment = picture
      }
      user {
        to_email = email
        subject = TEXT
        subject.data = LLL:fileadmin/templates/ext/formhandler/lang.xml:email_user_subject
        sender_email = email@example.tld
      }
    }
    2.class = Finisher_SubmittedOK
    2.config {
      returns = 1
    }
  }
}
```

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_DB`
		`}`
		`loggers.2 {`
		`  class = Tx_MyPackage_Logger`
		`  config {`
		`  table = tx_mypackage_log`
		`  }`
		`}`	

x.config	Array		

[plugin.Tx_Formhandler .settings.loggers]

NOTE: The Logger_DB will log into database and store the UID of the inserted record into the user session. You can access this UID later on using:

```
$data = $GLOBALS['TSFE']->fe_user->sesData;
$uid = $data['inserted_uid'];
```

NOTE: The Logger_DB will be added by the controller automatically, so you don't have to specify it. However this not valid for the loggers setting of Interceptor_AntiSpamFormTime and Interceptor_IPBlocking!

Logger_DB

Will log into tx_formhandler_log. Logging can be accessed via the “Formhandler” backend module.

Logger_DevLog

Will log into the devlog table. Use an extension like “devlog” to access the logged messages.

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 = Tx_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

 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 datebase), refer to the section “How to fill your own markers via TypoScript”. There is a neat example about how to generate the “option” tags.

prefill dynamically a dropdown menu (e.g. from the datebase), 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

PreProcessor_LoadGetPost

Loads GET/POST values.

The controller clears GET/POST params when the form is called for the first time. If you want to pass values to the form, you can insert this preProcessor to load the values.

This component doesn't need any further configuration.

PreProcessor_LoadDB

Prefills form fields with values from configured database fields. Config is done for each step and for each form field.

Example:

```
preProcessors {
    1.class = Tx_Formhandler_PreProcessor_LoadDB
    1.config
        1 {
            contact_via.mapping = email
        }

        select {
            table = my_table_name
        }
    }
}
```

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

Property:	Data type:	Description:	Default:
mapping	String | TypoScript Object	Field name.	-
Separator	String | TypoScript Object	The field separator, e.g. comma	

[
plugin.Tx_Formhandler.settings.preProcessors.x.class = Tx_Formhandler_PreProcessor_LoadDB
plugin.Tx_Formhandler.settings.preProcessors.x.config.select.
]

Property:	Data type:	Description:	Default:
selectFields	String | TypoScript Object		*
table	String | TypoScript Object	The table to fetch data from. Required.	
where	String | TypoScript Object		
groupBy	String | TypoScript Object		
orderBy	String | TypoScript Object		
limit	String | TypoScript Object		

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_Filtreatment`
		`}`
		`initInterceptors.2 {`
		`  class = Tx_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	

Property:	Data type:	Description:	Default:

x.config	Array	class name of 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 = Tx_MyPackage_ Interceptor`
		`  config {`
		`  param = value`
		`  }`
		`}`	

[plugin.Tx_Formhandler.settings.saveInterceptors]

Interceptor_Filtreatment

Removes malicious code from submitted values to prevent XSS attacks. This Interceptor is added automatically by Formhandler. It removes malicious code and by default some characters from the input values. For a list of characters, which will be removed, have a look at the following configuration section.

You can use the removeChars setting to remove bad words or characters by entering a comma seperated list or using a cObject like USER to connect to a service.

Example:

```
plugin.Tx_Formhandler.settings.initInterceptors.1.class = Tx_Formhandler_Interceptor_Filtreatment
plugin.Tx_Formhandler.settings.initInterceptors.1.config.fieldConf {
      name.removeChars = %,&,/
      email.removeChars = TEXT
      email.removeChars.value = ,|<|>
      email.separator = |
      company = USER
      company.userFunc = user_myClass->user_getRemoveWords
}
```

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

Property:	Data type:	Description:	Default:

fieldConf	array	Settings per form field.
		
		Key = field name or “global”
		Value = comma seperated list or cObject.
		
		Just configure “removeChars.disable=1” per field or globally to disable the removal. If you set specific configuration for a single field, the removal will take place even if it is disabled globally.
		
		Example:
		
		`fieldConf {`
		`  global.removeChars.disable = 1`
		
		`  email.removeChars = a,b,c`
		`  company.removeChars = ,|*|<|>`
		`  company.separator = |`
		`  subject.removeChars = TEXT`
		`  subject.removeChars.value = 3f4f5`
		`  subject.separator = f`
		`}`
			Default removal characters for each field are:
			
			<
			>
			'
			“

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/cObj	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,hoursand 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,hoursand days	
global	array	Settings for limiting submissions for a form. SAME SETTINGS AS FOR IP.	

loggers	array	Same as for the global loggers.
		
		Example:
		
		loggers {
		1.class = Logger_DB
		2.class = Logger_DevLog
		}	

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,hoursand days	-
maxTime.value	int	value	-
maxTime.unit	string	unit for the minimum time. Unit can be seconds,minutes,hoursand days	-
redirectPage	mixed	Page ID or URL to redirect in case Spam is detected	

loggers	array	Same as for the global loggers.
		Example:
		loggers {
		1.class = Logger_DB
		2.class = Logger_DevLog
		}	

Interceptor_CombineFields

With this interceptor you can combine values of different form fields into a new field, which will be accessible by other components later.

Example:

```
initInterceptors {
```

```
      1.class = Interceptor_CombineFields
      1.config {
            combineFields {
                  combined {
                        fields {
                           1 = name
                           2 = company
                        }
                        separator = ,
                        hideEmptyValues = 1
                  }
            }
      }
}
```

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

Property:	Data type:	Description:	Default:

combineFields	array	Array containing options which fields to combine.
		
		Scheme:
		
		`[newFieldname] {`
		`  [options]`
		`}`
			-

combineFields. [fieldname].fields	array	Array containing existing field names. The values of these fields will be combined into the new field [fieldname].	
combineFields. [fieldname].separator	string	Set the separator to use when combining the fields.	Space character
combineFields. [fieldname].hideEmptyValues	boolean (0/1)	If set to 1, empty values will be ignored when combining.	

Interceptor_TranslateFields

This interceptor will load a given translation. You can use values of form fields to load dynamic translations.

Example:

```
initInterceptors {
      1.class = Interceptor_TranslateFields
      1.config {
            translateFields {
                  newField {
                        langKey = label_salutation_|
                        field = salutation
                  }
            }
      }
}
```

This example will load the translation for the selected salutation into the field “newField”.

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

Property:	Data type:	Description:	Default:

translateFields	array	Array containing translation options.
		
		Scheme:
		
		`[newFieldname] {`
		`  [options]`
		`}`
			-

translateFields. [fieldname].langKey	string	Key in translation file. Use the “|” character to load content of “field” into the langKey.	
translateFields. [fieldname].field	string	Name of a form field. The value of this field will replace the “|” in langKey.	

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.`
		`  */`
		
		`  config {`
		`  disableErrorCheckFields = firstname,lastname`
		`  fieldConf {`
		`  firstname.errorCheck {`
		`  1 = required`
		`  2 = maxLength`
		`  2.value = 20`
		`  }`
		`  email.errorCheck {`
		`  1 = required`
		`  2 = email`
		`  }`
		`  }`
		`  }`
		`}`
		`validators.2 {`
		`  class = Tx_MyPackage_ Validator`
		`  config {`
		`  param = value`
		`  }`
		`}`	

x.config	Array		

[plugin.Tx_Formhandler.settings.validators]

Have a look at the section "Available error checks".

Validator_Default

This is the default validator. It supports all available error checks.

[
plugin.Tx_Formhandler.settings.validators.x.class = Tx_Formhandler_Validator_Default
plugin.Tx_Formhandler.settings.validators.x.config.
]

Property:	Data type:	Description:	Default:
disableErrorCheckFields	Comma seperated list	List of fieldnames which should not get validated	-
fieldConf	Array	Array of fields to validate	
fieldConf.[fieldname].	Array	Array containing the error check options. Example `fieldConf { name { errorCheck { 1 = required 2 = maxLength 2.value = 50 } } email { errorCheck { 1 = required 2 = email } } }`	

Validator_Ajax

This validator is called when using AJAX validation. Please do not add this class manually to your TypoScript.

Generators

Generators are used to use the submitted form data to save it into files. These files can be attached to an email or downloaded on the confirmation page.

Example:

```
finishers.3.class = Tx_Formhandler_Finisher_SubmittedOK
finishers.3.config {
  returns = 1
  actions {
    pdf {
      class = Generator_WebkitPdf
      config {
        pid = 23
      }
    }
    csv {
      class = Generator_Csv
    }
  }
}
```

Generator_WebkitPdf

Generates PDF files using the extension 'webkitpdf'.

Configuration only needs the ID of the page where a webkitpdf plugin record got inserted.

Configuration only needs the ID of the page where a webkitpdf plugin record got inserted.

Property:	Data type:	Description:	Default:

x.class	String	Only configuration option for Generator_WebkitPdf is 'pid', which has to be the ID of a page with a 'webkitpdf' plugin record.
		
		Example:
		
		`... {`
		`  class = Generator_WebkitPdf`
		`  config {`
		`  pid = 23`
		`  }`
		`}`	

x.config	Array		

Generator_PdfGenerator

Generates PDF files using the extension 'pdf_generator2'.

Configuration only needs the page type for PDF generation.

Property:	Data type:	Description:	Default:

x.class	String	Configuration options for Generator_PdfGenerator are 'type' and 'target'. 'type' is the page type configured in the pdf_generator2 plugin and 'target' is the target of the link.
		
		Example:
		
		`... {`
		`  class = Generator_PdfGenerator`
		`  config {`
		`  type = 123`
		`  target = _blank`
		`  }`
		`}`	

x.config	Array		
x.config.type	int	Page type for PDF generation.	123
x.config.target	string	Target for the PDF link	_blank

Generator_TcPdf

Generates PDF files using TCPDF.

Subpart ###TEMPLATE_PDF### is required for this component! In this subpart you can adjust the PDF ouput using HTML.

Property:	Data type:	Description:	Default:

x.class	String	Example:
		
		`... {`
		`  class = Generator_TcPdf`
		`  config {`
		`  storeInTempFile = 1`
		`  checkBinaryCrLf = lastname,firstname`
		`  returnFileName = 1`
		`  }`
		`}`	

x.config	Array		

Property:	Data type:	Description:	Default:
x.config.storeInTempFile	int (1/0)	If set, the generator will save the output in a temporary file rather than sending it directly as content. Needed for usage with Finisher_Mail in combination with “returnFileName”	
x.config.checkBinaryCrLf	List of field names	Line breaks in the values of the entered fields will be replaced with HTML break tags.	
x.config.returnFileName	int (1/0)	If set, the generator will return only the file name of the generated PDF. Needed for usage with Finisher_Mail in combination with “storeInTempFile”	

Generator_Csv

Just enter the class name. No configuration required for this component,

Finishers

Property:	Data type:	Description:	Default:

x.class	String	Enter as many finishers as you like. Each entry requires a class name of the finisher. Optional you can enter specific configuration for the finisher in the confg section. More information about available finishers and their settings is provided just below this table.
		
		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 {`
		
		`  }`
		`}`	

x.config	Array		

[plugin.Tx_Formhandler.settings.finishers]

Finisher_ClearCache

Clear the page cache. If no further configuration is given, the current page's cache will be cleared.

Property:	Data type:	Description:	Default:

cacheCmd	int/string/cObj	Enter as many finishers as you like. Each entry requires a class name of the finisher. Optional you can enter specific configuration for the finisher in the confg section. More information about available finishers and their settings is provided just below this table.
		
		Example:
		
		`finishers.1.class = Tx_Formhandler_Finisher_ClearCache`
		
		`# The cache of page 15 will be cleared`
		`finishers.1.config.cacheCmd = 15`
		
		`# cObject is supported...`
		`finishers.1.config.cacheCmd = TEXT`
		`finishers.1.config.cacheCmd.data = GP:someparameter`
		
		`# for other cacheCmds see phpdoc in t3lib_TCEmain->clear_cacheCmd()`
			Current page

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_SubmittedOK

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

[
plugin.Tx_Formhandler.settings.finishers.x.class = Tx_Formhandler_Finisher_SubmittedOK
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!	-

actions	Array	Container for various ouput actions like PDF or CSV. Finisher_SubmittedOK will call each class entered in the actions and fill a marker ###[action]_LINK### with the result.
		
		Example:
		actions {
		pdf {
		class = Generator_WebkitPdf
		config.pid = 23
		}
		}
		
		Generator_WebkitPdf will fill the marker ###PDF_LINK### with a link to generatte a PDF file of the submitted ok page.
		
		Same for CSV:
		actions {
		csv {
		class = Generator_Csv
		}
		}
		
		You can define your own actions with your own generators if you want to.	

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.
]

The data saved to DB is accessible through template markers, e.g.: ###value_saveDB|tablename|field### or ###value_saveDB|fe_users|username###. If data is saved to the same table more than once (i.e. multiple rows are created or updated), a number suffix is added to the table name, e.g.: ###value_saveDB|fe_users_2|username###.

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].nullIfEmpty	int(1/0)	If the user did not enter something in the form field, a NULL value is inserted into database	
Fields.[db_field].zeroIfEmpty	int(1/0)	If the user did not enter something in the form field, a zero value (int: 0) is inserted into database	
fields.[db_field].separator	string	If the field value is an array 8e.g. Checkboxes) you can enter a separator 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

mailer	array	Optional:
		
		Use your own class to send the emails.
		
		Example:
		
		mailer {
		  class = Mailer_HtmlMail
		  config {
		  ...
		  }
		}
		
		A mailer must fulfill the given MailerInterface.
			mailer {
			  class = Mailer_HtmlMail
			}

admin	array	Settings for the mail sent to site admin	-
admin.header	string/cObj	Manually set the email header. All options can be strings, TypoScript objects or names of form fields.	-
admin.to_email	string/cObj	Comma seperated list of email addresses. Can be addresses or form fields.	-
admin.to_name	string/cObj	Comma seperated list of names to the according addresses in to_email. Can be name or form fields.	-
admin.replyto_email	string/cObj	Reply to email address	-
admin.replyto_name	string/cObj	Name of the according replyto_email	-
admin.cc_email	string/cObj	A CC email will be sent to this address.	-
admin.cc_name	string/cObj	Name added to the cc_email.	-
admin.bcc_email	string/cObj	A BCC email will be sent to this address.	-
admin.bcc_name	string/cObj	Name added to the bcc_email.	-
admin.sender_email	string/cObj	Email address to be set as sender of the email. Can be addresses or form fields.	-
admin.sender_name	string/cObj	Name of the email sender. Can be name or form fields.	-
admin.subject	string/cObj	Subject of the email	

admin.return_path	string/cObj	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_WebkitPdf	
admin.attachPDF.config	array	Configuration for the generator class	
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
		}
		
		This will not work for usage with Generator_WebkitPdf since the file naming will be done by the extension 'webkitpdf'
			formhandler_

admin.plain.arrayValueSeparator	string/cObj	Specify your own separator for array values in this type of emails. See `plugin.Tx_Formhandler.settings.arrayValueSeparator` for more details.	,
admin.html.arrayValueSeparator	string/cObj	`Specify your own separator for array values in this type of emails. See plugin.Tx_Formhandler.settings.arrayValueSeparator for more details.`	,
user	array	Settings for the email sent to the user. SAME SETTINGS AS FOR ADMIN	

Backend Module

Add this TypoScript as page Ts config or user TS.

Property:	Data type:	Description:	Default:

csv	List of field names	Enter a list of field names to export in CSV. This way you will not have to select the fields manually each time you export.
		
		Example:
		
		tx_formhandler_mod1.config {
		  csv = firstname, lastname, email, address
		}	

pdf	List of field names	Enter a list of field names to export in PDF. This way you will not have to select the fields manually each time you export.
		
		Example:
		
		tx_formhandler_mod1.config {
		  pdf = firstname, lastname, email, address
		}	

[tx_formhandler_mod1.config]

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/cObj
containsOne	Checks if a field contains at least one of the configured values	words	comma seperated/cObj
containsOnly	Checks if a field contains only the configured values	words	comma seperated/cObj
containsAll	Checks if a field contains all of the configured values	words	comma seperated/cObj
equals	Checks if a field equals the configured value	word	string/cObj
equalsField	Checks if a field value equals another field value	field	string/cObj
notEqualsField	Checks if a field value does not equal another field value	field	string/cObj
notDefaultValue	Checks if a field equals a default value set by a pre processor	defaultValue	string/cObj
minValue	Checks if the value of a field is at least the configured value	value	numeric/cObj
maxValue	Checks if the value of a field is less than the configured value	value	numeric/cObj
minLength	Checks if the value of a field has at least the configured length	value	integer/cObj
maxLength	Checks if the value of a field has less than the configured length	value	integer/cObj

Check:	Description:	Parameters:	Datatype:
betweenValue	Checks if the value of a field is between the configured values	minValue maxValue	numeric/cObj numeric/cObj
betweenLength	Checks if the length of the value of a field is between the configured values	minValue maxValue	integer/cObj integer/cObj
minItems	Checks if a field contains at least the configured amount of items (e.g. checkboxes)	value	integer/cObj
maxItems	Checks if a field contains less than the configured amount of items (e.g. checkboxes)	value	integer/cObj
betweenItems	Checks if a field contains values between the configured amount of items (e.g. checkboxes)	minValue maxValue	integer/cObj integer/cObj
isInDBTable	Checks if the value of a field is in a configured field in a configured table	table field additionalWhere	string/cObj string/cObj string/cObj
isNotInDBTable	Checks if the value of a field is not in a configured field in a configured table	table field additionalWhere	string/cObj string/cObj string/cObj
pregMatch	Checks a field value using the configured perl regular expression	value	regExp/cObj
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”	none	-
mathGuard	Checks if a field value equals the result of the generated question of MathGuard	none	-
dateRange	Checks if a field value is between a configured date range	pattern min max	string/cObj string/cObj string/cObj
date	Checks if a field value is a valid date	pattern	string/cObj
time	Checks if a field value is a valid time	pattern	string/cObj
fileAllowedTypes	Checks if the filetype of an uploaded file is allowed	allowedTypes	comma seperated/cObj
fileMaxCount	Checks if the files uploaded from a field are less than the configured value	maxCount	integer/cObj
fileMinCount	Checks if the files uploaded from a field are more than or equal the configured value	minCount	integer/cObj
fileMinSize	Checks if the size of an uploaded file is at least the configured value (in Byte)	minSize	integer/cObj
fileMaxSize	Checks if the size of an uploaded file is at less than the configured value (in Byte)	maxSize	integer/cObj
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_SUBMITTEDOK###	If you use the confirmation view after successful form submission you can enter your template in this subpart.
###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: <input type=”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: <input type”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: <input type”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 emailaddress automatically. Attention: the fieldname has to be written in CAPITAL letters.
###[fieldname]###	GET/POST value of the given inputfield.
###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 inputfield 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 inputfield 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###>e-mail
###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.
###formValuesPrefix###	This marker contains the value of the formValuesPrefix defined in TS.
###PRINT_LINK###	Only usable in ###TEMPLATE_SUBMITTEDOK###. Shows a link to print view.
###CSV_LINK###	Only usable in ###TEMPLATE_SUBMITTEDOK###. Shows a link to export the submitted values as CSV.
###PDF_LINK###	Only usable in ###TEMPLATE_SUBMITTEDOK###. Shows a link to export the submitted values as PDF.
###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
###field_[masterkey]_[fieldname]###	Insert fields predefined in master template. More details in section “How to use a master template”
###HIDDEN_FIELDS###	Formhandler will insert all needed hidden fields here. If you don't add this marker, Formhandler will insert the hidden fields just before the closing <form> tag.

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 conditions and change the settings plugin.Tx_Formhandler.settings.[stepnumber].templateSuffix
8. Save and view the page

How the conditions work:

```
plugin.Tx_Formhandler.settings {
  if {
    1 {
      conditions.OR1.AND1 = field1=value1
      isTrue {
        # Make settings for a different step 3
        3 {
          templateSuffix = _alternative
          validators.1.config.disableErrorCheckFields = firstname,lastname,email
        }
      }
      else {
        # Do something if the condition was not true. This will not be needed often.
      }
    }
    2 {
      conditions.OR1 {
        AND1 = age<21
        AND2 = product=alcohol
      }
      isTrue {
        templateSuffix = _underage
        validators.1.config.disableErrorCheckFields = firstname,lastname,email
      }
      else {
        # Do something if the condition was not true. This will not be needed often.
      }
```

```
    }
}
```

Using the “if” setting you can specifiy conditions for GET/POST parameters. If the conditions evaluates to TRUE, the settings in “isTrue” will be merged with the original Formhandler settings.

The conditions can be specified in two levels. Level 1 will be OR conditions, level 2 will be AND conditions.

Example:

```
plugin.Tx_Formhandler.settings {
  if {
    1 {
      conditions {
        OR1 {
          AND1 = field1=value1
          AND2 = field2=value2
        }
        OR2 {
          AND1 = field3=value3
        }
      }
    }
  }
}
```

This will result in the following condition:

(field1 == 'value1' && field2 == 'value2') || (field3 == 'value3')

You can use the following operators:

=, <, >, !=

NOTE: If you don't use an operator, Formhandler will check if the field is set in GET/POST.

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.

How to 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>
```

or enter a default message for all fields:

```
<label index="is_error_default">Error occurred!</label>
```

A possible TypoScript configuration looks like this:

```
plugin.Tx_Formhandler.settings {
  isErrorMarker {
    global = Global message if an error occurred (filled into ###is_error###)
    default = class=”error” (filled into ###is_error_[fieldname]###
    field1 = TEXT
    field1.value = Some message (filled into ###is_error_field1###)
  }
}
```

Example:

```
###error_firstname###
<div class=”row” ###is_error_firstname###>
  <label for=”firstname”>###LLL:firstname###</label>
  <input type=”text” name=”formhandler[firstname]” value=”###value_firstname”  />
</div>
```

Using this you can display an error message like “Firstname is missing” and add a CSS class to the surrounding DIV to highlight the row containing the error.

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_SUBMITTEDOK### 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 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 use your own component (Interceptor, Finisher, ...)

This is for advanced users only!

There are two ways to integrate your own component.

1. Using an extension:
   - Create a new extension (Example: myext)
   - Create a folder 'Classes' and put your components in there.
     Example:
     myext/Classes/Finisher/Tx_MyExt_Finisher_MyFinisher.php
   - Make sure your class extends the abstract base class (Example: Tx_Formhandler_AbstractFinisher)
   - Use your class in TypoScript.
     Example: finishers.1.class = Tx_MyExt_Finisher_MyFinisher
2. Using additional include paths
   - Create your class somewhere (Example: fileadmin/scripts/Tx_Formhandler_Finisher_MyFinisher.php)
   - Make sure the class name starts with Tx_Formhandler
   - Make sure your class extends the abstract base class (Example: Tx_Formhandler_AbstractFinisher)
   - Enter the path in TypoScript:
     Example: plugin.Tx_Formhandler.settings.additionalIncludePaths.1 = fileadmin/scripts/
   - Use your class in TypoScript.
     Example: finishers.1.class = Tx_MyExt_Finisher_MyFinisher

How to XCLASS Formhandler

Formhandler doesn't use the default pibase architecture. Therefore there is no possibility to change Formhandler's behaviour by Xclassing. But since Formhandler is structured in different classes, you can simply extend an existing class (e.g. Finisher_DB) to satisfy your needs and enter your class name in

Formhandler doesn't use the default pibase architecture. Therefore there is no possibility to change Formhandler's behaviour by Xclassing. But since Formhandler is structured in different classes, you can simply extend an existing class (e.g. Finisher_DB) to satisfy your needs and enter your class name in TypoScript.

TypoScript.

Example:

```
class Finisher_Extended extends Finisher_DB {
      public function process() {
            ...do something else than the default Finisher_DB
      }
}
```

TypoScript:

```
finishers.2 {
  class = Finisher_Extended
  config {
    …
  }
}
```

How to use a master template

Master templates can be used to define a structure for your form fields or email templates. You can add the fields to your template by simply using a marker.

Example:

Master template:

```
<!-- ###field_input### -->
###error_###fieldname######
<label for="###fieldname###">###LLL:###fieldname######</label>
<input type="text" name="###fieldname###" id="###fieldname###" value="###value_###fieldname######" />

<!-- ###field_input### -->

<!-- ###field_submit### -->
###LLL:label_submit###
<input type="submit" ###submit_nextStep### value="Abschicken" />
 <!-- ###field_submit### →
```

As you can see, there are two subparts containing the structure for an input field and the submit button.

Each subpart must begin with either “field_” or “master_” the rest is freely choosable.

In your HTML template you can simply add a marker referring to this subpart plus entering a fieldname as suffix:

```
<!-- ###TEMPLATE_FORM1### -->
<form action="###rel_url###">
      <input type="hidden" name="formhandler[submitted]" value="1"/>
      <input type="hidden" name="id" value="###pid###" />
      ###field_input_name###
      ###field_submit###
</form>
<!-- ###TEMPLATE_FORM1### -->
```

The marker ###field_input_name### refers to the subpart ###field_input###. The additional “_name” is the value to replace the ###fieldname### markers with. If you don't add this suffix, the marker ###fieldname### doesn't get replaced (as in ###field_submit###).

The replaced HTML will look like this:

```
<!-- ###TEMPLATE_FORM1### -->
<form action="###rel_url###">
      <input type="hidden" name="formhandler[submitted]" value="1"/>
      <input type="hidden" name="id" value="###pid###" />
      ###error_name###
```

```
      <label for="name">###LLL:name###</label>
      <input type="text" name="name" id="name" value="###value_name###" />
      ###LLL:label_submit###
      <input type="submit" ###submit_nextStep### value="Abschicken" />
</form>
<!-- ###TEMPLATE_FORM1### -->
```

Using the master template you can predefine HTML for fields and manage it in a single file. You can import the subpart to your HTML template as often as you want.

NOTE: If you use multiple master templates equally named subparts will be overriden by the last master template containing the subpart!

How to use AJAX validation

To validate each field just after the user clicked outside it, you first have to add an AjaxHandler to your TypoScript:

```
plugin.Tx_Formhandler.settings {

  ajax {
    class = Tx_Formhandler_AjaxHandler_JQuery
    config {
      notOk = <span class="notok"> </span>
      ok = <span class="ok"> </span>
      initial = <span class="###is_error_###fieldname######"> </span>
      loading = <img src="fileadmin/templates/images/loader.gif" />
    }
  }
}
```

The handler in this example will use the jQuery framework (www.jquery.com).

As you can see, you can specify the initial content, the content while loading and the content if the check was OK/not OK via TypoScript. This content will be inserted in markers like ###validate_[fieldname]###.

Example:

```
<label for="contact_name">###LLL:label_name######required_name###</label><br />
<input class="input" id="contact_name" type="text" name="contact[name]" value="###value_name###" />
###validate_name###
```

The AJAX validation will use the error checks defined in the “fieldConf” section of Tx_Formhandler_Validator_Default.

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.

TypoScript:

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.

HTML:

Upgrading the HTML from th_mailformplus to Formhandler is easier than you might think.

Formhandler uses nearly the same markers and subparts as th_mailformplus.

Compare these two HTML templates for a simple contact form.

th_mailformplus:

```
<-- ###TEMPLATE_FORM### begin -->
<div class="mailformplus_contactform">
###ERROR###
<form name="contact_form" method="post" action="###REL_URL###">
```

```
      <fieldset>
            <legend>###LLL:legend###</legend>
            <input type="hidden" name="id" value="###PID###" />
            <input type="hidden" name="submitted" value="1" />
            <input type="hidden" name="L" value="###value_L###" />
            <input type="hidden" name="type" value="###value_type###" />
            <div>
                  <label for="name">###LLL:name######required_name###</label>
                  <input type="text" name="name" id="name" value="###value_name###"/>
                  <br/>
                  <label for="subject">###LLL:subject###</label>
                  <input type="text" name="subject" id="subject" value="###value_subject###"/>
                  <br/>
                  <label for="email">###LLL:email######required_name###</label>
                  <input type="text" name="email" id="email" value="###value_email###"/>
                  <br/>
                  <label for="phone">###LLL:phone###</label>
                  <input type="text" name="phone" id="phone" value="###value_phone###"/>
                  <br/>
                  <label for="text">###LLL:text######required_name###</label>
                  <textarea cols="50" rows="5" name="text" id="text"
style="width:320px;">###value_text###</textarea>
                  <br/>
                  <div class="caption">###LLL:contact_via###</div>
                  <input type="radio" name="contact_via" value="email" id="contact_email" style="border-
style:none;" ###checked_contact_via_email### /><label class="radio_caption"
for="contact_email">###LLL:contact_via_email###</label><br />
                  <input type="radio" name="contact_via" value="phone" id="contact_phone" style="border-
style:none;" ###checked_contact_via_phone### /><label class="radio_caption"
for="contact_phone">###LLL:contact_via_phone###</label>
                  <br/>
                  <p>###LLL:required_fields###</p>
                  <input type="submit" value="###LLL:submit###"/>
            </div>
      </fieldset>
</form>
</div>
<!-- ###TEMPLATE_FORM### end -->
```

Now we take this template and make it compatible with Formhandler.

The changes will be:

- Change subpart to ###TEMPLATE_FORM1###
- Add ###submit_nextStep### to submit button

No more changes are required. All other markers are exactly the same in Formhandler.

- ###value_[fieldname]###
- ###error_[fieldname]###
- ###LLL:[key]###
- ###required_[fieldname]###
- ###checked_[fieldname]_[fieldvalue]###
- ###selected_[fieldname]_[fieldvalue]###

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.

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.