FirstSpirit™
Your Content Integration Platform

FirstSpirit FormEdit

FirstSpirit Version 4.0, 4.1 and 4.2

Version	**1.06**
State	**RELEASED**
Date	**2009-08-17**
Department	Professional Services
Author/ Authors	T. Klein
Copyright	2009 e-Spirit AG
File name	FORM40EN_FirstSpirit_Modules_FormEdit

e-Spirit AG

Barcelonaweg 14
44269 Dortmund | Germany

T +49 231 . 286 61-30
F +49 231 . 286 61-59

info@e-spirit.de
www.e-spirit.de

e-Spirit^AG

Table of content

1 Introduction

The "FirstSpirit FormEdit" module consists of an editorial component for the creation of web forms using the FirstSpirit Java or WebClient and a web component in the form of a servlet, which accepts and processes data entered by the user.

1.1 Overview of the functions

The following forms of processing of form data are supported:

- Save the data in a file in CSV format:
  This type of processing saves all values sent by the form within a freely definable file in CSV format.
- Save the data in a JDBC-compatible database:
  By using this type of processing, it is possible to save the values within a database. The configuration setting can be used to define individual mapping for the form fields.
- Dispatch the data as an e-mail:
  With this type of processing, any form data can be sent by e-mail. The e-mail layout can be individually designed by means of an e-mail template. Among other things, the e-mails can be sent with file attachments, and cc and bcc recipients are also possible.
- Output of the data in the log file of the servlet engine:
  This function is used to output all values of the form within the log files of the servlet engine, in which the FormServlet is initialised.
- Calling a URL with parameter passing

  This function enables a URL with the defined parameters to be called, without the user seeing this page in the browser. (e.g. tracking)

Further, it is possible to evaluate the forms via your own implementations. The processing methods named above can also be combined with each other.

1.2 Topics covered in this document

Chapter 1: Provides a brief introduction to the layout and functional scope of the "FirstSpirit FormEdit" module (from page 4).

Chapter 2: Describes installation of the "FirstSpirit FormEdit" module on the server and installation of the web component in a project (from page 7).

Chapter 3: This chapter explains configuration of the "FirstSpirit FormEdit" module (from page 15).

Chapter 4: Describes the creation of the form for the "FirstSpirit FormEdit" module and lists all the available form elements (from page 31).

Chapter 5: This chapter describes use of the exemplary stylesheet file "formedit_css", with which form components can be quickly and easily adjusted. In addition, use and the functions of the "jQuery" framework and its plug-ins are explained, with whose help, for example, the form components used can be checked for content correctness while it is being entered (from page 71).

Chapter 6: This chapter explains the example of the auto completion function supplied with the package. The chapter acts as a concept for developing your own solutions for finding meaningfully completed terms.

Chapter 7: This chapter uses an example to describe the actions an editor must perform to prepare a form for a competition, which not only sends data by e-mail but also stores the data entered in a database (from page 75).

1.3 Layout and function

The following graphic shows the module's layout and how it functions using the example of the live server. The web and application servers used in FirstSpirit are used for use of the module within the preview or staging.

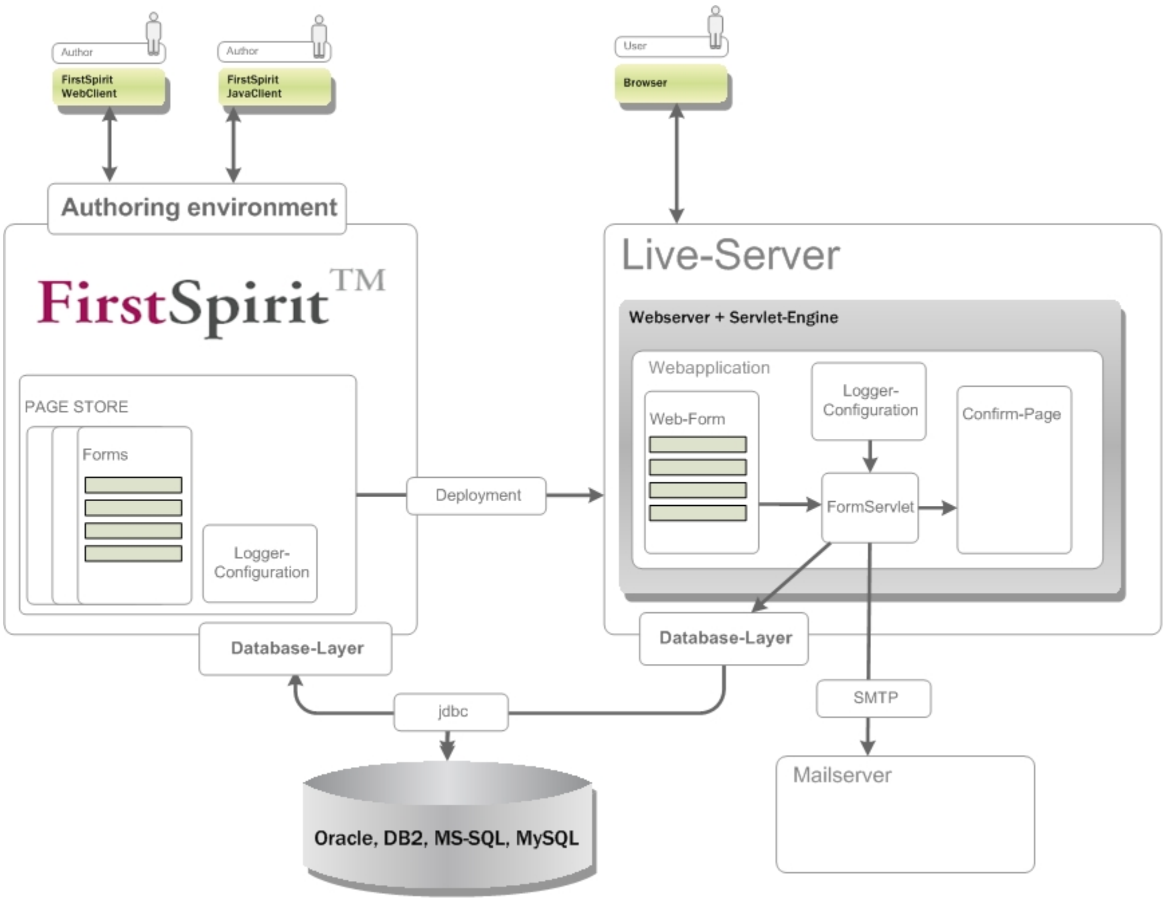


Figure 1-1: Layout and function

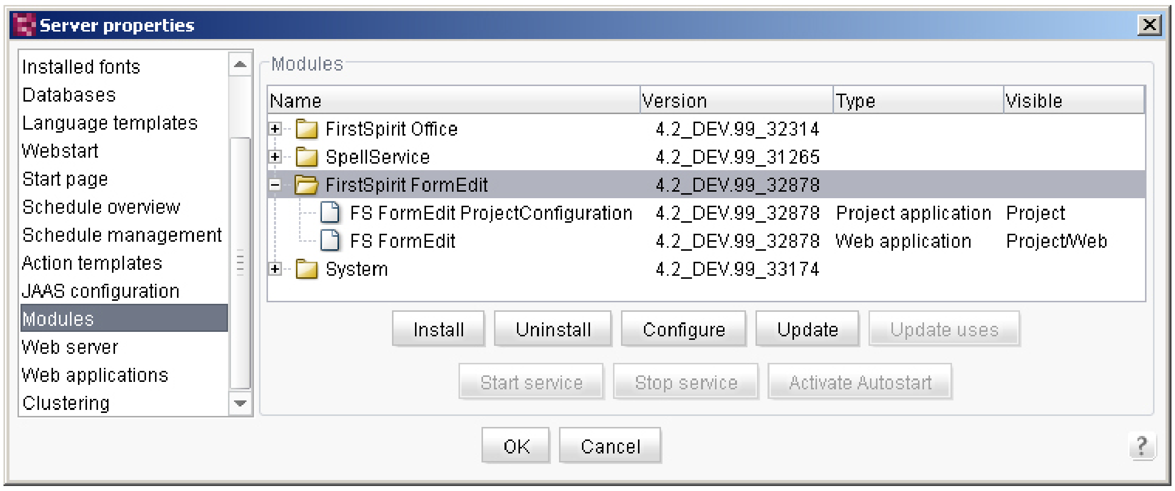
2 Installation and Configuration

"FirstSpirit FormEdit" is installed in five steps:

- Installation of the module: see Chapter 2.1 page 7
- Installation of the project component: see Chapter 2.2.1 page 9
- Installation of the templates supplied with the module: see Chapter 2.2.2 page 10
- Installation of the web component: see Chapter 2.3 page 10
- Configuration of the web component: see Chapter 2.3.1 page 13

2.1 Installing the module on the server

The "FirstSpirit FormEdit module" must first be installed within the server and project configuration application. To this end, the "Modules" menu entry is selected in the "Server Properties" area. Click the "Install" button to open a file selection dialog. The fsm file to be installed can be selected here. The successfully installed module is then displayed in the "Server Properties" dialog:



Name	Version	Type	Visible
FirstSpirit Office	4.2_DEV.99_32314		
SpellService	4.2_DEV.99_31265		
FirstSpirit FormEdit	4.2_DEV.99_32878		
FS FormEdit ProjectConfiguration	4.2_DEV.99_32878	Project application	Project
FS FormEdit	4.2_DEV.99_32878	Web application	ProjectWeb
System	4.2_DEV.99_33174		

Figure 2-1: Installing the module on the FirstSpirit server

The project application "FS FormEdit ProjectConfiguration" and the web application "FS FormEdit" are parts of the "FirstSpirit FormEdit" module.

The **Project Application** provides media, page, section, script and table templates which can be used to design forms. The component is "visible" for the "Project" area. It is therefore a "project locale" component. This can be added following installation of the project component within the required projects (see Chapter 2.2.1 page 9).

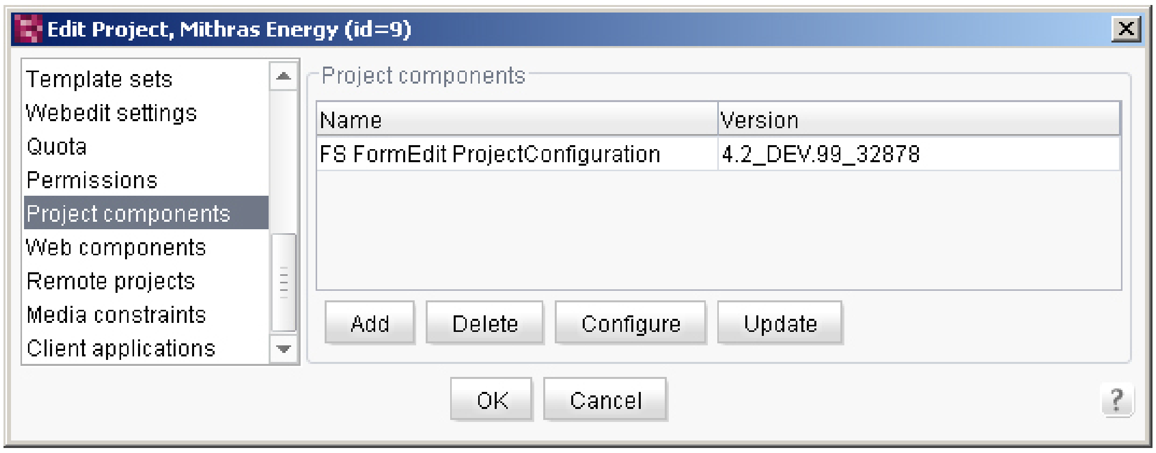
The **web application** provides servlets, which can be used and called within the project. The component is "viewable" for the "Project/Web" areas. It is therefore a "web locale" component. This can be added to the different web areas (previews, staging, live, webedit) within the required projects following installation (see Chapter 2.3, page 10).

For further information on this dialog, see *FirstSpirit Manual for Administrators*.

2.2 Installing the project component

2.2.1 Adding the project component

The project component must now be installed in the required project. To do this, the "Project Components" menu entry within the project properties is opened.



Name	Version
FS FormEdit ProjectConfiguration	4.2_DEV.99_32878

Figure 2-2: Installing the project component

Add: Click the button to open the "Add" dialog. The list shows all project components installed on the server (see Chapter 2.1 page 4). Select the "FS FormEdit ProjectConfiguration" entry.

For further information on this dialog, see *FirstSpirit Manual for Administrators*.

2.2.2 Adding the template to the project

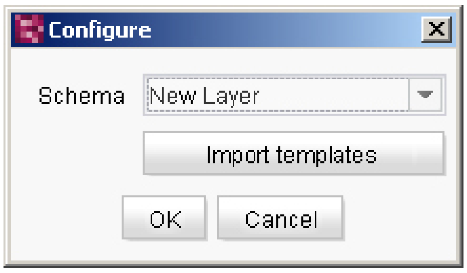


Figure 2-3: Configuring the project component

Configure: Select the "FS FormEdit ProjectConfiguration" entry just added and click the "Configure" button. Select a database layer from the "Schema" combobox and click "Import templates".

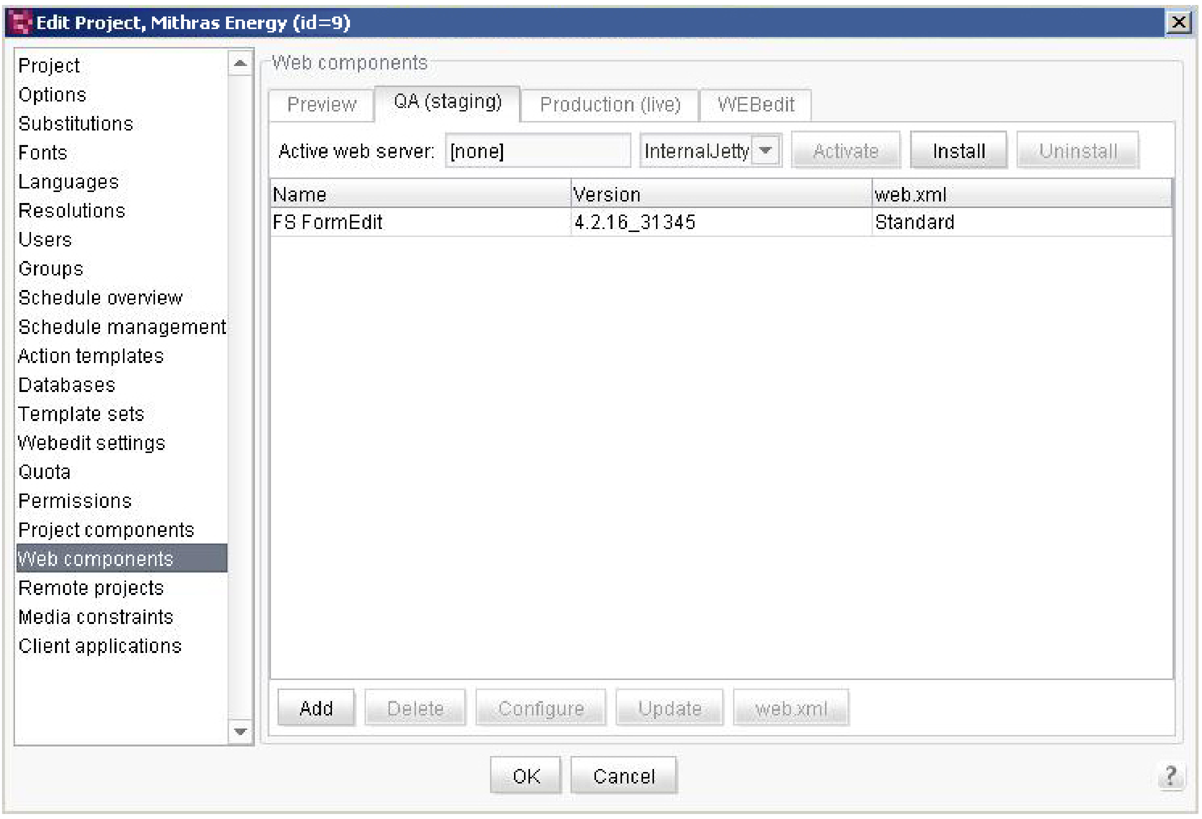
The selection list contains all database layers approved for the project. If you have not used any layers to date, or if you want to use your own database for the module, select "New layer". If this option is selected, a new layer is generated, which points to FirstSpirit's internal Derby database.

For further information on database layers, please refer to the *FirstSpirit Manual for Administrators*.

After you have clicked the "Import templates" button and close the dialog, it is not possible to make any more changes to the configuration. Renewed importing is only possible by means of "Delete" and renewed "Add"-ing of the project component.

2.3 Installing the web application in the project

The web application must now be installed in the required project. To do this, the "Web Components" menu entry is opened within the project properties. The web components for a project can be activated in this area.



Active web server: [none] InternalJetty

Name	Version	web.xml
FS FormEdit	4.2.16_31345	Standard

Figure 2-4: Installing the web application within the web areas

Four different web areas exist for each project. The respective tab can be used to individually enable and configure the web components for each area:

Figure 2-5: Web areas within a project

- Preview: Location for the project content, for which a preview has been requested.
- QA (staging): Location for the generated project content
- Production (live): Location for the deployed project content
- WEBedit: Location for the project content if using a project-locale WEBedit environment

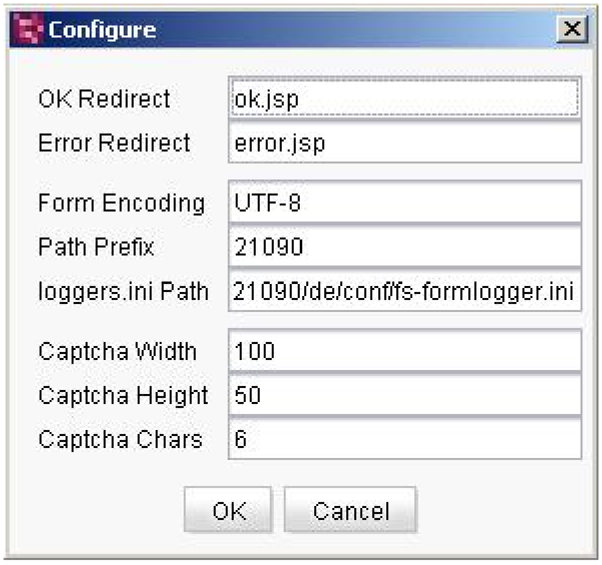
Add: Click the button to open the "Add" dialog. The list displays all web components installed on the server (see Chapter 2.1 page 4).

After adding them to a web area, it is possible to configure the components; either with a web.xml generated by the component or a generic GUI (see Chapter 2.3.1 page 13). Following configuration, the components must then be enabled. A component within a project can be enabled or disabled for specific areas only.

On configuring for production (live), note that there is no automatic deployment of the web applications and their configuration files; instead, the .war file generated using the "Download" button must be manually transferred to the live server.

For further information on this dialog, see *FirstSpirit Manual for Administrators*.

2.3.1 Configuring the web application



OK Redirect	ok.jsp
Error Redirect	error.jsp
Form Encoding	UTF-8
Path Prefix	21090
loggers.ini Path	21090/de/conf/fs-formlogger.ini
Captcha Width	100
Captcha Height	50
Captcha Chars	6

Figure 2-6: Configuring the web application

Different parameters are available for configuring "FirstSpirit FormEdit".

OK Redirect: Use this field to specify the path to the file displayed following successful sending of the form data. This value is used if a special page was not given in the form (see Chapter 4.2.1 page 33). The forwarding behaviour can be influenced using the "redirect:" or "forward:" prefix. "forward:ok.jsp", for example, would generate forwarding with all parameters to the page ok.jsp. If neither "forward:" nor "redirect:" is given, redirect always takes place.

Error Redirect: Use this field to specify the path to the file displayed following incorrect sending of the form data. This value is used if a special page was not given in the form (see Chapter 4.2.1 page 33). The forwarding behaviour can be influenced using the "redirect:" or "forward:" prefix. "forward:error.jsp", for example, would generate forwarding with all parameters to the page error.jsp. If neither "forward:" nor "redirect:" is given, redirect always takes place.

Form Encoding: Use this field to specify the encoding to be used for sending the form data. This is also a retrieval (fall-back)) value, if no encoding was given in the configuration of a processing component (see Chapter 3.2 page 18). Examples are "UTF-8" or "ISO-8859-1".

Always ensure that the encoding chosen matches the encoding used in the generated pages (MetaTags or encoding of the language in the server and project properties).

Path Prefix: Use this field to specify a prefix, which is placed in front of the path to the mail template, in order that it can be used. This prefix describes the partial path between the WebApp root and the folder created by FirstSpirit. For example, for the staging environment, this would be the schedule ID.

Loggers.ini Path: The path to the configuration file `fs-formlogger.ini` must be given in this field. If this field is empty, or if the file cannot be found, an **empty** configuration file is used.

Staging example:
`2708/de/configuration/fs-formlogger.ini`

The schedule ID – here `2708` – is placed in front of the path for the staging environment.

Live example:
`de/configuration/fs-formlogger.ini`

The path to be given here can also be given as an absolute value. This example searches for the file relative to the WebAppRoot.

Captcha Width: Use this field to determine the display width of the Captcha graphic in pixels. If this field is empty, an internal retrieval value of the servlet is used: 100.

Captcha Height: Use this field to determine the display height of the Captcha graphic in pixels. If this field is empty, an internal retrieval value of the servlet is used: 100.

Captcha Chars: Use this field to specify the number of characters displayed in the Captcha graphic. If this field is empty, an internal retrieval value of the servlet is used: 6.

3 Configuration

For this section, it is assumed that the reader is familiar with handling FirstSpirit "Data sources" (content).

For information about handling content, please refer to the *FirstSpirit Manual for Developers* and the *FirstSpirit Manual for Editors (JavaClient)*.

The various process options of the form data are configured in the project by so-called loggers. Each logger is assigned a specific processing type (e.g. MailLogger) and appropriate parameters. The various loggers are maintained as data sets (content store data) in a data source (content). The logger configuration file "fs-formlogger.ini“ is generated on the basis of the logger configuration. The content schema and table templates necessary for this are generated on installation of the project component. To create loggers, it is now only necessary to create content for the table template "form_edit.formLogger“. For information on the logger types and their configuration options, please refer to Chapter 3.2 from page 18.

Please ensure you set mapping for the missing languages within the table template "form_edit.formLogger". (On delivery, only "German" is mapped.) Additional columns with "_<language abbreviation>" should be created for the language-dependent "formLogger_description" column, e.g. "formLogger_description_EN".

3.1 Creating the logger

Open the project in JavaClient. There are now two options for creating or editing the logger:

1. directly via the content (content: FormLogger) in the Content Store

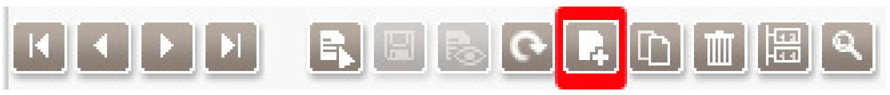


Figure 3-1: New data record (Content)

2. from the "form-start" section (see Chapter 4.2.1 page 33)

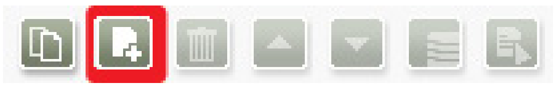


Figure 3-2: New data record (form start)

Clicking the "New Entry" button in the content view or in the input component within the "formstart" section to open the following form:

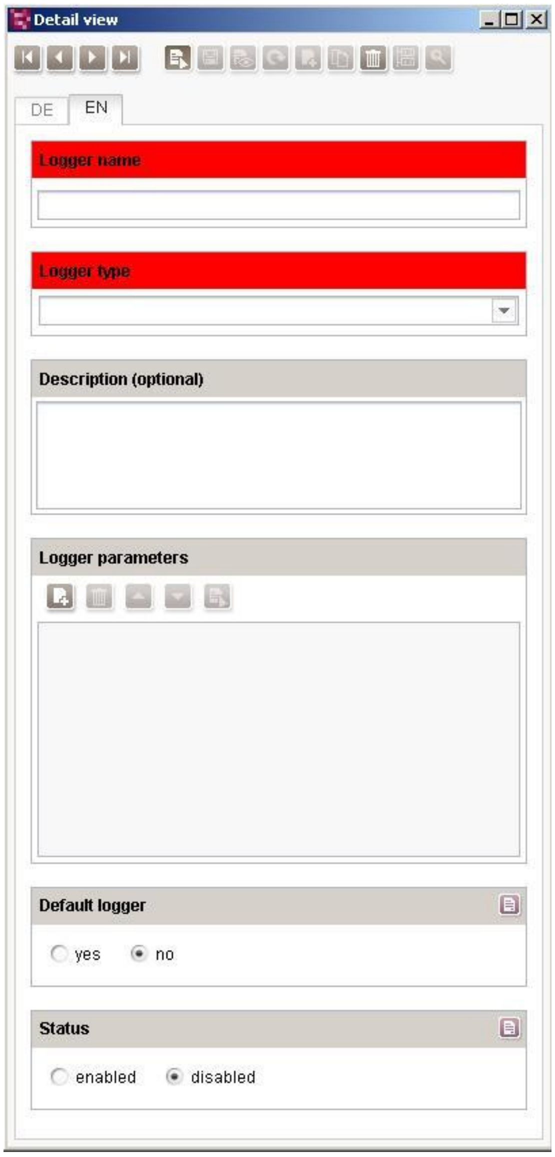


Figure 3-3: Creating the logger

All logger-specific data is entered in this form:

Logger name: Here, the logger can be given a name. The name is used as a reference name within the configuration file. The name may not contain any spaces or special characters/symbols and must be unique throughout the whole project.

Logger type: The logger type is required for creating the logger configuration file "fs-formlogger.ini" (see Chapter 3.5 page 30). The following logger types are available (for a description, see Chapter 3.2 ff., from page 18):

▪ ConsoleLogger	(see Chapter 3.2.1 page 18)
▪ CSVLogger	(see Chapter 3.2.2 page 19)
▪ jdbcLogger	(see Chapter 3.2.3 page 20)
▪ MailLogger	(see Chapter 3.2.4 page 23)
▪ MailUploadLogger	
▪ UrlLogger	(see Chapter 3.2.5 page 24)

Description: Use this field to enter a brief description in this field, so that you can more easily assign the logger at a later date. Input is optional, and has no effect on the function of the logger.
Logger parameters: The logger-specific configuration parameters can be specified here. You can choose between two templates for an entry:

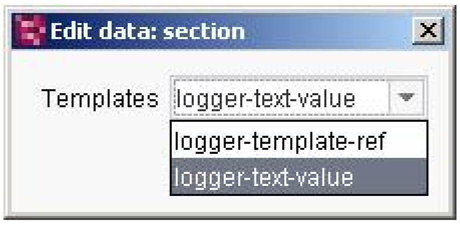


Figure 3-4: Add parameter

logger-text-value (preselected): Apart from one exception – choice of the mail template – this template is used for all parameters.

logger-template-ref: This template is chosen if a mail template is to be selected from the structure for the MailLogger or MailUploadLogger.

The parameters which can be used here are explained in Chapter 3.2, from page 18.

Default logger: These radio buttons can be used to mark a logger as the "default" logger. If one or several default loggers exist, they are also used for forms which cannot be assigned to a logger, e.g. due to a configuration error.

Status: These radio buttons can be used to switch a logger to active (“enabled”) or inactive (“disabled”). Only active loggers can be selected as a processing option in a form.

3.2 Logger configuration of the processing

Each logger is assigned a specific processing type (e.g. MailLogger) and appropriate parameters. The various loggers are maintained as data sets (content store data) in a data source (content). The logger configuration file "fs-formlogger.ini“ is generated on the basis of the logger configuration.

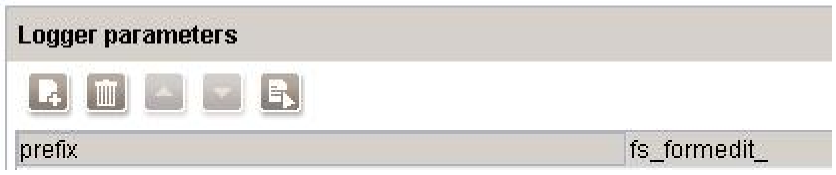
3.2.1 Configuring log file processing

Task:
Output of the form data in the log file of the servlet engine.

Parameters (parameter name, expected value):

`class`	de.espirit.firstspirit.opt.formedit.ConsoleLogger
`prefix`	Text placed in front of the log output

Example:



prefix	fs_formedit_

Figure 3-5: ConsoleLogger 1 parameters: ConsoleLogger parameters

Comments:

`class`	This parameter is generated automatically and does <u>not</u> have to be created. .

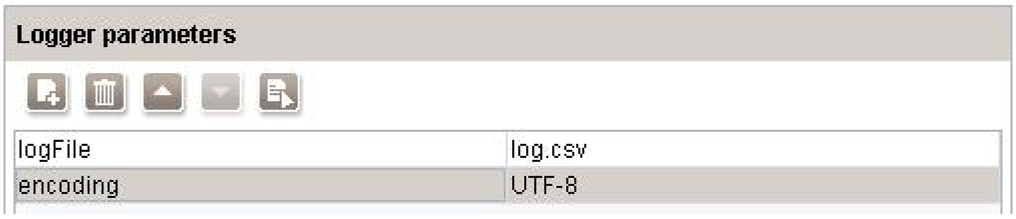
3.2.2 Configuring CSV processing

Task:
Output of the form data in a CSV file.

Parameters (parameter name, expected value):

`class`	de.espirit.firstspirit.opt.formedit.CSVLogger
`logFile`	(Absolute) path to the CSV file
`encoding`	Encoding for sending the e-mail, e.g. "UTF-8"

Example:


Logger parameters

logFile	log.csv
encoding	UTF-8

Figure 3-6: ConsoleLogger 2 parameters: CSVLogger parameters

Comments:

`class`	This parameter is generated automatically and does <u>not</u> have to be created.
`logFile`	The path can be given as an absolute or relative to the web application.
`encoding`	If this parameter is not given, the default value from the configuration of the web application is used (see Chapter 2.3.1 page 13).

3.2.3 Configuring database processing

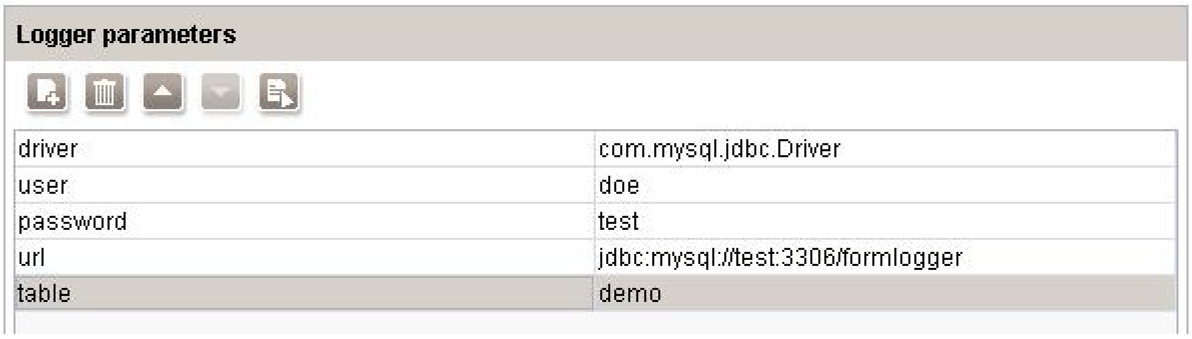
Task:
Output of the form data in a database (interfaced via JDBC).

Unlike the "simple" CSV and log file loggers, the JdbcLogger has several more parameters. The following main aspects can be configured:

JDBC parameters (parameter name, expected value):

`class`	de.espirit.firstspirit.opt.formedit.JdbcLogger
`driver`	JDBC driver, e.g. "org.gjt.mm.mysql.Driver"
`user`	Database user, e.g. "cms"
`password`	Password of the database user
`url`	JDBC-URL to the database, e.g. "jdbc:mysql://localhost:3306/logging"
`table`	Name of the table in the database in which the logging is to take place.

Example:



driver	com.mysql.jdbc.Driver
user	doe
password	test
url	jdbc:mysql://test:3306/formlogger
table	demo

Figure 3-7: JdbcLogger parameters

Comments:

`class`	This parameter is generated automatically and does <u>not</u> have to be created.

Mapping rules:
It is possible to define into which table column each form parameter is to be added. If the parameters are not explicitly assigned, the software tries to use the parameter name as the column name. If this also fails, an entry is only made in the

"unmappedColumn" (see below, Item Additional parameters).

The following schema applies here:

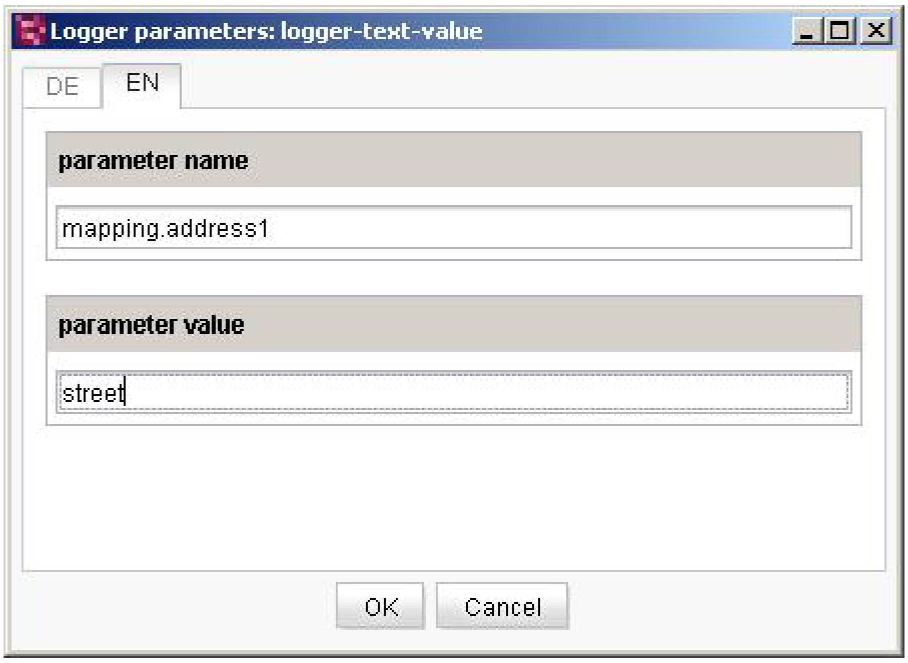
formparameter

Unique identifier of the form element

columnName

Name of the column in which the value is to be written

Example:



parameter name: mapping.address1

parameter value: street

Figure 3-8: JdbcLogger Mapping parameters

The value of the form element "address1" is to be written in the database, in the "street" field:

Additional parameters (Parameter name, expected value):
The following special rules can be specified in addition to the mapping rules:

`csvColum`	Name of the table column in which the complete form data record is entered in CSV form.
`unmappedColumn`	Name of the table column in which all form data in CSV form NOT processed by mapping rules is entered.

`timestampColumn` Name of the table column in which the date and time at which the request is received are saved in timestamp format.

3.2.4 Configuring e-mail processing

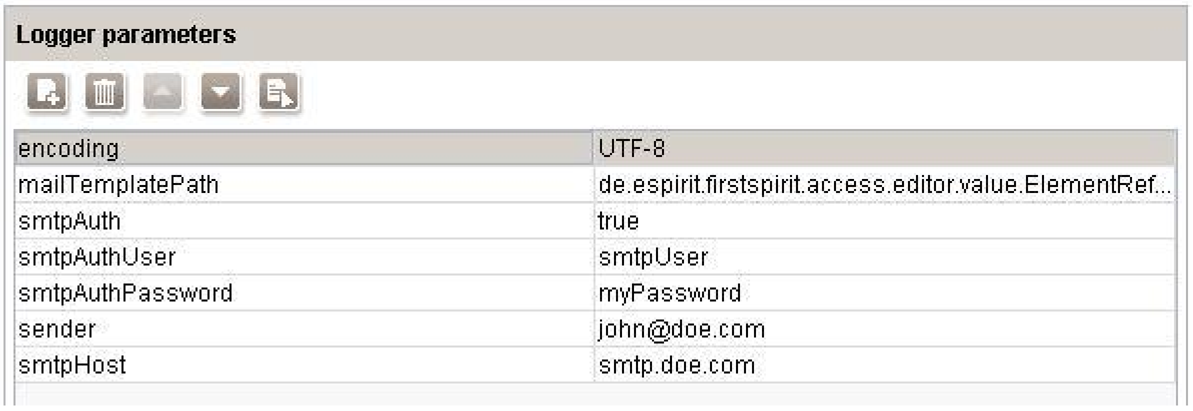
Task:
Output of the form data in the form of an (configurable by means of a file) e-mail (optionally with file attachment too).

The e-mail logger is used to send the form data by e-mail. A separate e-mail is sent for each form. The format and/or text of the e-mail can be configured in a (separate) file. Due to the form-specific logger configuration, if necessary, an e-mail configuration file can be assigned to each form.

Parameters:

`class`	de.espirit.firstspirit.opt.formedit.JdbcLogger de.espirit.firstspirit.opt.formedit.MailUploadLogger
`smtpHost`	Name of the e-mail server
`sender`	E-mail address of the sender
`mailTemplatePath`	(absolute) path to the e-mail configuration file
`encoding`	Encoding for sending the e-mail
`smtpAuth`	(Optional) "true," if the smtp server requires authentication (smtpAuth)
`smtpAuthUser`	Name of the user for the authentication
`smtpAuthPassword`	Password for the authentication

Example:


Logger parameters

encoding	UTF-8
mailTemplatePath	de.espirit.firstspirit.access.editor.value.ElementRef...
smtpAuth	true
smtpAuthUser	smtpUser
smtpAuthPassword	myPassword
sender	john@doe.com
smtpHost	smtp.doe.com

Figure 3-9: MailLogger parameters

Comments:

`class`	This parameter is generated automatically and does <u>not</u> have to be created.

`mailTemplatePath`	"logger-template-ref" should be selected here as the template. This can be used to select the template from the structure.
`encoding`	If this parameter is not given, the default value from the configuration of the web application is used (see Chapter 2.3.1 page 13).
`smtpAuthUser /`	If the "smtpAuth" parameter is set with the value "true",
`smtpAuthPassword`	these parameters are mandatory parameters and must be given. "logger-text-password" can be selected here as the template, to display the data concealed.

3.2.5 Configuring URL processing

Task:
URL call with parameter passed to another page

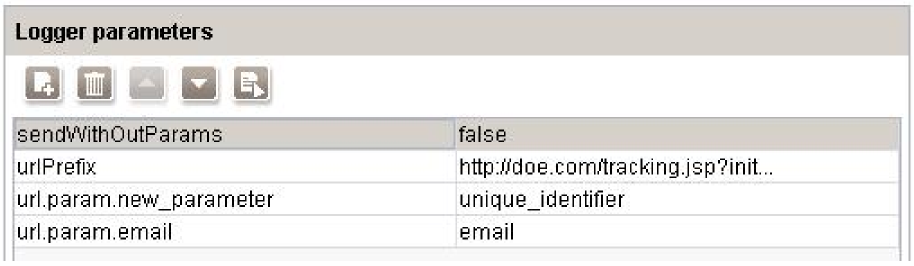
The URL logger is used to call another page (URL) with the option of passing on defined parameters of the form. However, the user does not call this page in their browner. A classic application case is, for example, tracking.

Parameters:

`class`	de.espirit.firstspirit.opt.formedit.URLLogger
`url.sendWithOutParams`	Pass parameters (true / false)
`url.urlPrefix`	Destination URL (e.g. http://myserver.com/tracking.jsp)
`url.param.<identifier>`	Unique identifier of the form component

! *Similar to the JdbcLogger, it is necessary to map the form's parameters to new parameters. In the case of URLLogger, only the parameters specified in the configuration are attached to the UrlPrefix.*

Example:



sendWithOutParams	false
urlPrefix	http://doe.com/tracking.jsp?init...
url.param.new_parameter	unique_identifier
url.param.email	email

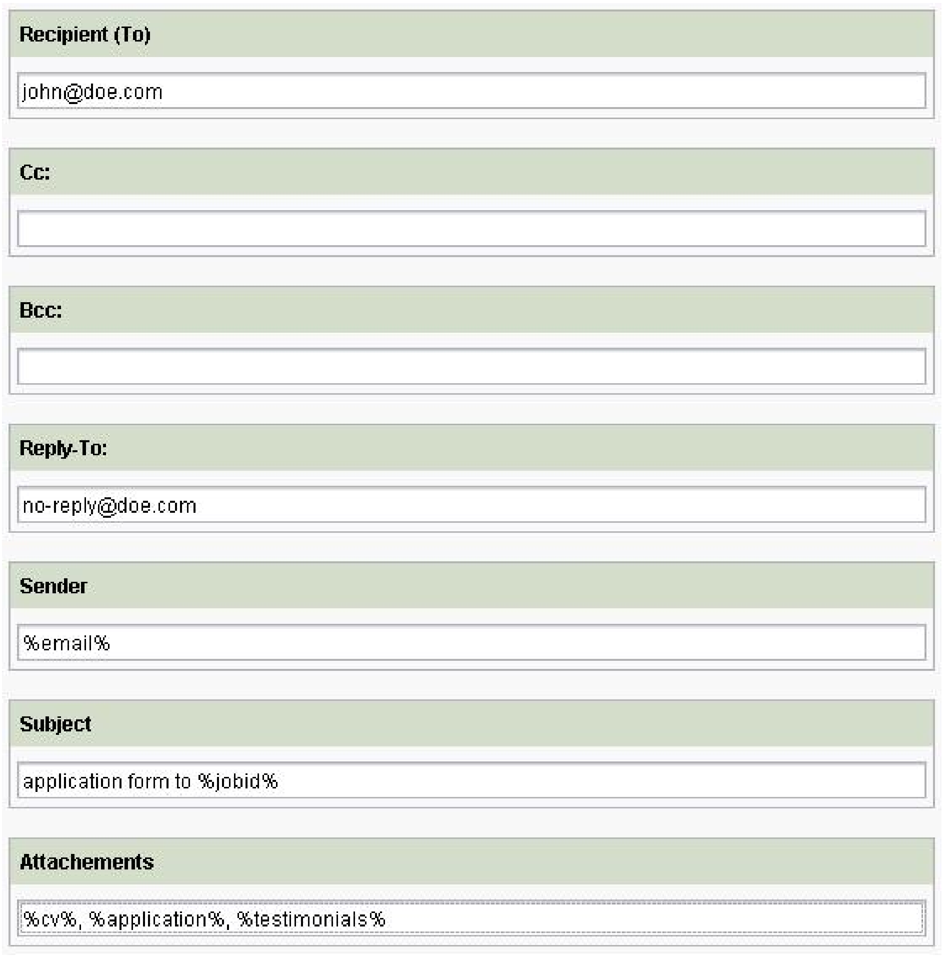
Figure 3-10: URLLogger parameters

Comments:

`class`	This parameter is generated automatically and does not have to be created.
`url.param.<param>`	`<param>` Defines the parameter name, in the way it is to be attached to the URL.

3.3 E-mail configuration file

Task:
The e-mail to be sent is configured using the page template "mailtemplate". The interface is similar to that of an e-mail program.



Recipient (To)	john@doe.com
Cc:	
Bcc:	
Reply-To:	no-reply@doe.com
Sender	%email%
Subject	application form to %jobid%
Attachements	%cv%, %application%, %testimonials%

Figure 3-11: Mail template (header)

Recipient (To), Cc, Bcc: One or several recipients' e-mail addresses can be given here. If using several addresses, they must be separated with a “;“ (semi-colon).

Reply to: An e-mail address for a reply can be entered here. This is used if the user clicks "Reply" in their e-mail program.

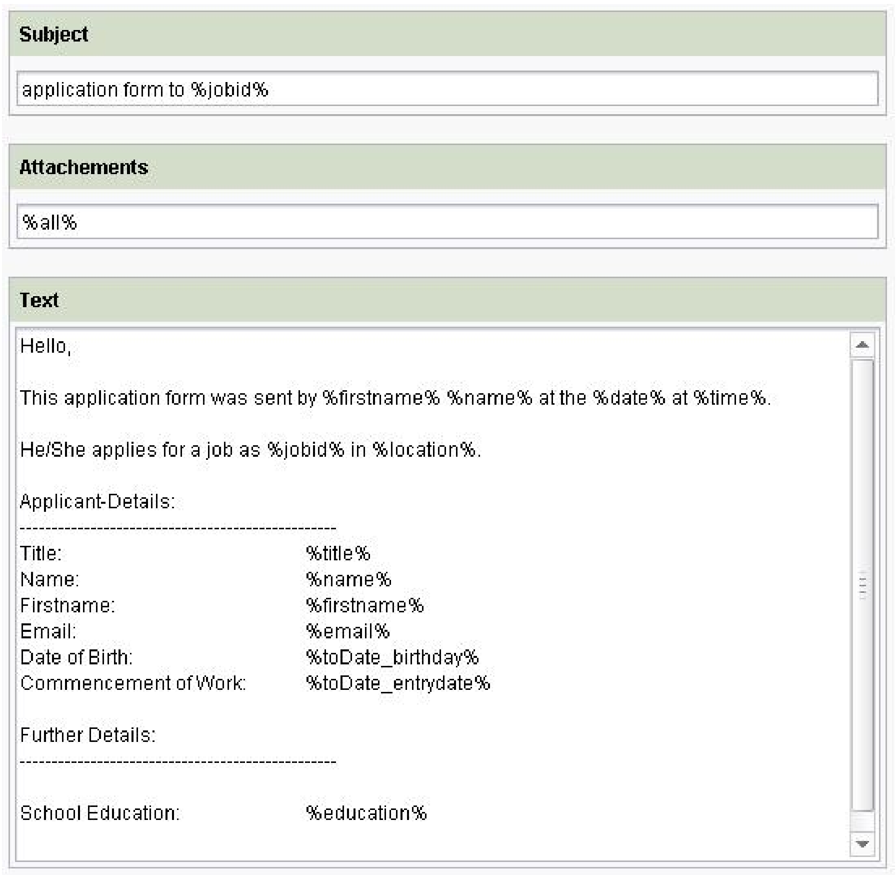
Sender: An e-mail address can be defined here, which is displayed as the sender. Alternatively, it is also possible to use %parameter% (here: %e-mail%) to access a form element which contains a valid e-mail address. If a value is given here, the "sender" value given in the logger configuration is overwritten.

Subject: The subject of the e-mail to be sent can be given here.

Attachments: Here you can configure the file attachments. This can be done on the one hand, using %parameter% or using %all%.

%parameter%	If this parameter is given, only the file passed via the form element with the identifier "parameter" is attached to the e-mail. Several files must be separated by commas (,). Example: %lebenslauf%,%foto%
%all%	With this input, all files sent with the form are attached to the e-mail.

Text: After the information required for the e-mail header has been give, the e-mail text is entered This text can contain wildcards in the form %name%, which are used to access values from the form.



Subject

application form to %jobid%

Attachements

%all%

Text

```
Hello,

This application form was sent by %firstname% %name% at the %date% at %time%.

He/She applies for a job as %jobid% in %location%.

Applicant-Details:
-------------------------------------------------
Title:                          %title%
Name:                           %name%
Firstname:                      %firstname%
Email:                          %email%
Date of Birth:                  %toDate_birthday%
Commencement of Work:           %toDate_entrydate%

Further Details:
-------------------------------------------------

School Education:               %education%
```

Figure 3-12: Mail template (message part)

The following parameters are available:

%date%	Date on which the form is sent
%time%	Time at which the form is sent
%csv%	List of all form parameters as CVS
%unmapped%	List of all form unedited parameters as CSV (see below)

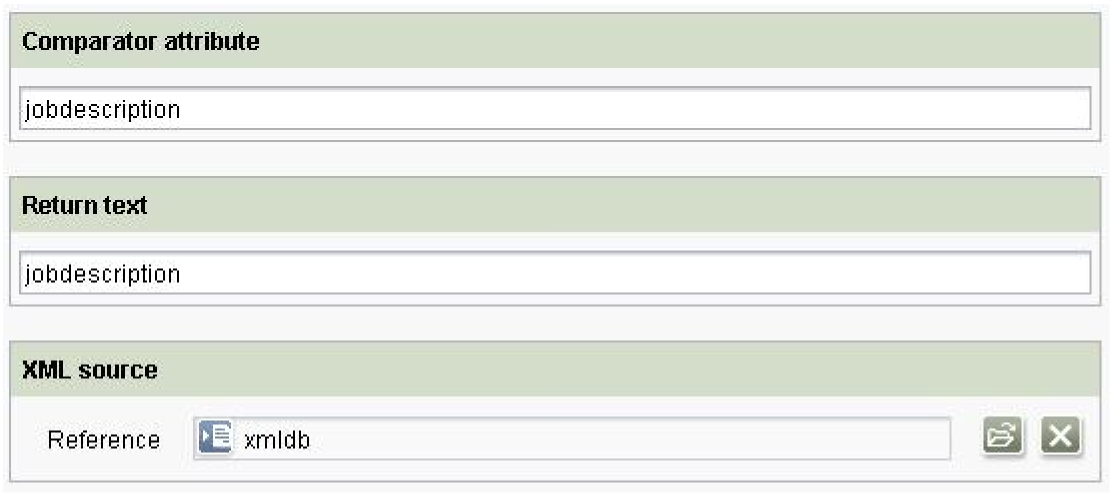
In addition to these parameters, each form parameter can be used. This is done using the % notation and the parameter name: %parameter%

Form data which has been sent, but was not output in the mail template via % notation, can be output using the parameter %unmapped%.

3.4 Autocomplete request

This page template is a functional **example template**. It is also possible to check data against a database or similar. This file is **not generally valid and must be adjusted for each specific project**, if the source is not an XML file. For further information please refer to Chapter 6 page 73.

Task:
This page template is used to process an input source such as XML. If the user makes an entry in an autocomplete form field, the source given on this page is browsed through and the results are shown to the user.



Comparator attribute

jobdescription

Return text

jobdescription

XML source

Reference xmldb

Figure 3-13: Autocomplete request

Comparator attribute: The attribute within the XML source file which is to be compared with the user's input is given here.

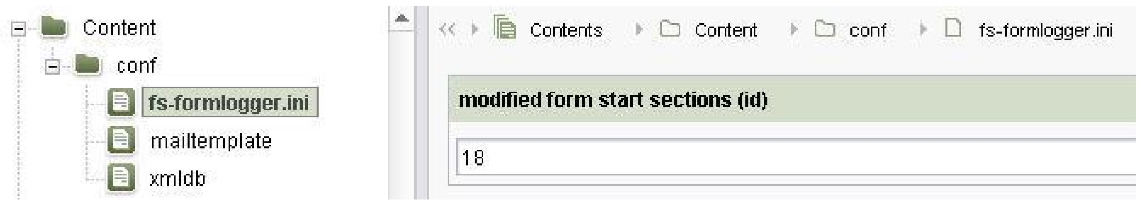
Return text: Here you can define which attribute the XML source file is to show the user as the return value.

XML source: Here you select the XML source file which is to provide the results.

3.5 "fs-formlogger.ini" configuration file

This configuration file contains all the information of your forms or assignment to the processing configurations and these configurations themselves. The content of this file is generically generated by FirstSpirit during generation.

In order for this file to be correctly generated, a page based on the "logger-ini-file" template must be created in the Page Store, and the object ID of the "form start" template, or the templates generated by you which fulfil the function of the "form start" template, must be entered on this page. The ID is displayed with the keyboard shortcut "ALT + P" in the selected section template



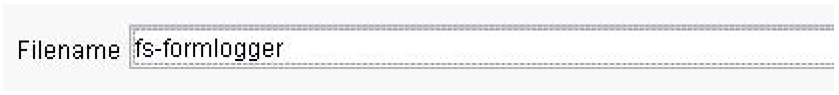
modified form start sections (id)

18

Figure 3-14: "fs-formlogger.ini" page

When referencing in the structure, note that the file is filed in the place given when configuring the web component (see Chapter 2.3.1 page 13).

Further, the file name of the page must be correctly set in the Site Store. The name "fs-formlogger" is used within this document:



Filename fs-formlogger

Figure 3-15: File name of the configuration file

4 Creating Forms

4.1 Form layout

A specific section template order must be adhered to when creating the form to ensure the form works. The section template **form-start** (Chapter 4.2.1 page 33) always marks the start and the section template **form-end** (Chapter 4.2.4 page 38) always marks the end of a form. Any number of sections of the type **form-block** (Chapter 4.2.2 page 37) and **form-divider** (Chapter 4.2.3 page 37) can occur between these two sections.

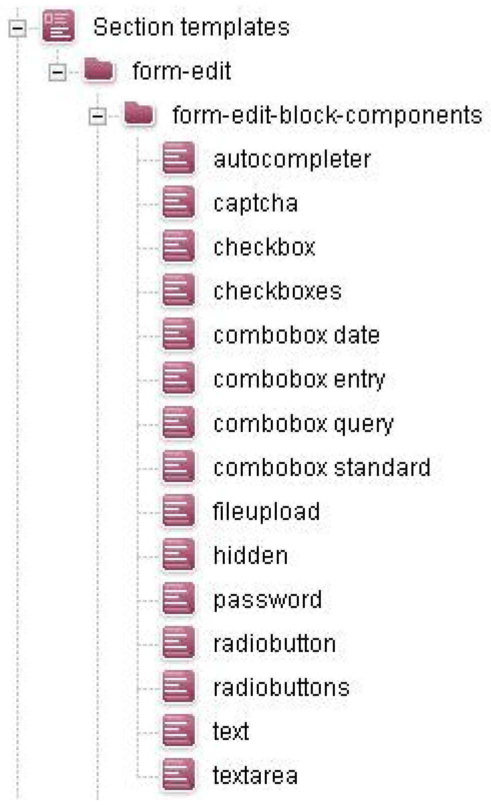

Section templates
form-edit
form-edit-block-components
autocompleter
captcha
checkbox
checkboxes
combobox date
combobox entry
combobox query
combobox standard
fileupload
hidden
password
radiobutton
radiobuttons
text
textarea

Figure 4-1: Form templates

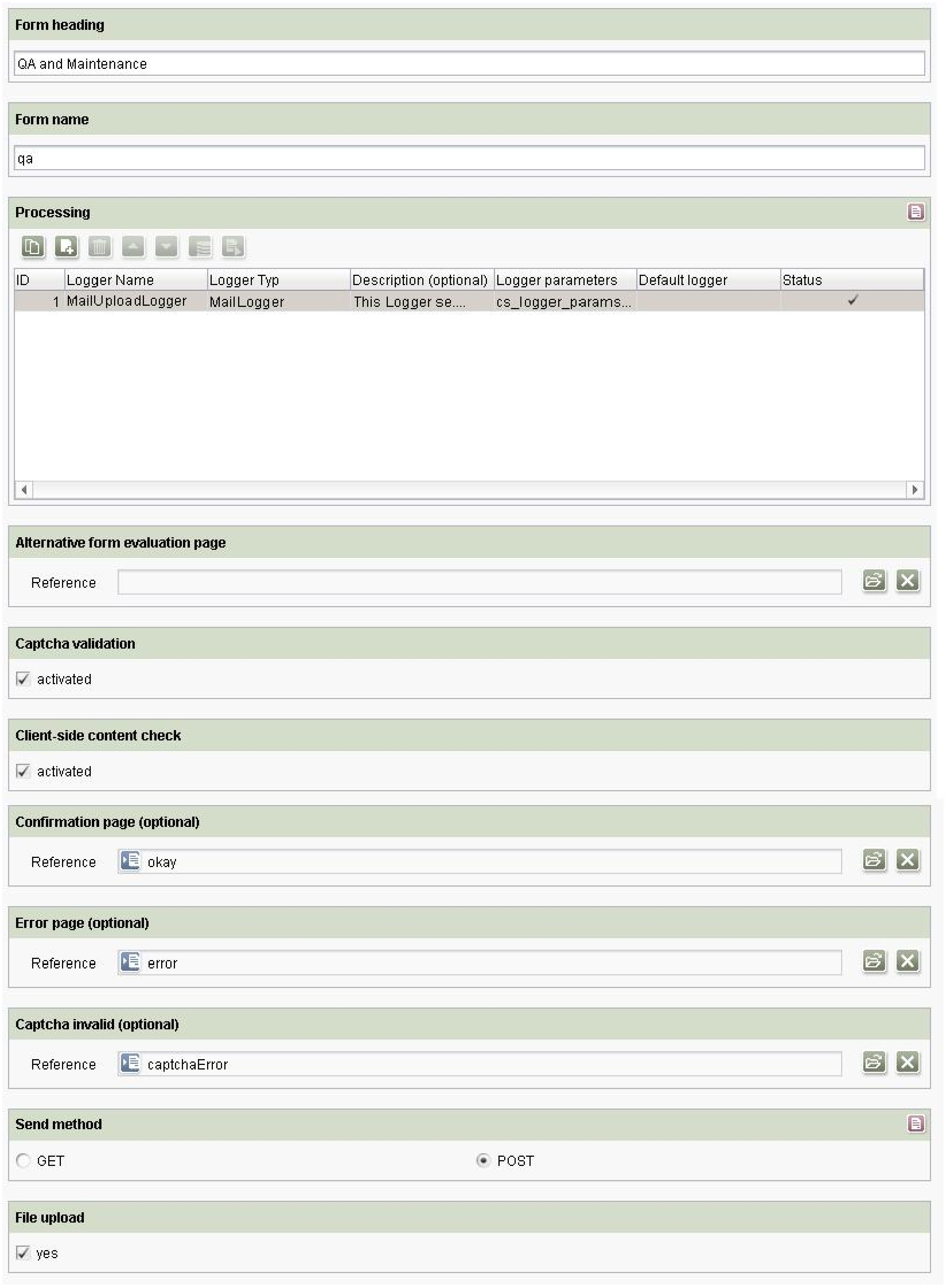
A **form-block** element can contain any number of form elements.

The section templates of the form editor provide all the form elements available in HTML and also provide sufficient options for configuring the components. It is necessary to adjust to specific design requirements first before using the components. On the one hand, this can be done by direct adjustment of the HTML code in the section templates and / or by defining cascading style sheets in the integrated stylesheet file. Each form component can be individually assigned a stylesheet class.

4.2 Form configuration

4.2.1 form-start

The section template "form-start" introduces a new form. The basic configurations, which solely concern this form, can be made here.



Form heading

QA and Maintenance

Form name

qa

Processing

ID	Logger Name	Logger Typ	Description (optional)	Logger parameters	Default logger	Status
1	MailUploadLogger	MailLogger	This Logger se....	cs_logger_params...		✓

Alternative form evaluation page

Reference

Captcha validation

☑ activated

Client-side content check

☑ activated

Confirmation page (optional)

Reference okay

Error page (optional)

Reference error

Captcha invalid (optional)

Reference captchaError

Send method

○ GET ● POST

File upload

☑ yes

Figure 4-2: form-start section

Form heading (text): Display of the form's heading

Form name (text): Unique identifier for the form ("name" attribute of the "form" element)

This identifier may not contain any spaces or special characters/symbols, as the form name is used as part of the servlet call.

Processing (ContentList): Selection and display of the loggers for this form

Alternative form evaluation page (page reference): If a logger is not wanted, a form evaluation page can be given here as an optional alternative.

Captcha validation (checkbox): Activation of server-side captcha validation. If this checkbox is enabled, the form must contain the captcha form element.

As a default, the servlet is notified of the status by means of a concealed form field (input type="hidden"). This can be recognised by senders of spam. Instead of

<input type="hidden" name="useCaptcha" value="true" />

we recommend using the following jsp code in the template:

<% session.setAttribute("useCaptcha","true"); %>

Client-side content check (checkbox): Activation of client-side content checking

Confirmation page (optional) (page reference): This form-specific confirmation page is displayed if the e-mail has been successfully sent. If a page is selected here, the global configuration in the web.xml is overwritten. The type of forwarding can be defined in the template by a "forward:" or "redirect:" placed in front of the reference. If no prefix is given, redirection is to the destination page.

```
<input type="hidden" name="okRedirect"
value="forward:$CMS_REF(st_noerrorPage)$" />
<input type="hidden" name="errorRedirect"
value="redirect:$CMS_REF(st_errorPage)$" />
```

As a default, no prefix is set in the template.

Error page (optional) (page reference): This form-specific confirmation page is displayed if sending the e-mail was unsuccessful. If a page is selected here, the global configuration in the web.xml is overwritten. The type of forwarding can be defined in the template by a "forward:" or "redirect:" placed in front of the reference. If no prefix is given, redirection is to the destination page.

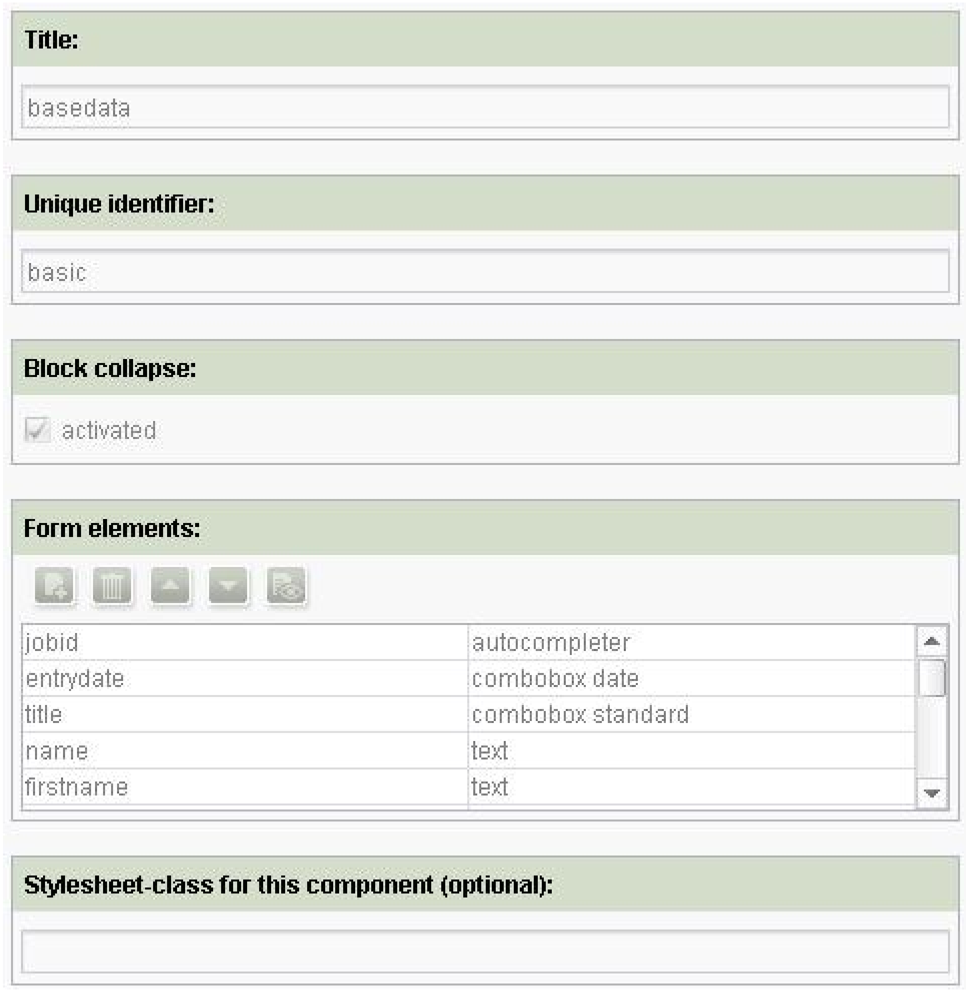
Captcha invalid (optional) (page reference): This form-specific page is displayed if the user has not made any or has made an incorrect input. If a page is not selected here, the "error page" is displayed instead (see above).

Send method (RadioButton): Use this component to specify the transmission mode for the form values. Default selection: "POST“

File upload (checkbox): If this option is set, files can be passed to the web server via the form.

4.2.2 form-block

Any number of form elements can be created within a "form-block" element. The order of the components is irrelevant for the form's ability to function. Each component can be individually configured. For example, stylesheet classes can be used and the display width and height of the form defined. For precise information and configuration examples, please refer to Chapter 4.3, from page 39.



Title:
basedata

Unique identifier:
basic

Block collapse:
✓ activated

Form elements:

jobid	autocompleter
entrydate	combobox date
title	combobox standard
name	text
firstname	text

Stylesheet-class for this component (optional):

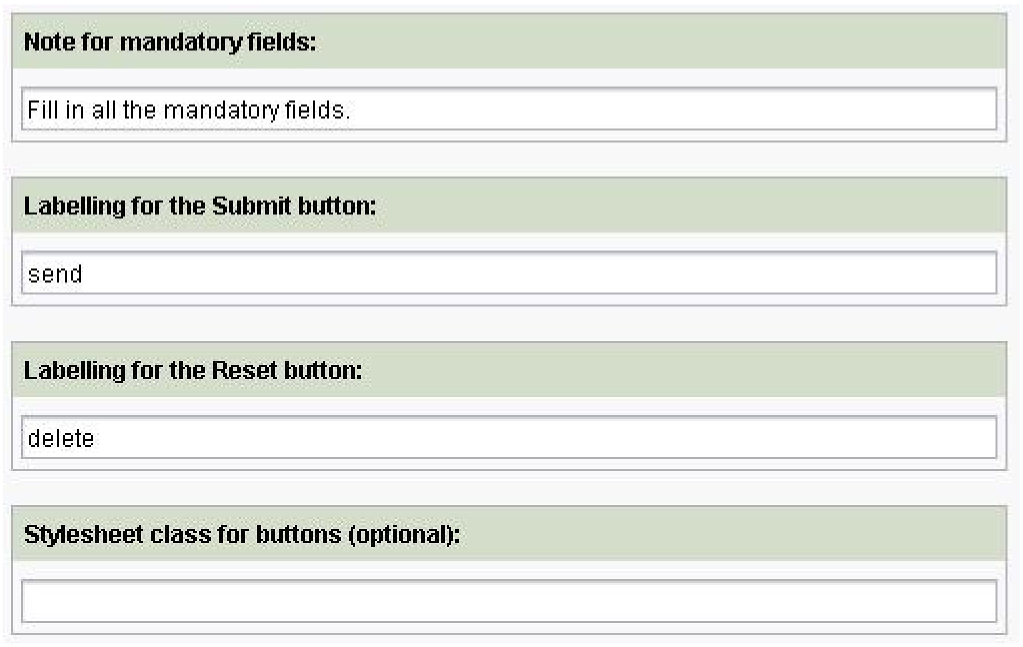
Figure 4-3: form-block section

4.2.3 form-divider

The "form-divider" element generates a graphic separation within the form and otherwise has no function.

4.2.4 form-end

The "form-end" section template defines the end of the form.



Note for mandatory fields:	Fill in all the mandatory fields.
Labelling for the Submit button:	send
Labelling for the Reset button:	delete
Stylesheet class for buttons (optional):	

Figure 4-4: form-end section

Note for mandatory fields: A note for mandatory fields can be given here; it is displayed below the form fields.

Labelling for the Submit/Reset button: Enter the identifier (name) for the button, with which the form data is sent or all the form's entries are deleted. If no entries are made, a button is not generated.

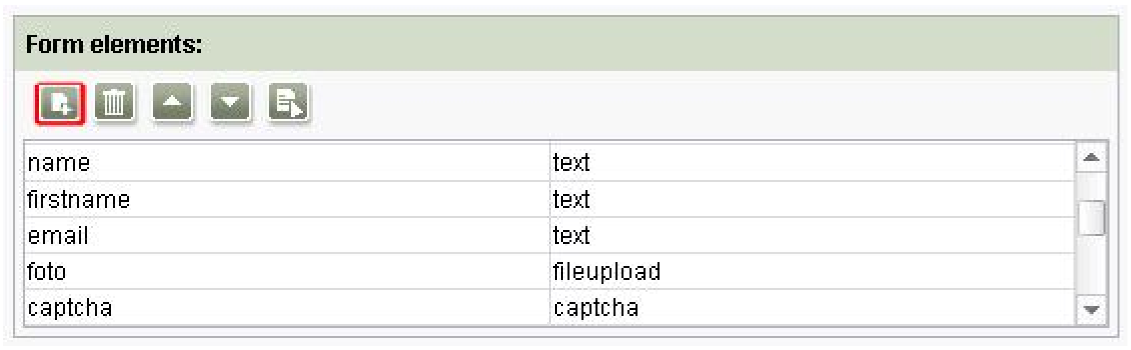
Stylesheet class for buttons (optional): Here you can give the name of a stylesheet class for the design of the buttons. If a class is not given, the buttons are displayed in the standard look-and-feel of the browser or your stylesheet file.

4.3 Available form elements

The form editor enables the editor to use the complete set of HTML form elements. As already mentioned, before using the components it is advisable to make an adjustment to the HTML source code together with the stylesheet file. In this way, individual design templates can be adhered to and are available on the whole of the web page. However, the HTML tags of the form elements should not be affected by the changes, to ensure correct function of the JavaScript checks.

The form components can only be used in the "form-block" section template!

To add a new form element, select an already created section of the type "form-block" and add a new form element to the content area list:


Form elements:

name	text
firstname	text
email	text
foto	fileupload
captcha	captcha

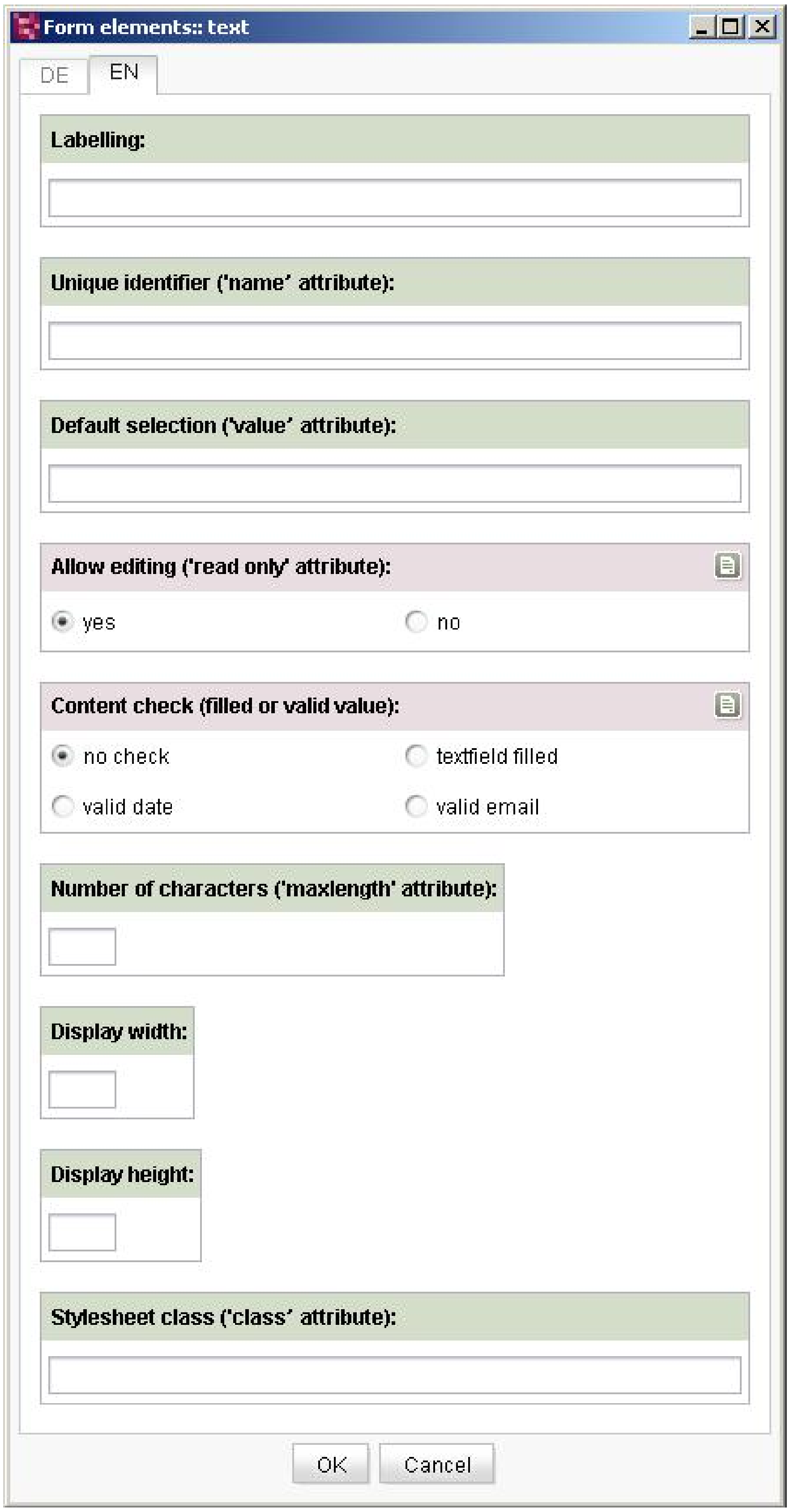
Figure 4-5: Form elements

The following gives an overview of the available standard elements and their configuration options and functions.

The content of the form field can be accessed at a later date via the unique identifier or unique group identifier. For example, if a mail / MailUploadLogger is used, this takes place via %uniqueIdentifier%

4.3.1 Form component "text"

This component provides the user with a form text field. The component must be assigned a designator which is unique in this form ("name" attribute), so that the evaluation of the form and check of its content can work properly.


Form elements:: text

DE EN

Labelling:

Unique identifier ('name' attribute):

Default selection ('value' attribute):

Allow editing ('read only' attribute):

[✓ yes] no

Content check (filled or valid value):

[✓ no check] textfield filled

valid date valid email

Number of characters ('maxlength' attribute):

Display width:

Display height:

Stylesheet class ('class' attribute):

OK Cancel

Figure 4-6: Form component "text"

Labelling: Use this text field to enter the title of the form field.

Unique identifier ('name' attribute): Use this text field to enter the unique identifier.

Default selection ('value' attribute): Use this text field to enter text with which the form field is filled.

Allow editing ('read only' attribute): This radio button is used to control whether the user may edit the form field or not.

Content check (filled or valid value): This radio button can be used to create a check which checks for content or valid data.

Number of characters ('maxlength' attribute): This text field can be used to define the maximum number of characters that can be entered.

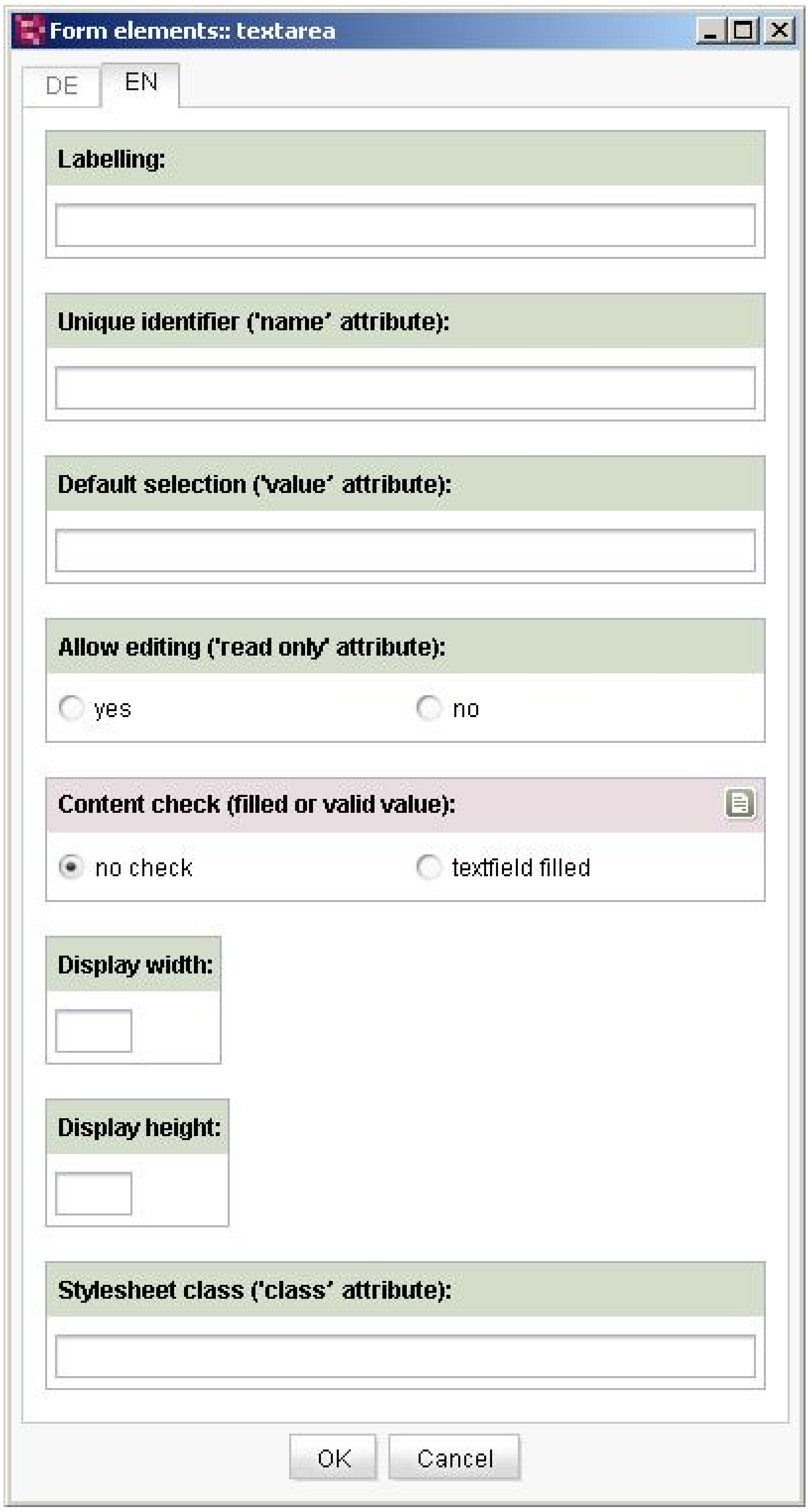
Display width: Use this field to specify the width of the form field in pixels.

Display height: Use this field to specify the height of the form field in pixels.

Stylesheet class ('class' attribute): Use this text field to enter the name of a CSS class.

4.3.2 Form component "textarea"

The "Textarea" form component provides a multi-line text input field for the user. The component must be assigned a designator which is unique in this form ("name" attribute), so that the evaluation of the form and check of its content can work properly.


Form elements:: textarea

DE EN

Labelling:

Unique identifier ('name' attribute):

Default selection ('value' attribute):

Allow editing ('read only' attribute):

yes no

Content check (filled or valid value):

[✓ no check] textfield filled

Display width:

Display height:

Stylesheet class ('class' attribute):

OK Cancel

Figure 4-7: Form component "textarea"

Labelling: Use this text field to enter the title of the form field.

Unique identifier ('name' attribute): Use this text field to enter the unique identifier.

Default selection ('value' attribute): Use this text field to enter text with which the form field is filled.

Allow editing ('read only' attribute): This radio button is used to control whether the user may edit the form field or not.

Content check (filled or valid value): This radio button can be used to create a check which checks for content or valid data.

Display width: Use this field to specify the width of the form field in pixels.

Display height: Use this field to specify the height of the form field in pixels.

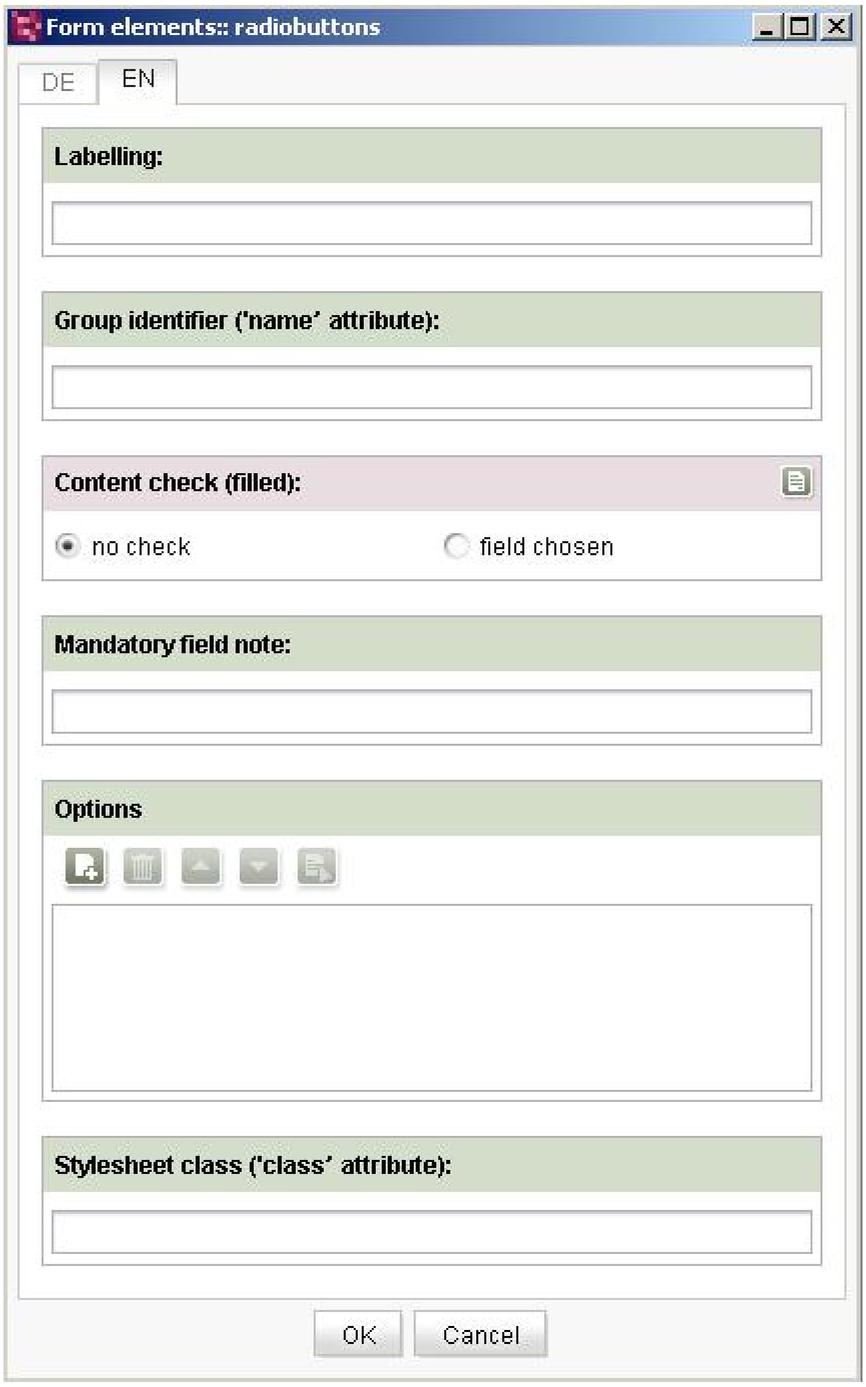
Stylesheet class ('class' attribute): Use this text field to enter the name of a CSS class.

4.3.3 Form component "RadioButtons"

This form component can be used to generate HTML radio buttons. On adding the radio buttons in the "Options" area, ensure that a meaningful default is given for each radio button. The radio buttons are shown with their corresponding designation (labelling), one after the other on the right-hand side of the component.

4.3.3.1 "RadioButtons" subcomponent

This component can be added within a "form-block" section. The values which are the same for all radio buttons are set in this section. The actual RadioButtons are added within a ContentAreaList in this section.


Form elements:: radiobuttons

DE EN

Labelling:

Group identifier ('name' attribute):

Content check (filled):

[● no check] ○ field chosen

Mandatory field note:

Options

Stylesheet class ('class' attribute):

OK Cancel

Figure 4-8: Form component "RadioButtons"

Labelling: Use this text field to enter the title of the form field.

Group identifier ('name' attribute): Use this text field to enter the unique identifier.

Content check (filled): This radio button can be used to create a check which checks whether an entry has been selected.

Mandatory field note: A message which appears if the content check fails must be given for this type of form field.

Stylesheet class ('class' attribute): Use this text field to enter the name of a CSS class.

4.3.3.2 "RadioButton" subcomponent

This component can only be used within the form "RadioButtons" component. It is used to add the actual radio buttons within the component named above.

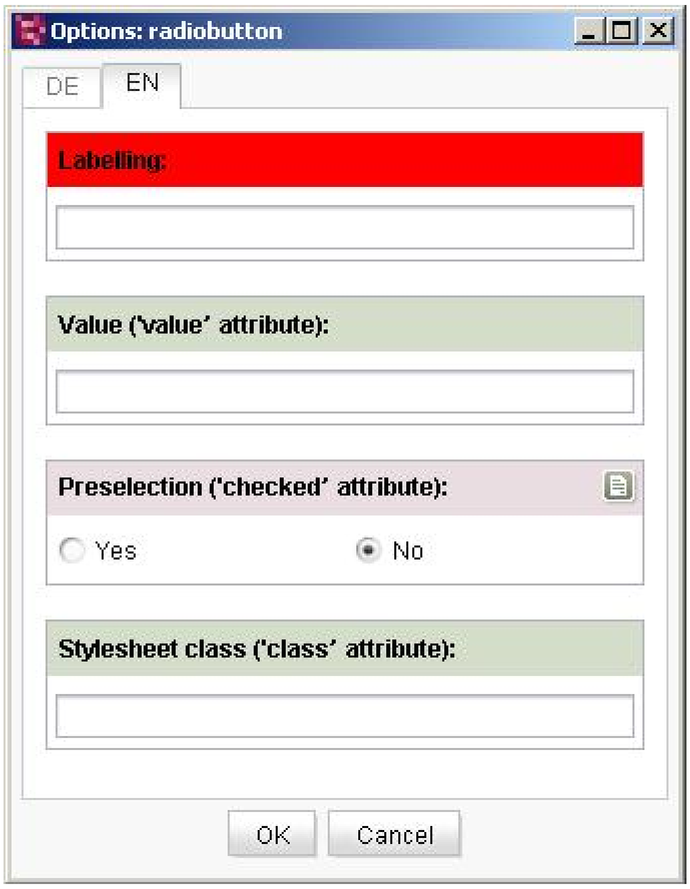

Options: radiobutton
DE
EN
Labelling:
Value ('value' attribute):
Preselection ('checked' attribute):
Yes [✓ No]
Stylesheet class ('class' attribute):
OK
Cancel

Figure 4-9: Form component "RadioButton"

Labelling: Use this text field to enter the labelling of the option.

Value ('value' attribute): Use this text field to enter the value for this option.

Preselection ('checked' attribute): These radio buttons can be used to make a preselection, whether this form field should be selected or not.

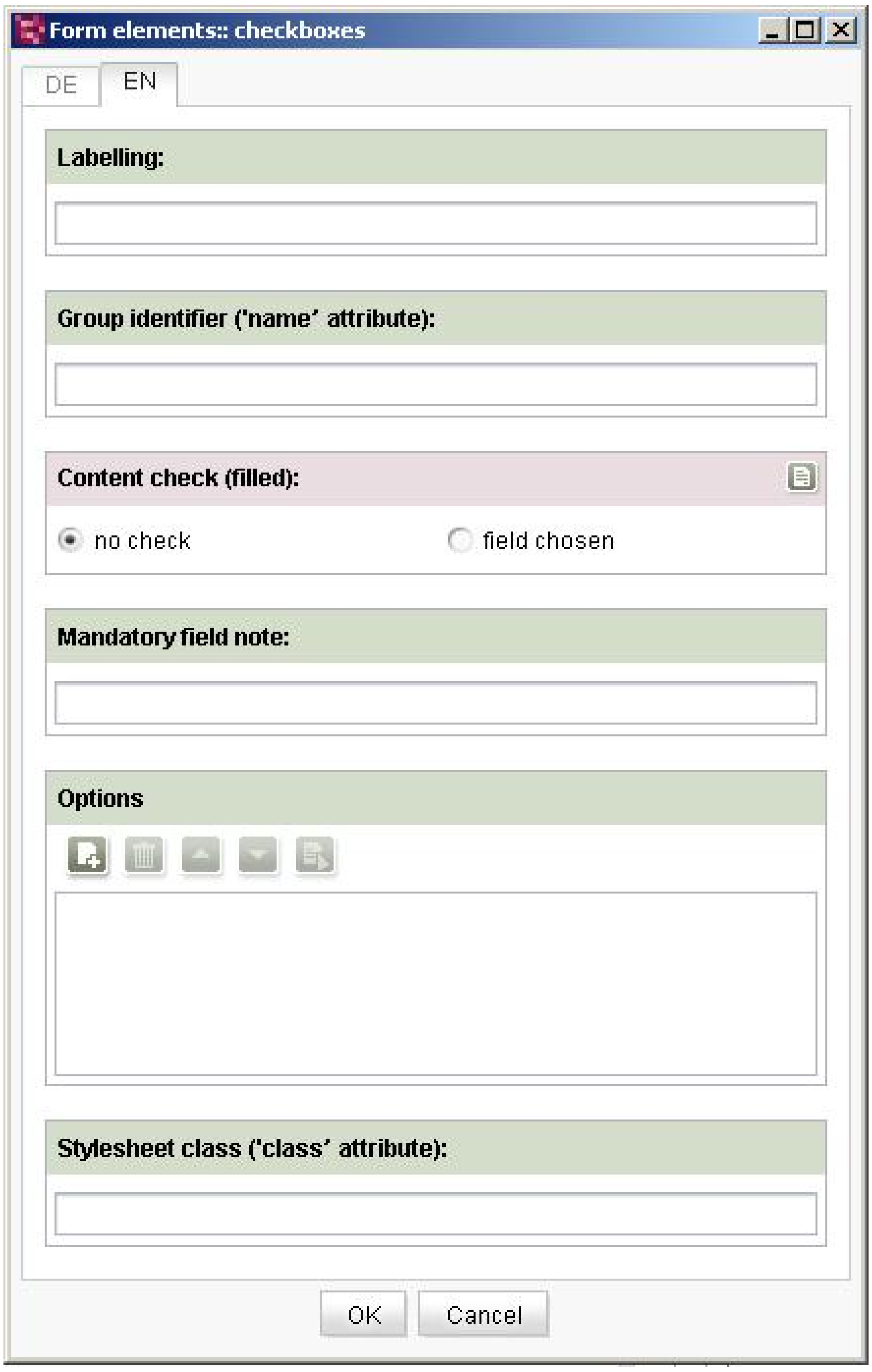
Stylesheet class ('class' attribute): Use this text field to enter the name of a CSS class.

4.3.4 Form component "Checkboxes"

This form component can be used to generate HTML checkboxes. When adding the checkboxes in the "Options" area it is necessary to ensure that a meaningful value is given for each checkbox. The checkboxes are shown with their corresponding designation (labelling), one after the other on the right-hand side of the component.

4.3.4.1 "Checkboxes" subcomponent

This component can be added within a "form-block" section. The values which are the same for all checkboxes are set in this section. The actual checkboxes are added within a ContentAreaList in this section.


Form elements:: checkboxes
DE
EN
Labelling:
Group identifier ('name' attribute):
Content check (filled):
no check
field chosen
Mandatory field note:
Options
Stylesheet class ('class' attribute):
OK
Cancel

Content check (filled): (● no check ○ field chosen)

Figure 4-10: Form component "Checkboxes"

Labelling: Use this text field to enter the title of the form field.

Group identifier ('name' attribute): Use this text field to enter the unique identifier.

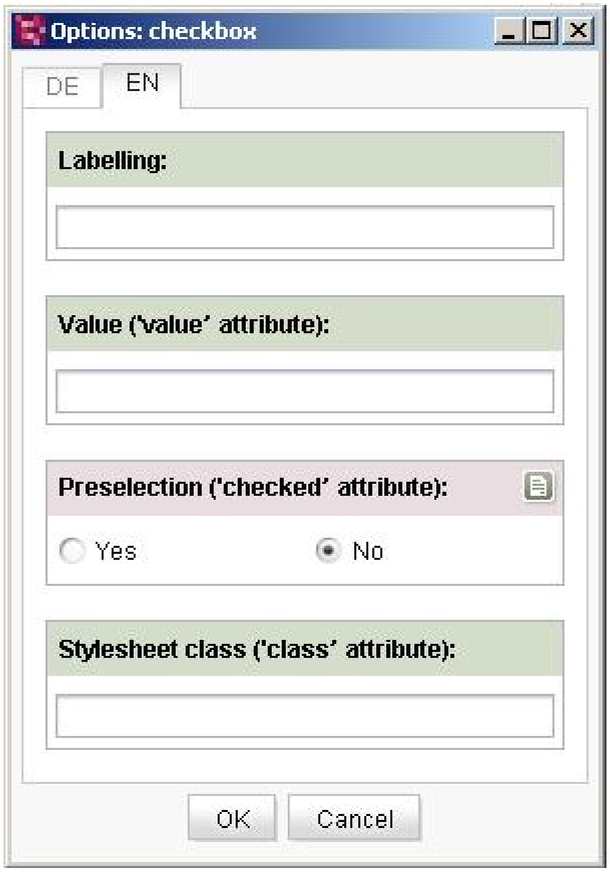
Content check (filled): These radiobuttons can be used to create a check, which checks whether or not at least one checkbox in this group has been selected.

Mandatory field note: A message which appears if the content check fails must be given for this type of form field.

Stylesheet class ('class' attribute): Use this text field to enter the name of a CSS class.

4.3.4.2 "Checkbox" subcomponent

This component can only be used within the form "Checkboxes" component. It is used to add the actual checkboxes within the component named above.



Options: checkbox

DE | EN

Labelling:

Value ('value' attribute):

Preselection ('checked' attribute):

Yes [✓ No]

Stylesheet class ('class' attribute):

OK | Cancel

Figure 4-11: Form component "Checkbox"

Labelling: Use this text field to enter the labelling of the option.

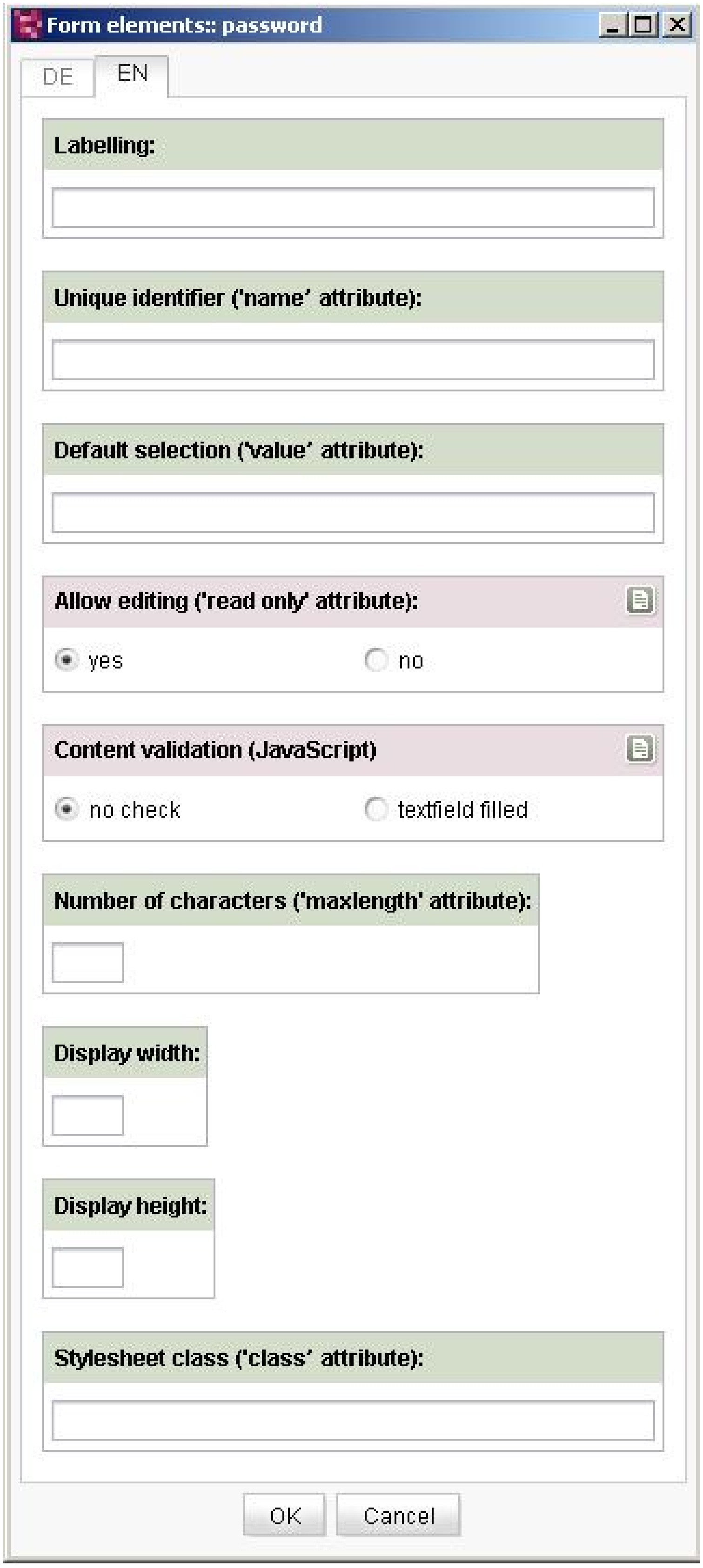
Value ('value' attribute): Use this text field to enter the value for this option.

Preselection ('checked' attribute): These radio buttons can be used to make a preselection, whether this checkbox should be selected or not.

Stylesheet class ('class' attribute): Use this text field to enter the name of a CSS class.

4.3.5 Form component "Password"

This component can be used to provide a password field for the user. Here too, the editor must assign a unique identifier for the component. Characters entered are shown in the component as "*" (asterisks).


Form elements:: password

DE | EN

Labelling:

Unique identifier ('name' attribute):

Default selection ('value' attribute):

Allow editing ('read only' attribute): [✓ yes] no

Content validation (JavaScript): [✓ no check] textfield filled

Number of characters ('maxlength' attribute):

Display width:

Display height:

Stylesheet class ('class' attribute):

OK | Cancel

Figure 4-12: Form component "Password"

Labelling: Use this text field to enter the title of the form field.

Unique identifier ('name' attribute): Use this text field to enter the unique identifier.

Default selection ('value' attribute): Use this text field to enter text with which the form field is filled.

Allow editing ('read only' attribute): This radio button is used to control whether the user may edit the form field or not.

Number of characters ('maxlength' attribute): This text field can be used to define the maximum number of characters that can be entered.

Display width: Use this field to specify the width of the form field in pixels.

Display height: Use this field to specify the height of the form field in pixels.

Stylesheet class ('class' attribute): Use this text field to enter the name of a CSS class.

4.3.6 Form component "Hidden"

This form component can be used to create fields which are not displayed in the browser, but can be evaluated in the servlet or PHP script, i.e. additional non-visible information. Here too, the editor <u>must</u> assign a <u>unique</u> identifier for the component.

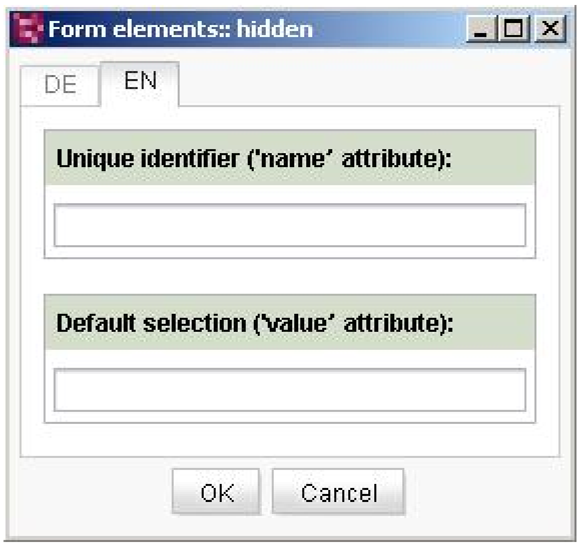

Form elements:: hidden
DE
EN
Unique identifier ('name' attribute):
Default selection ('value' attribute):
OK
Cancel

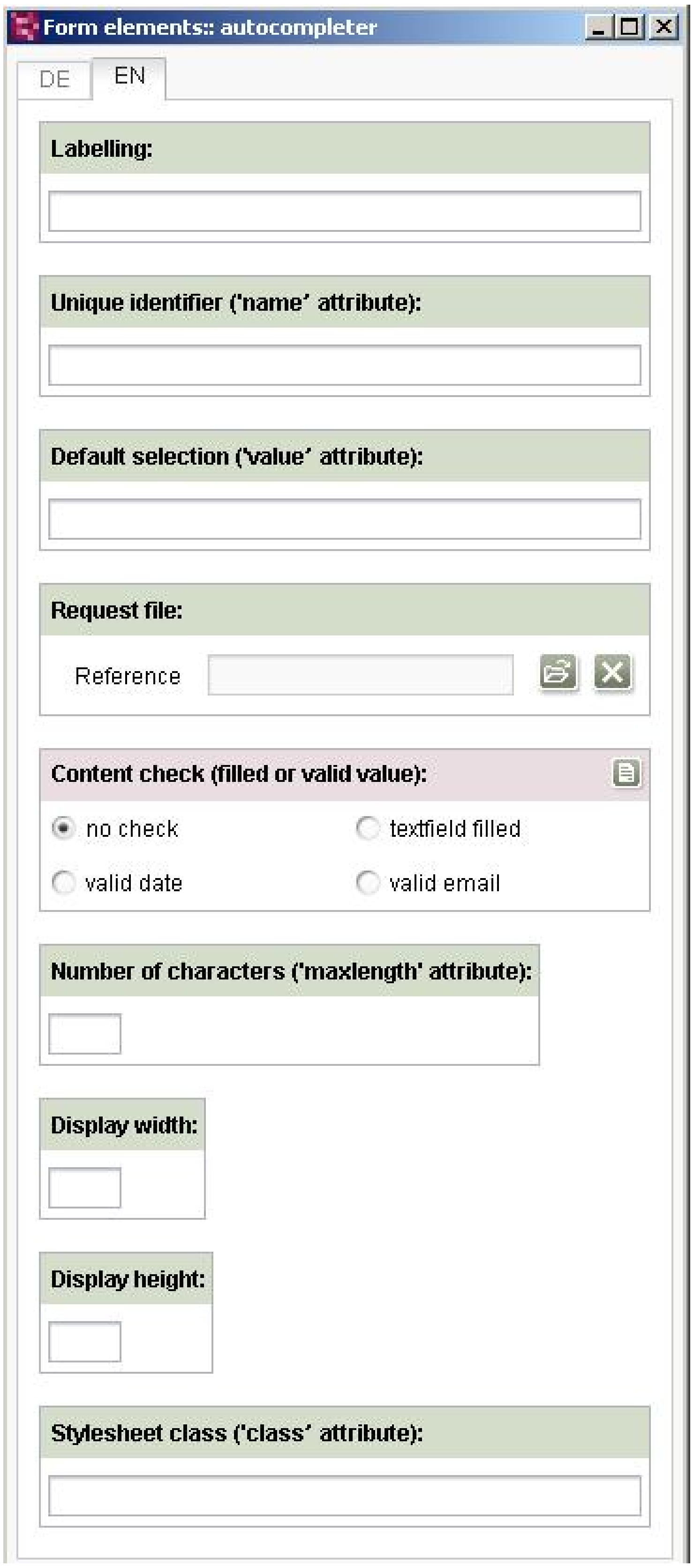
Figure 4-13: Form component "Hidden"

Unique identifier ('name' attribute): Use this text field to enter the unique identifier.

Default selection ('value' attribute): Use this text field to enter text with which the form field is filled.

4.3.7 Form "Autocompleter" component

This form component can be used to offer the user the convenience of having suggestions for completion of their input provided while they are typing. Here too, the editor <u>must</u> assign a <u>unique</u> identifier for the component.


Form elements:: autocompleter

DE EN

Labelling:

Unique identifier ('name' attribute):

Default selection ('value' attribute):

Request file:

Reference

Content check (filled or valid value):

[✓ no check] textfield filled

valid date valid email

Number of characters ('maxlength' attribute):

Display width:

Display height:

Stylesheet class ('class' attribute):

Figure 4-14: Form component "Autocompleter"

Labelling: Use this text field to enter the title of the form field.

Unique identifier ('name' attribute): Use this text field to enter the unique identifier.

Default selection ('value' attribute): Use this text field to enter text with which the form field is filled.

Request file: This component can be used to select a page from the structure which accepts the input and returns the applicable values.

Content check (filled or valid value): This radio button can be used to create a check which checks for content or valid data.

Number of characters ('maxlength' attribute): This text field can be used to define the maximum number of characters that can be entered.

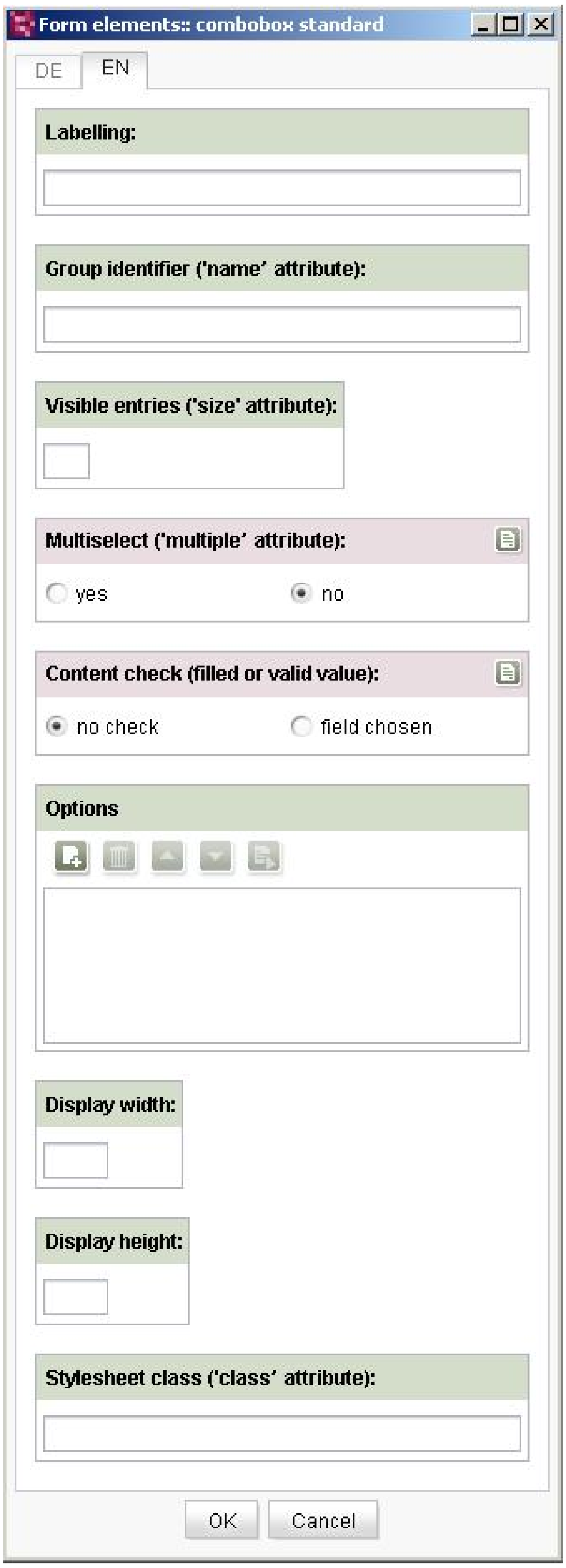
Display width: Use this field to specify the width of the form field in pixels.

Display height: Use this field to specify the height of the form field in pixels.

Stylesheet class ('class' attribute): Use this text field to enter the name of a CSS class.

4.3.8 Form component "combobox standard"

The form "combobox standard" component provides the editor with a convenient way of making a selection list available to the form user. The contents of the list and the number and layout are freely formattable. Here the editor must give a unique group designator.


Form elements:: combobox standard

DE | EN

Labelling:

Group identifier ('name' attribute):

Visible entries ('size' attribute):

Multiselect ('multiple' attribute):
yes [✓ no]

Content check (filled or valid value):
[✓ no check] field chosen

Options

Display width:

Display height:

Stylesheet class ('class' attribute):

OK Cancel

Figure 4-15: Form component "combobox standard"

Labelling: Use this text field to enter the title of the form field.

Group identifier ('name' attribute): Use this text field to enter the unique identifier.

Visible entries ('size' attribute): Use this field to enter the number of visible entries as a number.

Multiselect ('multiple' attribute): These radio buttons are used to control whether several entries in this form field can be selected or not

Content check (filled or valid value): This radio button can be used to create a check which checks for content or valid data.

Options: The contents / selection options of the form field can be entered in this list.

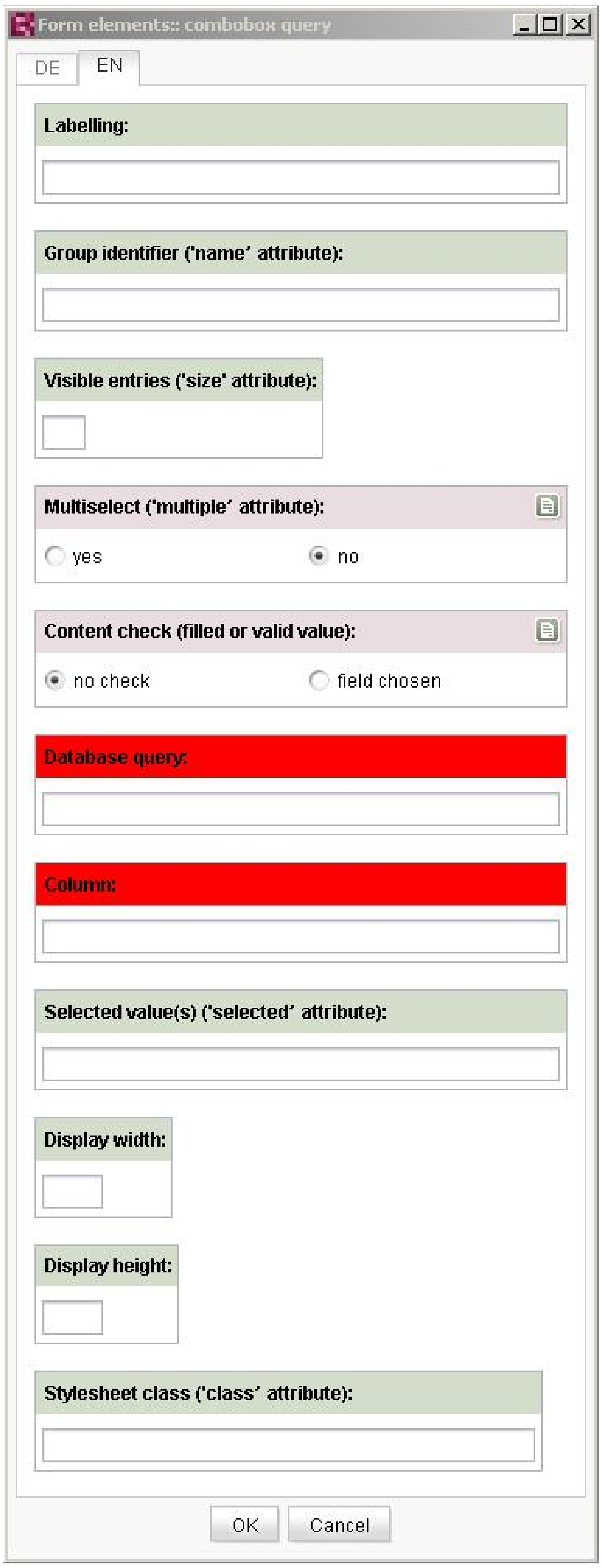
Display width: Use this field to specify the width of the form field in pixels.

Display height: Use this field to specify the height of the form field in pixels.

Stylesheet class ('class' attribute): Use this text field to enter the name of a CSS class.

4.3.9 Form component "combobox query"

The form "combobox query" component provides the editor with a convenient way of making a selection list available to the form user. The contents of the list and the number and layout are freely formattable. This special SelectBox filters and outputs the selection options from the database via a "Query". Here the editor must give a unique group designator.



Form elements:: combobox query

DE EN

Labelling:

Group identifier ('name' attribute):

Visible entries ('size' attribute):

Multiselect ('multiple' attribute):

yes [✓ no]

Content check (filled or valid value):

[✓ no check] field chosen

Database query:

Column:

Selected value(s) ('selected' attribute):

Display width:

Display height:

Stylesheet class ('class' attribute):

OK Cancel

Figure 4-16: Form component "combobox query"

Labelling: Use this text field to enter the title of the form field.

Group identifier ('name' attribute): Use this text field to enter the unique identifier.

Visible entries ('size' attribute): Use this field to enter the number of visible entries as a number.

Multiselect ('multiple' attribute): These radio buttons are used to control whether several entries in this form field can be selected or not

Content check (filled or valid value): This radio button can be used to create a check which checks for content or valid data.

Database query: Use this text field to enter the name (UID) of a database query (Query) in the Template Store.

Column: Use this text field to enter the name of a database column or database field. The values of this column are available to choose from in this form field.

Selected value(s) ('selected' attribute): Use this text field to specify one or several values which are preselected in the form field. If you specify several values, separate them with a "," (comma).

Display width: Use this field to specify the width of the form field in pixels.

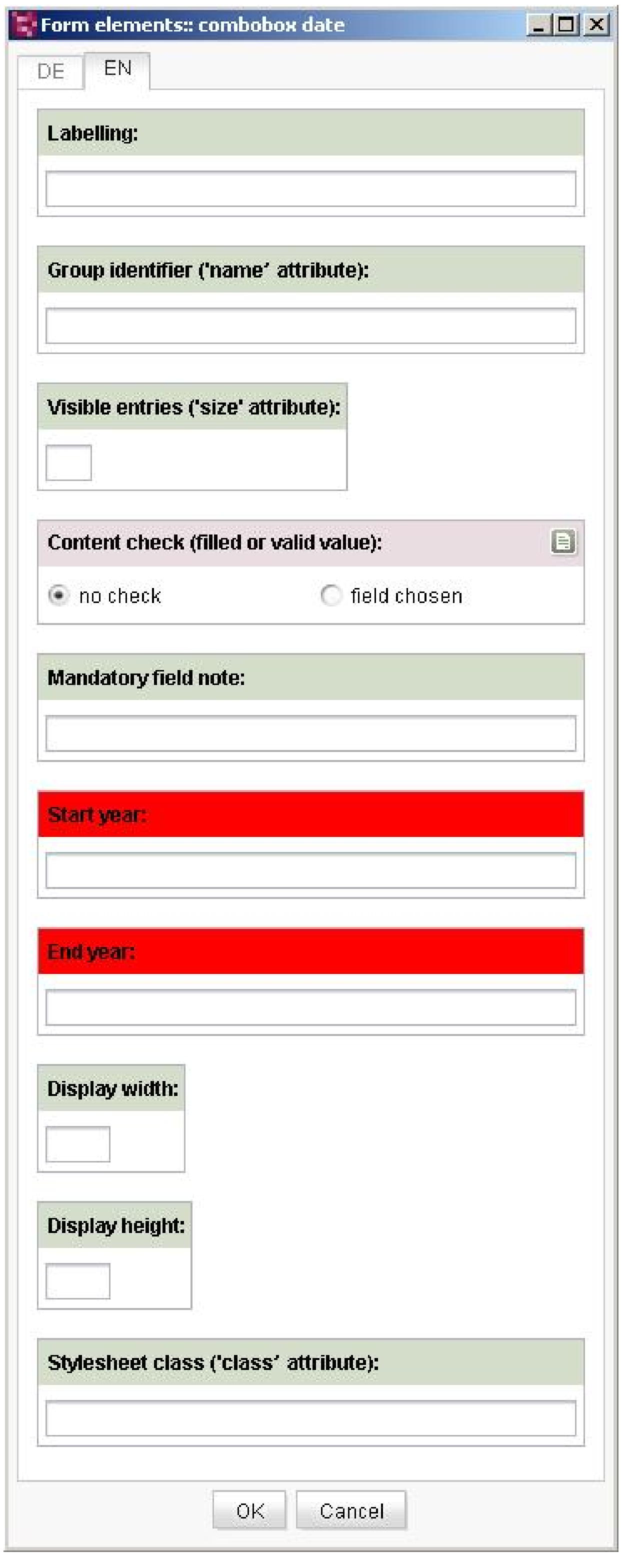
Display height: Use this field to specify the height of the form field in pixels.

Stylesheet class ('class' attribute): Use this text field to enter the name of a CSS class.

4.3.10 Form component "combobox date"

The form "combobox date" component provides the editor with a convenient way of making a selection list available to the form user. The contents of the list, the number of entries and layout are freely formattable. Three boxes are automatically generated in this special SelectBox, which can be used to conveniently select a date. Here the editor must give a unique group designator.

To convert into a correct date format within the servlet, a "toDate_“! is placed in front of the unique identifier, in order to make the field identifiable for the servlet. If "date" was entered as the unique identifier, the field's identifier is "toDate_date" and must therefore also be used in this form, e.g. in the mail template.



Form elements:: combobox date

DE | EN

Labelling:

Group identifier ('name' attribute):

Visible entries ('size' attribute):

Content check (filled or valid value):

(● no check) ○ field chosen

Mandatory field note:

Start year:

End year:

Display width:

Display height:

Stylesheet class ('class' attribute):

OK | Cancel

Figure 4-17: Form component "combobox date"

Labelling: Use this text field to enter the title of the form field.

Group identifier ('name’ attribute): Use this text field to enter the unique identifier.

Visible entries ('size' attribute): Use this field to enter the number of visible entries as a number.

Content check (filled or valid value): This radio button can be used to create a check which checks for content or valid data.

Start year: Use this text field to enter a date (year). Date selection is possible from this value up to the value of the "End year" field.

End year: Use this text field to enter a date (year). Date selection is possible from the value of the "Start year" field up to this value.

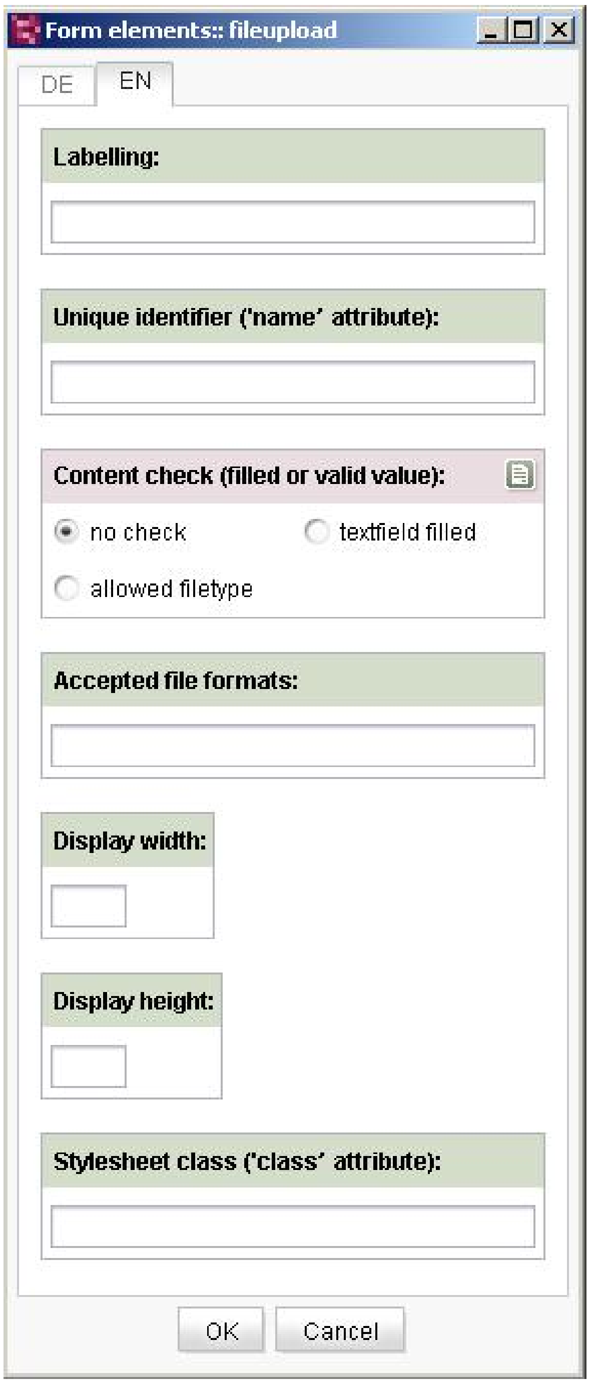
Display width: Use this field to specify the width of the form field in pixels.

Display height: Use this field to specify the height of the form field in pixels.

Stylesheet class ('class’ attribute): Use this text field to enter the name of a CSS class.

4.3.11 Form component "fileupload"

This form component enables the user to send files via the form. In addition, it is possible to limit the file formats. Here too, the editor must assign a unique identifier for the component.



Form elements:: fileupload

DE EN

Labelling:

Unique identifier ('name' attribute):

Content check (filled or valid value):

no check [✓] / textfield filled / allowed filetype

Accepted file formats:

Display width:

Display height:

Stylesheet class ('class' attribute):

Figure 4-18: Form component "fileupload"

Labelling: Use this text field to enter the title of the form field.

Unique identifier ('name' attribute): Use this text field to enter the unique identifier.

Content check (filled or valid value): This radio button can be used to create a check which checks for content or valid file types.

Accepted file formats: Use this text field to enter a comma-separated list of file extensions. An evaluation only takes place if "valid file type" is activated for the content check. Several file extensions must be separated from each other by "|".

Display width: Use this field to specify the width of the form field in pixels.

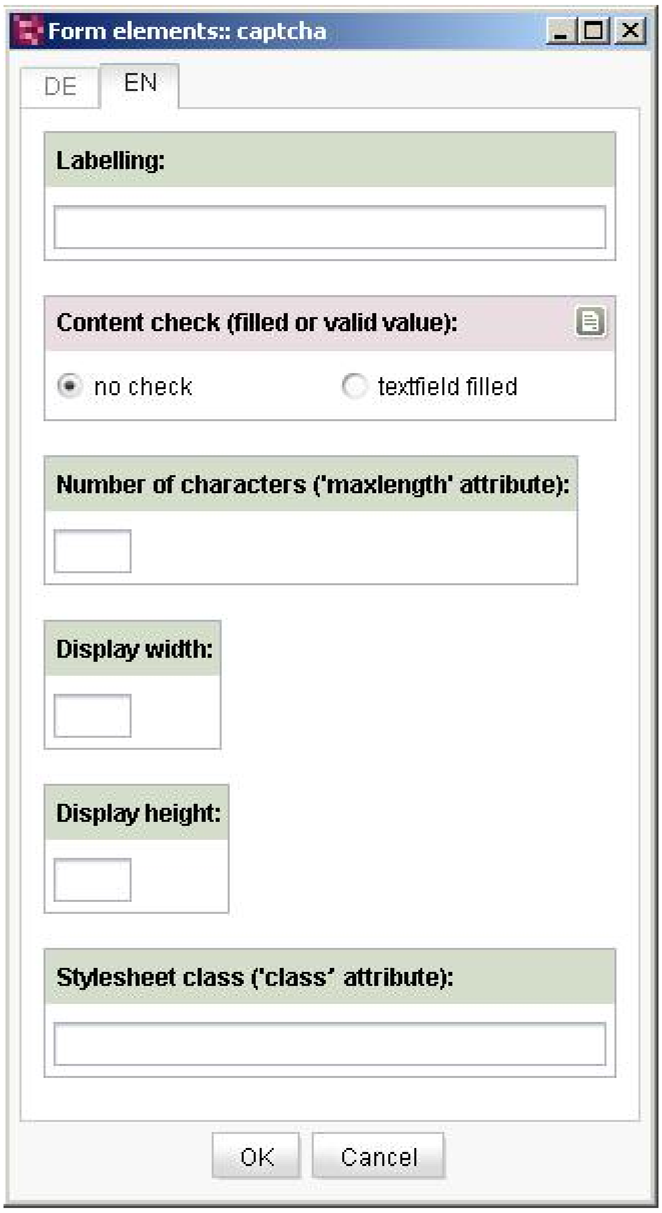
Display height: Use this field to specify the height of the form field in pixels.

Stylesheet class ('class' attribute): Use this text field to enter the name of a CSS class.

4.3.12 Form component "captcha"

This form component is used to generate a captcha graphic, a link, which generates a new graphic, and a text field for entering the captcha code, in the form. This component should be used to refuse use of the form by non-human users.

Select the "Captcha Validation" checkbox in the "form-start" section so that the input is checked by the servlet.



Form elements:: captcha

DE EN

Labelling:

Content check (filled or valid value):

(● no check) ○ textfield filled

Number of characters ('maxlength' attribute):

Display width:

Display height:

Stylesheet class ('class' attribute):

Figure 4-19: Form component "captcha"

Labelling: Use this text field to enter the title of the form field.

Number of characters ('maxlength' attribute): This text field can be used to define the maximum number of characters that can be entered.

Display width: Use this field to specify the width of the form field in pixels.

Display height: Use this field to specify the height of the form field in pixels.

Stylesheet class ('class' attribute): Use this text field to enter the name of a CSS class.

Overlapping of the characters is affected by the height, width and number of characters to be rendered (see Chapter 2.3.1 page 13). Always ensure that the characters are not too easy to read, as otherwise they can be read by spam robots. With the default configuration of 100 x 100 pixels with 6 characters, a captcha can look like this:

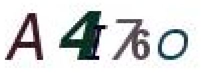


Figure 4-20: Form "captcha" graphic component

5 Media Required and their Function

5.1 Stylesheet file

Configuration fields for stylesheet classes are provided for virtually every component to enable form components to be quickly and easily adjusted. Each field can therefore be assigned its own individual style by specifying a stylesheet class. An exemplary stylesheet file "formedit_css" is added to the project when the project component is installed. The class names should be entered accordingly in the configuration fields of the components. We urgently recommend adjusting the stylesheet file to the individual design requirements.

```
<script type="text/javascript"
src="$CMS_REF(media:"formedit_js")$"></script>
```

In order to be able to use your own stylesheet classes, they must be declared in the HTML header of the (page) templates by means of the <style> tag, or added to an existing or new stylesheet file in the project.

5.2 Javascript file

Javascript functions are used within the templates supplied with the module, e.g. to enable individual form blocks to be shown and hidden. These functions are in the medium "formedit_js". The medium must be referenced within the HTML header of all (page) templates in which Javascript functions can be used:

```
<script type="text/javascript"
src="$CMS_REF(media:"formedit_js")$"></script>
```

5.3 jQuery

This free Javascript framework provides various functions which are used by the templates supplied with the module. In addition, this library is a requirement for use of the form "Autocompleter" component (see Chapter 5.5 page 72) and form field validation (see Chapter 5.4 page 72). The version supplied is compatible with the plug-ins supplied. If an update is necessary, ensure that it is compatible with the plug-ins. jQuery can be used in parallel with other frameworks, e.g. MooTools. The framework must be referenced within the HTML header in all templates which use the functions of jQuery or its plug-ins:

```
<script type="text/javascript"
src="$CMS_REF(media:"jquery")$"></script>
```

Further information and updates: http://www.jquery.com

5.4 Form validation

The jQuery "validate" plug-in provides a convenient option for checking the form fields used for correct content on entering the data. The form components use the basic functions of this plug-in. This plug-in can also be used to check dependencies between form fields and other limitations and constraints. To use this function in the project, the "jquery_validate.js" file must be referenced within the HTML header of all (page) templates in which form components can be used:

```
<script type="text/javascript"
src="$CMS_REF(media:"jquery_validate")$"></script>
```

Alternatively, it is possible to add the function to an existing Javascript file.

This plug-in also supplies the (error) messages in 19 languages, which appear – inline – in the event of missing or incorrect input components. These error messages are deposited in the language-dependent medium "jquery_validate_messages" and if necessary can be changed and extended. In order for these (error) messages to appear, the medium must be referenced within the HTML header of all (page) templates in which form components can be used:

```
<script type="text/javascript"
src="$CMS_REF(media:"jquery_validate_messages")$"></script>
```

Further information and updates: http://plugins.jquery.com/project/validate

5.5 Autocompleter

A jQuery plug-in is also used for the "Autocompleter" component. However, it only provides the function of setting up a request with the entered characters to another page and displaying the response to the user. The "jquery_autocomplete" file must be referenced within the HTML header of all (page) templates in which the form "Autocompleter" component can be used:

```
<script type="text/javascript"
src="$CMS_REF(media:"jquery_autocomplete")$"></script>
```

For further information on complete integration, please refer to Chapter 6 page 73.

Further information and updates: http://plugins.jquery.com/project/autocompletex

6 "Auto Completion" Concept

In real terms, the form "Autocompleter" component only provides a completely normal text input component. The logic which provides this component with the autocompletion function consists of two parts:

- The **first part** is the jQuery "autocomplete" plug-in (see Chapter 5.5 page 72). This plug-in enables the characters entered to be sent to another dynamic server or a servlet in the background. Further, it accepts the reply (response) and shows it to the user.
- The **second part** must be implemented for each specific case: To do this, a dynamic page must be generated (e.g. jsp or php), which accepts a request from the autocompleter and then uses it to browse through a source (e.g. XML, database or CSV). In another step, the data found must then be returned to the autocompleter by means of a response.

This concept is implemented in the templates supplied with the module on the basis of an example XML file. The logic is implemented on the jsp-side and also requires the Java "jdom" library.

The Java "jdom" library is not part of the "FirstSpirit FormEdit" module, but must be separately copied onto the applicationServer or FirstSpirit Server or installed as a module. Further information: http://www.jdom.org.

Example:
This example is based on the following XML file ("xmlDataBase" page template):

```
<?xml version="1.0" encoding="UTF-8"?>
<jobs>
  <job jobnr="Project Manager (jb-1742-a01)"
jobdescription="Project Manager"/>
  <job jobnr="Apprentice (jb-1743-a02)"
jobdescription="Apprentice"/>
  <job jobnr="Template Developer (jb-1744-a03)"
jobdescription="Template Developer"/>
  <job jobnr="Product Development Manager (jb-1745-a04)"
jobdescription="Product Development Manager"/>
</jobs>
```

On using the form "Autocompleter" field, after entering, e.g. 3 characters, a request is sent to the page referenced in the form component. This page is based on the "autocompleterequest" page template. Logic is implemented in this page template, which accepts this request and the transferred character chain.

```
incomingValue = (String) request.getParameter("q");
```

Within the referenced **XML source,** the **comparator attribute** is searched for occurrence of the requested character string.

```
for (Element child : children) {
   if (incomingValue == null ||

child.getAttribute(compareAttr).getValue().toLowerCase().contains(
incomingValue.toLowerCase())) {
buffer.append(child.getAttribute(resultAttr).getValue() + "\n");
}
}
```

If an applicable entry is found, a response is sent to the form input component:

```
Project Manager
Product Development Manager
```

The data found is now shown to the form user, e.g. here the value of the "jobdescription" attribute (**return text**). If the user now selects an option, it is saved as the value of the form field.

7 Case Study: "Competition"

This example describes all actions to be performed by the editor, which are necessary to create a form with which a user can register for a competition.

The data entered by the user is to be used on the one hand for a personalised participation confirmation and on the other is to be stored in a database containing all participants.

To carry out this example, the "FirstSpirit FormEdit" module must be installed for a FirstSpirit project, as described in Chapter 2, from page 7. Furthermore, a database and an available e-mail server are required.

7.1 Creating the form

Create a new page in the Page Store and choose the "form" template. Add the "form start", "form block" and "form end" sections, one after the other in a section area of the page:

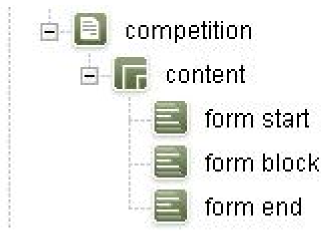
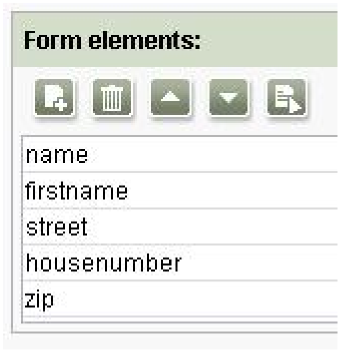


Figure 7-1: Create page with form sections

Now enter a heading and form name in the "form start" section. Choose "Post" as the send method and enable client-side content checking. If you already have a confirmation page and an error page in your structure, you can reference them here.

Now switch to the "form block" section and create a text field for each of the following: Name, first name, street, house number, post code, town/city and e-mail address. Use the content check "Field filled" on each. To enable easy filing of the data in the database, it is advisable to choose the column name in the database as the unique identifier. For this example we use "name“, "firstname“, "street", "housenumber", "zip", "city" and "email" as unique identifiers:



Form elements:

name
firstname
street
housenumber
zip

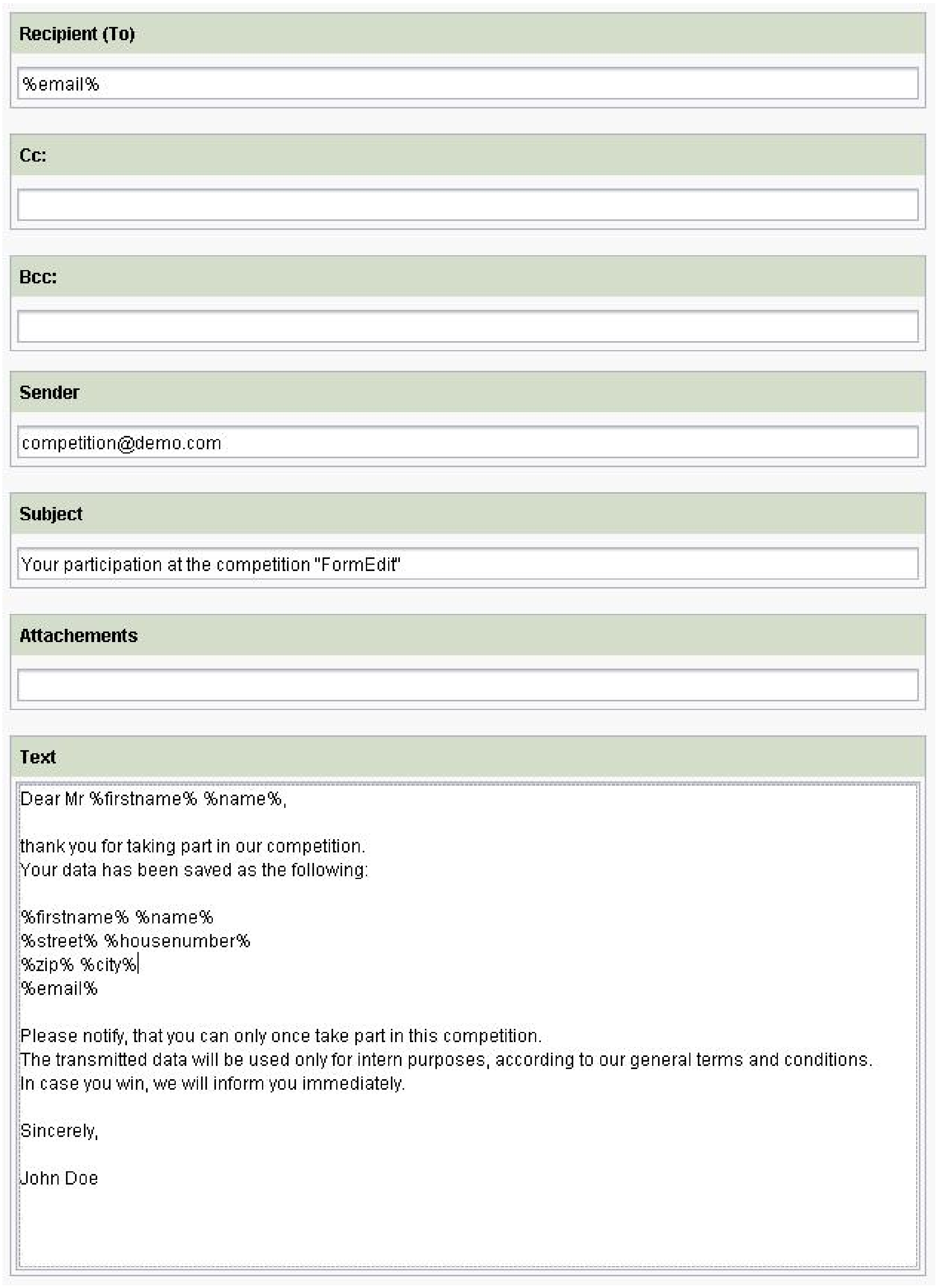
Figure 7-2: Form elements

All the necessary fields in the "form end" section are already completed, but you can now adjust them to your wishes.

7.2 Creating the mail template

Create a new page in the Page Store on the basis of the "mailtemplate" template. We use the field from the form as the recipient, as the e-mail is to be sent to the form user. I.e., in this example, **%e-mail%**.

You can also define the term yourself, just like the message text and the sender's address. You can use %uniqueIdentifier% in the message text to access all form fields:



Recipient (To)

%email%

Cc:

Bcc:

Sender

competition@demo.com

Subject

Your participation at the competition "FormEdit"

Attachements

Text

Dear Mr %firstname% %name%,

thank you for taking part in our competition.
Your data has been saved as the following:

%firstname% %name%
%street% %housenumber%
%zip% %city%
%email%

Please notify, that you can only once take part in this competition.
The transmitted data will be used only for intern purposes, according to our general terms and conditions.
In case you win, we will inform you immediately.

Sincerely,

John Doe

Figure 7-3: Mail template

Now use Drag&Drop to drag the mail template you have created into the Site Store or reference it via the context menu from the Site Store.

7.3 Creating the logger configuration (e-mail)

Now switch back to your "form start" section and add a new data record in the "Processing" component.

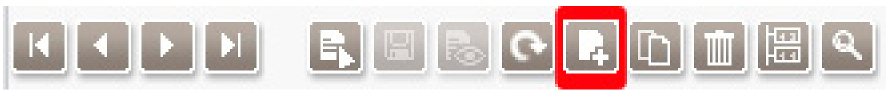
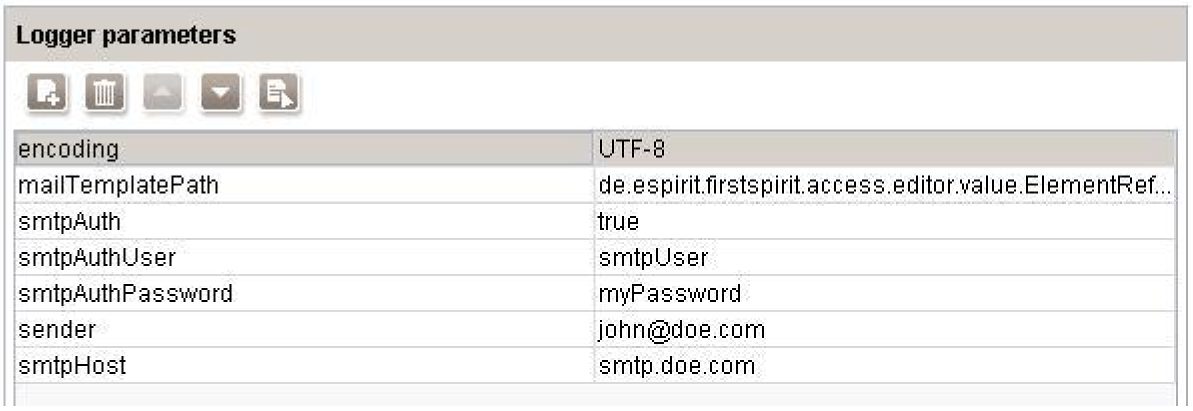


Figure 7-4: Add data record

In the following dialog, enter a name for the e-mail processing and choose "MailLogger" as the LoggerType. You can add an optional description.

Now add a new data record to the "Logger Parameter" component by clicking "Add Section" and select the "logger-text-value" entry and click "OK". In the following dialog, enter **"smtpHost**" as the parameter name and the address of your outgoing mail server as the parameter value, e.g. smtp.ANOther.de. Repeat this for the **"encoding**" parameter, using the name of the encoding you require (here: "UTF-8“).

Now create a new section; however, this time using the "logger-template-ref" template. Now enter **"mailTemplatePath**" as the parameter name and choose the mail template created by you in the "Mailtemplate" component. Further parameters are explained in Chapter 3.2.4 page 23.



Logger parameters

encoding	UTF-8
mailTemplatePath	de.espirit.firstspirit.access.editor.value.ElementRef...
smtpAuth	true
smtpAuthUser	smtpUser
smtpAuthPassword	myPassword
sender	john@doe.com
smtpHost	smtp.doe.com

Figure 7-5: MailLogger parameter

Finally, select "Activated" status in the component (see Chapter 3.1 page 15) and click the "**Save**" symbol in the top left-hand corner. The logger created by you is now automatically listed in the "Processing" component, in your "form start" section.

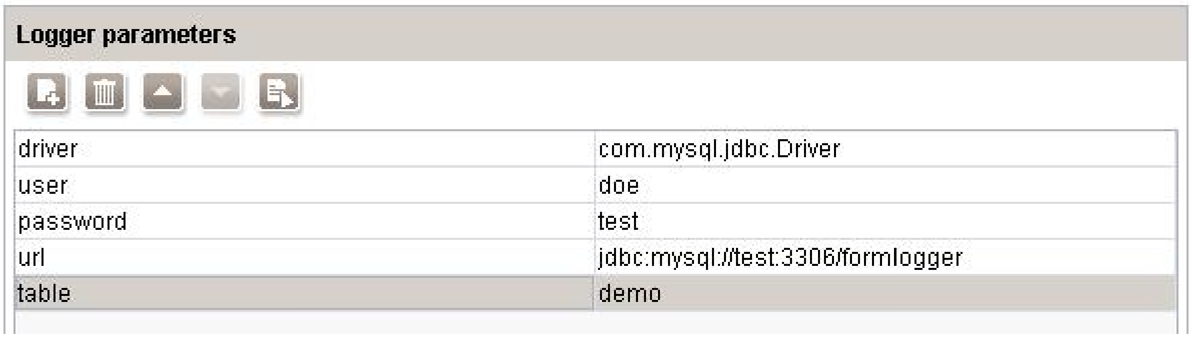
7.4 Creating the logger configuration (database)

The second configuration is created in exactly the same way as described in Chapter 7.3 page 79. However, here you select "**JdbcLogger**" as the logger type and create all parameters on the basis of the "logger-text-value" template. Please create the **"driver**" parameter first with the name of your database driver, e.g. "com.mysql.jdbc.Driver" for a MySQL database.

Please ensure that the database driver has already been added to the FirstSpirit server.

Use the **"url**" parameter to pass the address to your database, e.g. "jdbc:mysql://muster:3306/formlogger" and the **"table"** parameter to pass the name of your table (here: "demo“).

Now create the "user" and "password" parameters and fill these with the login data for the database, which is usually made available to you by your database administrator.


Logger parameters

driver	com.mysql.jdbc.Driver
user	doe
password	test
url	jdbc:mysql://test:3306/formlogger
table	demo

Figure 7-6: JdbcLogger parameter

Further parameters are explained in Chapter 3.2.3 page 20.

7.5 Creating the configuration file "fs-formlogger.ini"

Please read Chapter 3.5 from page 30.

7.6 Referencing and deploying

Then reference all the pages you have created in the structure and, if applicable, release them. If you or the project administrator have set up the "FirstSpirit FormEdit" module for staging, following generation, your form should be visible in its generated status (http://www.youreditorialserver.com:8000/fs4staging_<project_id>/...) and should be able to be used.

8 Legal Notices

The "FirstSpirit FormEdit" module is a product belonging to e-Spirit AG, Dortmund, Germany.

The user may only use the module as defined under the terms of the licence agreed with e-Spirit AG.

Details of possible external software products used, not produced by e-Spirit AG, their own licences and any update information, is given on the homepage of each FirstSpirit server, in the "Legal Notices" area.