

Developer Manual

Version 7.1.0



Formcentric for FirstSpirit: Developer Manual

Copyright © 2024 Formcentric GmbH
Breite Str. 61, 22767 Hamburg
Germany

The contents of this document – whether in whole or in part – may not be reproduced, conveyed, disseminated or stored in any form whatsoever without obtaining prior written permission from Formcentric GmbH.

Disclaimer

We reserve the right to alter the software and the contents of the manual without notice. We accept no liability for the accuracy of the contents of the manual, nor for any losses that may result from the use of this software.

Trademarks

In the course of this manual, references may be made to trademarks that are not explicitly marked as such. Even if such a mark is not given, the reader should not conclude that the name is free of third-party rights.

1. Introduction	1
1.1. Terminology	1
2. Overview	2
3. System requirements	3
3.1. Java	3
3.2. FirstSpirit	3
4. Installation and configuration	4
4.1. Installing the Formcentric module	4
4.2. Licence file	5
4.3. Installing the Formcentric web editor	5
4.4. Formcentric web applications	6
4.4.1. Installation	6
4.4.2. Configuration	8
4.5. Analytics Backend web application	17
4.5.1. Installation	17
4.5.2. Configuration	18
4.6. Analytics Reporting web application	26
4.6.1. Installation	26
4.6.2. Configuration	27
4.7. Solr web application	28
4.8. Installing the project component	29
4.9. Configuring the publication tasks	30
4.10. Password encryption	33
5. Extending the FirstSpirit project	35
5.1. Paragraph style sheet	35
5.1.1. Properties tab	35
5.1.2. Form tab	35
5.1.3. Internet (HTML) tab	45
5.2. formcentric_headless_url script	47
5.3. formcentric_encrypted_form script	47
5.4. formcentric_encrypted_refs script	48
5.5. formcentric_login_ticket script	48
5.6. Page template	48
5.7. Themes	49
5.8. CSS	50
6. Programming and customisation	51
6.1. Development workspace	51
6.2. Monday Maven plugin	52
6.3. Extending the input component in Site Manager	55
6.3.1. Developing a NodeEditorPane	56
6.3.2. Extending the EditorSetup class	57
6.3.3. Extending the Form Editor GUI object model	58
6.4. Extending the ContentCreator web application	59
6.4.1. Adding a new form element	59
6.4.2. Adding a new validator	62

6.4.3. Adding a new action	63
6.4.4. Adding new element properties	65
6.4.5. Input elements for element properties	66
6.4.6. Editing existing form elements	73
6.4.7. User interface internationalisation	73
6.5. Extending the Spring MVC web application	73
6.5.1. Spring configurations	73
6.5.2. Usage without Formcentric Analytics	79
6.5.3. Formcentric licence file	80
6.5.4. Web security	80
6.5.5. Saving the form status	83
6.5.6. Implementing an action	84
6.5.7. Adding variables for pre-filling form fields	86
6.5.8. Implementing a REST service	87
6.5.9. Template development	91
6.5.10. JavaScript	112
6.6. Extending the headless web application	117
6.6.1. Implementing an action	118
6.6.2. Adding variables for pre-filling form fields	119
6.6.3. Implementing a REST service	120
6.7. Formcentric client	121
6.7.1. Theme	122
6.7.2. Initialisation	122
6.7.3. Templates	122
6.7.4. Special integration scenarios	130
6.7.5. Troubleshooting	131

1. Introduction

This manual describes how to install, configure and extend the Formcentric form manager extension. It is intended to be read by administrators and developers. To get the most out of this document, you will need knowledge of FirstSpirit from both an administrator and user perspective, as well as experience in developing Java software.

Chapter 4, *Installation and configuration* : describes the steps that you need to complete in order to install and configure Formcentric. The installation instructions assume that you have deployed Formcentric Analytics. If this isn't the case, then you can skip all of the sections that relate to Formcentric Analytics.

Chapter 5, *Extending the FirstSpirit project* : describes the extensions and modifications that you complete within a FirstSpirit project.

Chapter 6, *Programming and customisation* : shows how you can extend Formcentric to offer additional functionality.

1.1. Terminology

This manual makes use of the following terms:

Term	Definition
Form author	The person that creates and edit forms.
User	The person that fills out a form.
Form	An HTML web form displayed in a web browser.
Form elements	All of the elements used when constructing a form (input fields, drop-down lists, check boxes, etc.).
Form Editor	Input component in FirstSpirit SiteArchitect or ContentCreator, used for creating and editing forms.
Form data	The data entered into the form by the user.
Editor	FirstSpirit SiteArchitect or ContentCreator
Formcentric client	React application for browser-based form presentation.

2. Overview

On the editing side, Formcentric extends the FirstSpirit system to provide an input component with which form authors can create and edit dynamic web-based forms.

Two additional web app modules are available for presenting the forms as well as processing the submitted form data. The first is a conventional web application with server-side generation of the HTML output (referred to in this manual as the *Spring MVC web app*) while the second is a modern headless application with client-side rendering. Both web applications include various Spring controllers for processing the data. A form controller validates the data it receives and forwards these to purpose-built actions, which then carry out the final processing. This approach permits the integration of various backend systems, such as mail servers, Formcentric Analytics or databases.

The Analytics component included with Formcentric provides storage and reporting functions for the form data submitted. Formcentric Analytics consists of two global web app modules. The Backend web app module is responsible for storing the data in a relational database. To do so, it provides a REST interface, which clients can use to communicate with the Backend. Alongside the actual form data, the Backend also stores form sessions if this feature has been activated for the form in question.

The Reporting web app module is a modern, single-page application with which the form data stored in the Backend can be displayed, deleted and exported.

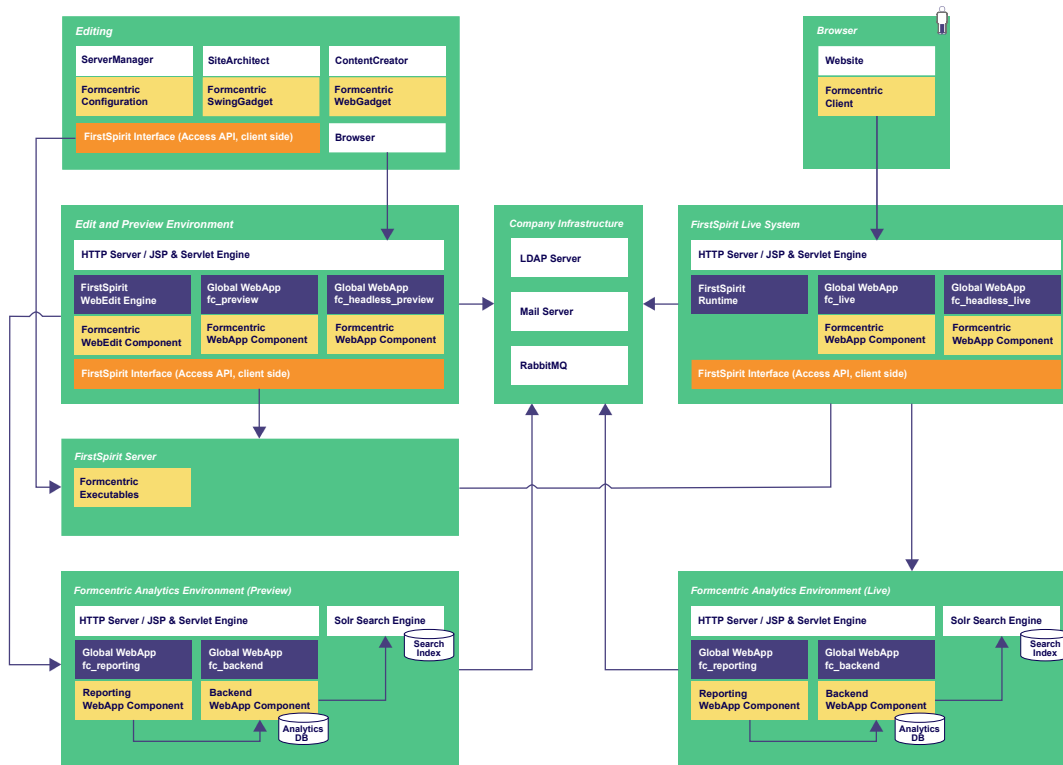


Figure 2.1. Architecture overview

3. System requirements

To be able to use Formcentric 7.1.0, you need to deploy FirstSpirit version 5.2.201209 or higher.

Formcentric requires the JavaScript framework “jQuery” from version 1.12.4.

If the PDF action loads the required PDF template documents from a local directory, then the external program *rsync* will be required on the FirstSpirit Server. For further information about installing and configuring “rsync” in conjunction with FirstSpirit, please see section 10 of the FirstSpirit “Administrator Documentation”.

3.1. Java

Formcentric can be used with the following Java versions. Execution is within a servlet container that is compliant with the JSP 2.3 and Java Servlet 3.1 specifications.

Java	Status
OpenJDK 17	supported (<i>recommended</i>)

The requirements for the Java version always depend on the FirstSpirit version used. From FirstSpirit 2023-11, at least Java 17 is required.

3.2. FirstSpirit

You can use Formcentric with the following versions of FirstSpirit. If you are using a different version, then modifications may need to be made to the FirstSpirit Runtime Java-Library (see Section 4.1, “Installing the Formcentric module”). This will certainly be the case if you are using a more recent version.

FirstSpirit	Status
FirstSpirit 2402.11 - 2404.08 (Isolated Mode)	supported (<i>recommended</i>)

4. Installation and configuration

You install and configure Formcentric using FirstSpirit's *Server and Project Configuration*.

4.1. Installing the Formcentric module

To do so, go to *Server Properties* and select the menu option *Modules*. Then click the *Install* button. This opens a file selection pop-up, in which you need to select the *formcentric-7.1.0.fsm* archive file to be installed.

The Formcentric module contains a service for querying licence information. This service is required for both installing and running Formcentric. After loading the archive file, the system therefore asks you if you want the services included in the module to be started automatically. Answer this question with *Yes*.

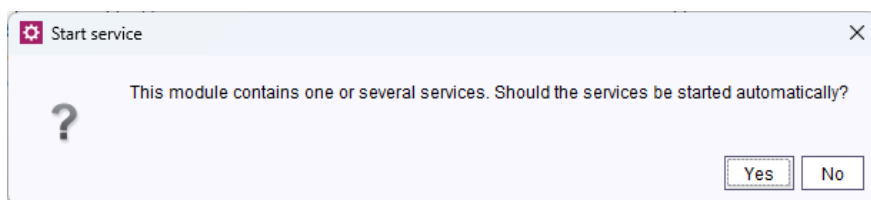


Figure 4.1. Starting services automatically

Once you have successfully installed the file, the system then displays the *Formcentric (I)* module with the components that it includes.

Now select the new *Formcentric (I)* entry, click *Configure*, check the check box *All rights* and confirm your change.

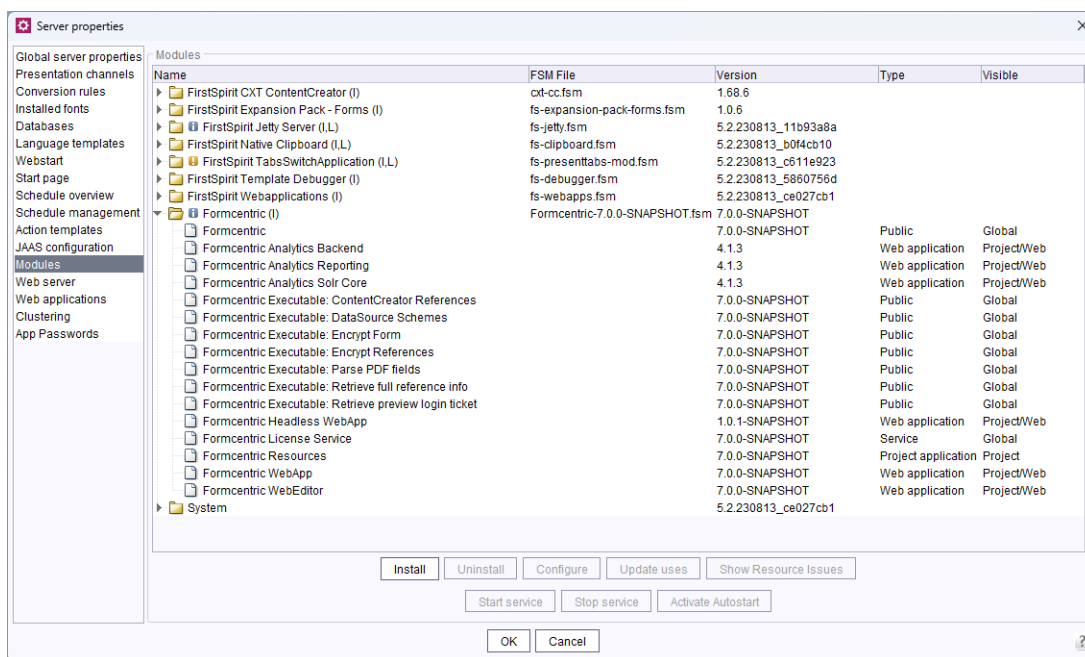


Figure 4.2. Module list under Server Properties

Module permissions become effective only once the FirstSpirit server has been restarted. Once restarted, continue the installation by following the instructions below.

4.2. Licence file

Double-click the *Formcentric License Service* entry in the module overview to open a configuration dialog. Here, upload the licence file that you have received from Formcentric. To do so, click the *Import licence file* button and select the licence file in the file selection pop-up that then appears.

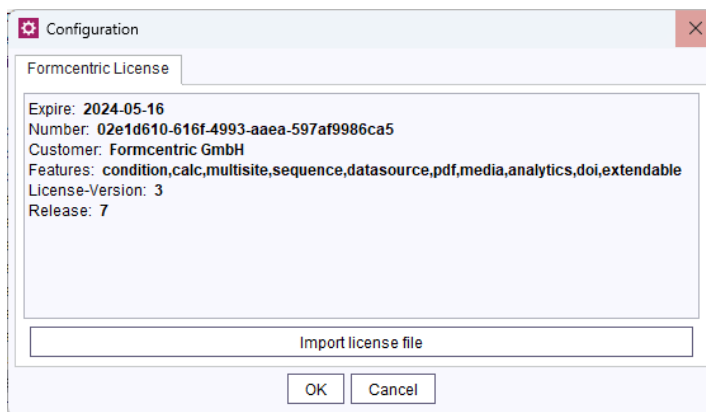


Figure 4.3. Configuring the licence file

Close the configuration dialog box by clicking *OK*.

To ensure that the licence file is also available in the Formcentric web applications, these applications must be updated after each licence update.

The licence file is typically valid for 12 months. During the last 30 days before the licence expires, a warning is output to the Formcentric web application log every six hours. **No other warning is given.** The execution of form actions stops instantly as soon as the licence expires. The forms will continue to be shown correctly on the website, however.

4.3. Installing the Formcentric web editor

For creating and editing forms in the FirstSpirit ContentCreator, Formcentric provides a ContentCreator extension.

To install this extension, switch to the *Web applications* section.

In this area, add the Formcentric web editor to the existing *ContentCreator* web application by clicking *Add*. In the following selection dialog, all of the available web components are displayed. Select the *Formcentric WebEditor* entry.

Close the selection dialog box by clicking *OK*.

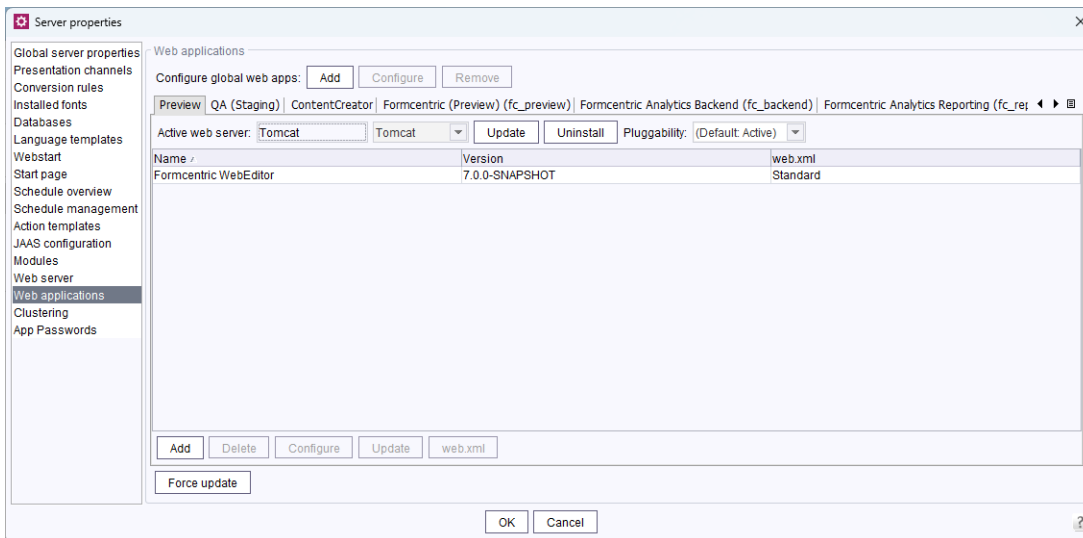


Figure 4.4. Installing the Formcentric web editor

You then need to update the ContentCreator web application on the web server.

4.4. Formcentric web applications

Formcentric provides two separate web app modules to handle form presentation and processing. One of these modules is a Spring MVC web application, while the other is a headless web application.

With the Spring MVC web application from Formcentric, data processing and rendering for forms is handled server-side based on Spring MVC plus JSP/FreeMarker templates. In contrast, the headless web application uses a modern, decentralised architecture that separates the backend (data processing) from the frontend (UI rendering). This application utilises Spring Boot for the backend and React for the frontend.

With the exception of the points discussed in section the section called “File upload tab”, configuration is the same for both web applications. The following section therefore describes configuration using the headless app as an example.

Typically, separate instances of the web applications are installed for the preview page and live page. For this reason, four global web applications are created during the first installation of the module: *fc_preview_headless* and *fc_live_headless* for the headless application, and *fc_preview* and *fc_live* for the Spring MVC web application. If you want to use different names, complete the manual steps given in section 4.4.1 for the affected web application.

4.4.1. Installation

Create a new global web application. This is required for form presentation within the web page.

You create a new web application by accessing the *Web applications* area and then clicking the *Add* button.

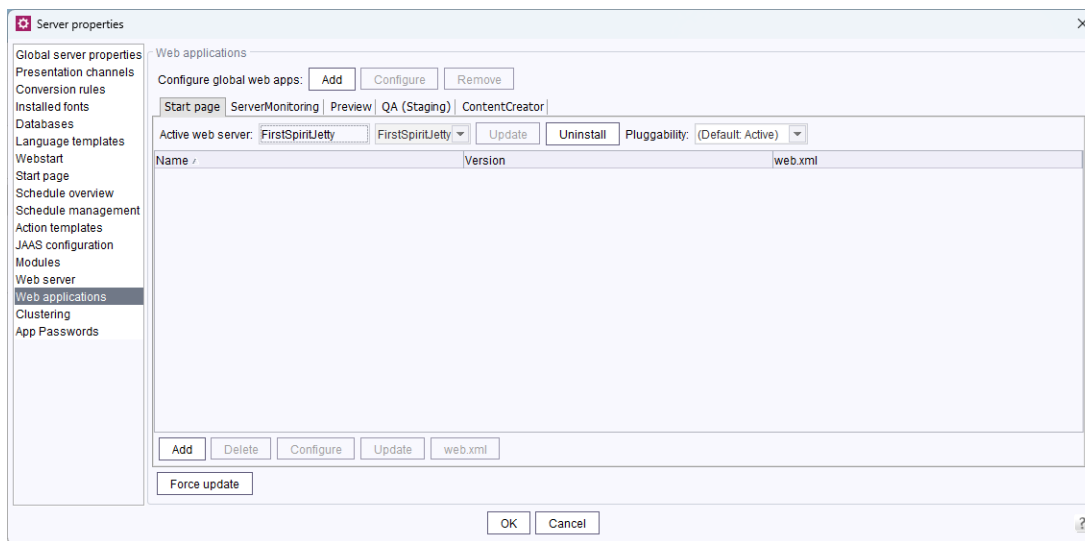


Figure 4.5. Adding a global web application

In the following dialog box, enter the web application's ID, name and context. Close the dialog box by clicking *OK*.

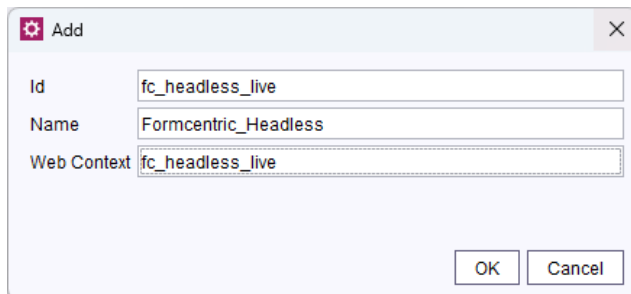


Figure 4.6. Adding a new web application



Typically, the following IDs are used: *fc_preview_headless* for the Preview application and *fc_live_headless* for the Live application. If you assign different IDs, you must also modify these in the script *formcentric_headless_url* (see Section 5.2, “*formcentric_headless_url* script”).

In the next step, you add the Formcentric web app module to the web application just created. Click *Add* and then select *Formcentric Headless WebApp*.

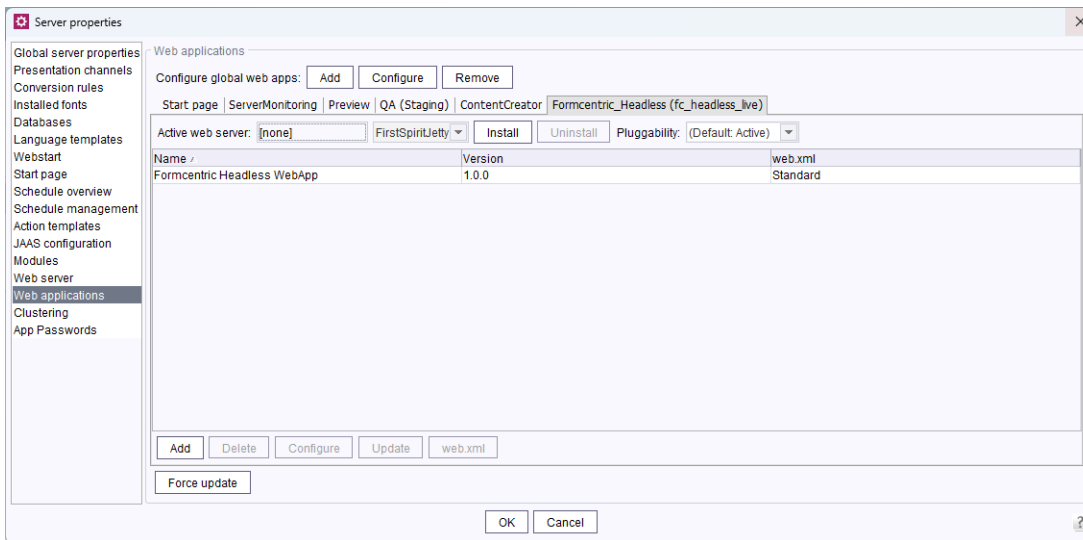


Figure 4.7. Global web application with installed web component

4.4.2. Configuration

Double-click the *Formcentric Headless WebApp* web component to open the configuration screen, which contains multiple tabs.

Mail server tab

You use the *Mail server* tab to configure the mail server used by the form extension to send its emails.

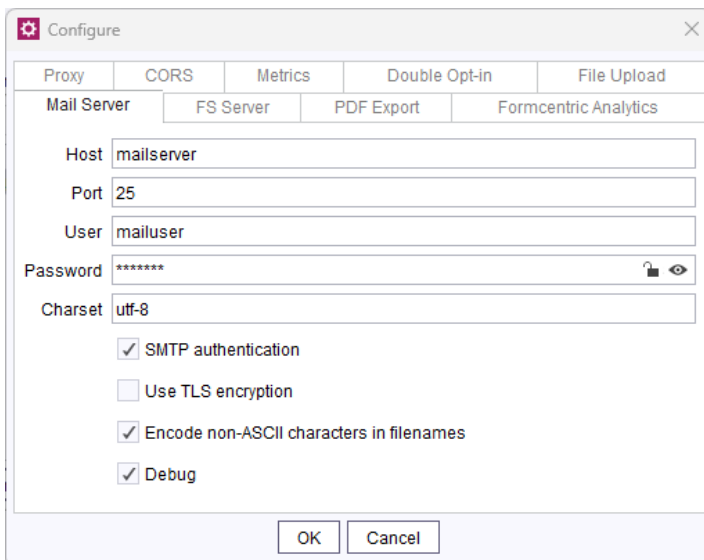


Figure 4.8. Configuring the web application on the “Mail server” tab

Host: Name or IP address of the SMTP mail server.

Port: Port number of the mail server.

User: The name used to log into the mail server.

Password: The password used to log into the mail server.

Charset: The character encoding in which emails are sent (such as utf-8, iso-8859-15, etc.).

SMTP authentication: By activating this check box, you specify that the login data (user, password) will be used when negotiating with the SMTP server.

Use TLS encryption: Configure this property if STARTTLS must be used.

Encode non-ASCII characters in filename: Check this box to encode all non-ASCII characters in the filename. Please note that this encoding does not comply with the MIME specification but is useful to maintain compatibility with some email clients that utilise this convention. The default value is TRUE.

Debug: By checking this check box, you specify that a detailed set of status information will be written to the log for every email sent. This information includes all of the body content, metadata and headers, which could contain personal data.

PDF export tab

The *PDF export* tab is used to configure global settings for the PDF action.

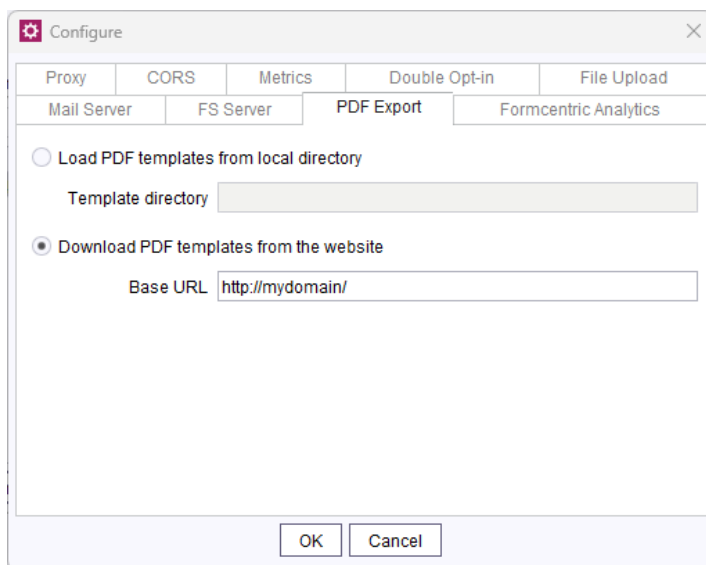


Figure 4.9. Configuring the web application on the “PDF export” tab

Load PDF templates from local directory: From version 5.5.0 of Formcentric, the PDF template documents are loaded by default directly from the Preview or Live web app. Select this option if you want to load the template documents from a local directory, as previously. In this case, you must publish the templates beforehand.

Template directory: In this field, you specify the path to the directory to which the PDF template documents are copied on publication (see also Section 4.9, “Configuring the publication tasks”). The web application must have access to this directory. The user account under which the web server runs also requires read permissions for the specified directory.

You can specify the template directory either as relative to the web app directory (for example *WEB-INF/pdf*) or as an absolute directory (for example */var/pdf* or *c:/var/pdf*). If the directory does not exist, then it is created when the application is started. In this case, the user must also have write permissions for the parent directory.

Download PDF templates from the website: Select this option if PDF template documents should be loaded from the Preview or Live web app. In this case, just as for other content, the documents must be published in the corresponding website.

Base URL: Enter the external address here at which the website is accessible.

FS server tab

On the *FS server* tab, you enter the connection data to your FirstSpirit server. These details are required for the *Data source* and *Media Management* actions.

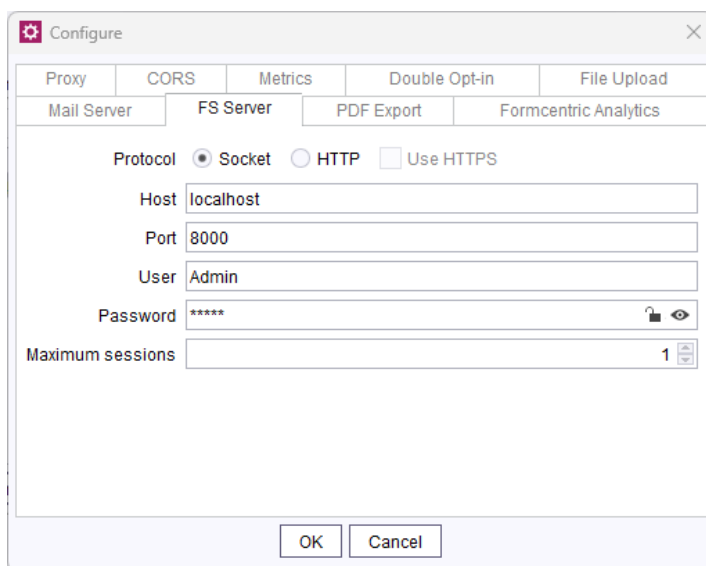


Figure 4.10. Configuring the web application on the “FS server” tab

Protocol: Here you select the communication protocol (HTTP or socket) used to communicate with the FirstSpirit server.

Use HTTPS: When using HTTP as the communication protocol, checking this check box activates encryption for all communication with the server.

Host: Name or IP address of the FirstSpirit server.

Port: Port number of the FirstSpirit server.

User: The name used to log into the FirstSpirit server.

Password: The password used to log into the FirstSpirit server.

No. of sessions: The maximum number of sessions that can be open simultaneously on the FirstSpirit server.



To establish a connection to the FirstSpirit server, the system needs the name of the host (hostname) from which the web app can access the server.

The web app must also be capable of establishing a connection to the port on the FirstSpirit server. In some circumstances, it may be necessary to configure access to this port if a firewall is used.



The maximum number of sessions will depend on your FirstSpirit licence. Please note that in some circumstances, the number of sessions specified on the tab page may no longer be available to your form authors.

Formcentric Analytics tab

You use the *Formcentric Analytics* tab to configure the connection parameters to the Formcentric Analytics Backend.

The screenshot shows a 'Configure' dialog box with the 'Formcentric Analytics' tab selected. The 'Backend URL' is set to 'http://localhost:8080/fc_backend'. The 'Backend authentication' field is masked with asterisks. The 'Collect personal metadata' checkbox is checked. The dialog has 'OK' and 'Cancel' buttons at the bottom.

Figure 4.11. Web application configuration on the “Formcentric Analytics” tab

Analytics Backend URL: URL at which the Formcentric Analytics Backend can be reached.

Backend authentication: Enter the secret used to log into the Analytics Backend here. This must match the *Backend client secret* that you issued during the configuration of the Analytics Backend (see the section called “Security tab”).

Collect personal metadata: Check this box to store the following metadata in addition to the form data entered: information about the browser being used (*User-Agent*), the configured browser language (*Language*) and the page visited before this page (*Referer*). In some cases, this metadata can be used to associate separate and self-contained data records with a specific individual.



Instead of the client secret, a pre-generated *access token* can also be used. To do so, you will need to modify the Spring configuration, however (see the section called “formcentric-analytics.xml”).

Proxy tab

PDF template documents are loaded by default directly from the Preview or Live web app. If access to the web applications should be made via a web proxy, you must configure this proxy on the *Proxy* tab.

The image shows a 'Configure' dialog box with a 'Proxy' tab selected. The dialog has a title bar with a gear icon and a close button. Below the title bar are four tabs: 'Mail Server', 'FS Server', 'PDF Export', and 'Formcentric Analytics'. The 'Proxy' tab is active, showing several sub-tabs: 'CORS', 'Metrics', 'Double Opt-in', and 'File Upload'. The main content area has two radio buttons: 'No proxy' (unselected) and 'Manual proxy configuration' (selected). Below the 'Manual proxy configuration' radio button are four text input fields: 'HTTP Proxy' with the value 'proxyhost', 'Port' with the value '80', 'Username' with the value 'myuser', and 'Password' with the value '*****'. Below these fields is another radio button: 'Automatic proxy configuration URL' (unselected). Below this radio button is a text input field labeled 'URL'. At the bottom of the dialog are two buttons: 'OK' and 'Cancel'.

Figure 4.12. Configuring a web proxy on the “Proxy” tab

No proxy: Activate this setting if you do not want to use a proxy.

Manual proxy configuration: Select this setting if you want to enter the proxy settings manually.

HTTP proxy: Host name of the proxy server that is to be used.

Port: The proxy server port.

Username: The username used to log into the proxy server.

Password: The password used to log into the proxy server.

Automatic proxy configuration: Select this option if you want to use a proxy configuration file (.pac) instead of configuring the proxy manually.

URL: Enter the address (URL) here from which the configuration file should be loaded.

CORS tab

As explained elsewhere, the forms are generated by a dedicated web app component and embedded dynamically into the surrounding web page. This web app is accessed by using an asynchronous JavaScript request (AJAX).

In a scenario where the forms web app is hosted on a separate (sub)domain (e.g. <http://forms.mydomain.com>), access requests to the form will be viewed as ‘cross-

origin requests'. These kinds of requests are normally prohibited by the 'same-origin policy' (SOP) and therefore blocked by modern browsers. However, these restrictions can be removed by setting access control headers for certain clients.

The *CORS* tab gives you the option of configuring these access control headers to suit your individual requirements.

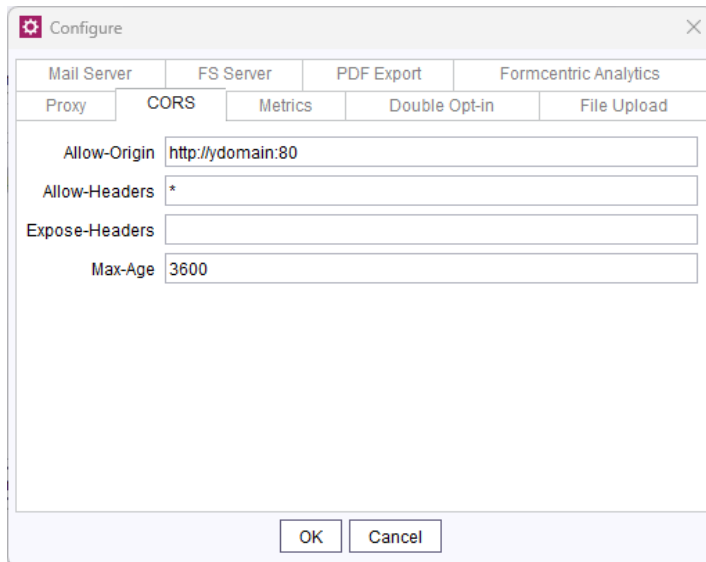


Figure 4.13. Configuring cross-origin resource sharing (CORS)

Allow-Origin: Enter a comma-separated list of domains here from which access to the form web app will be permitted (e.g. `www.mydomain.com, www.another-domain.com`). You can specify that access is permitted from any domain by entering an asterisk `"*"`. This is the default setting.

Allow-Headers: You use this parameter to specify the HTTP headers that are permitted to be passed across domain boundaries. You can specify that all HTTP headers are permitted by entering an asterisk `"*"`. This is the default setting.

Expose-Headers: You use this parameter to specify the HTTP headers that are permitted to be included in the server reply. The HTTP headers *X-Redirect-Location* and *X-Redirect-Delay* are evaluated by the Formcentric jQuery plugin and are therefore always permitted.

Max-Age: You use this parameter to specify the period of validity for the information from a pre-flight request.

If you operate the form web app under a separate domain, as described, then the application generates fully-qualified URLs in the HTML output automatically. The base URL that is used here is the base URL accessed by the browser, e.g. `https://forms.mydomain.com/`.

If the form application is accessed via an intervening load balancer or reverse proxy, the HTTP request will only receive information about the connection from the load

balancer to the application. The load balancer must set additional HTTP request headers in order to forward the connection data originally used by the accessing browser – such as host, port, etc. – to the application. Among other things, the application needs these headers in order to generate fully-qualified URLs.

The following headers must be added:

HTTP request header	Description
X-Forwarded-Proto	Protocol (HTTP or HTTPS) that the browser has used to make the connection to your load balancer.
X-Forwarded-Host	Host that the browser has used to make the connection to the load balancer.
X-Forwarded-Port	Port that the browser has used to make the connection to the load balancer.
X-Forwarded-For	IP address of the accessing browser

Metrics tab

Formcentric provides you with a range of metrics for monitoring system values that have relevance for operations like storage usage or processor load for the Live web app. You can access these by using the URL `/fc_live/servlet/secure/health`. The metrics are supplied in a standardised text format that can be processed by the *Prometheus* monitoring system.

You can access metrics about form usage at the URL `/fc_live/servlet/secure/usage`.

Access to the metrics is secured with a login (basic authentication). You specify the associated login credentials on the *Metrics* tab.

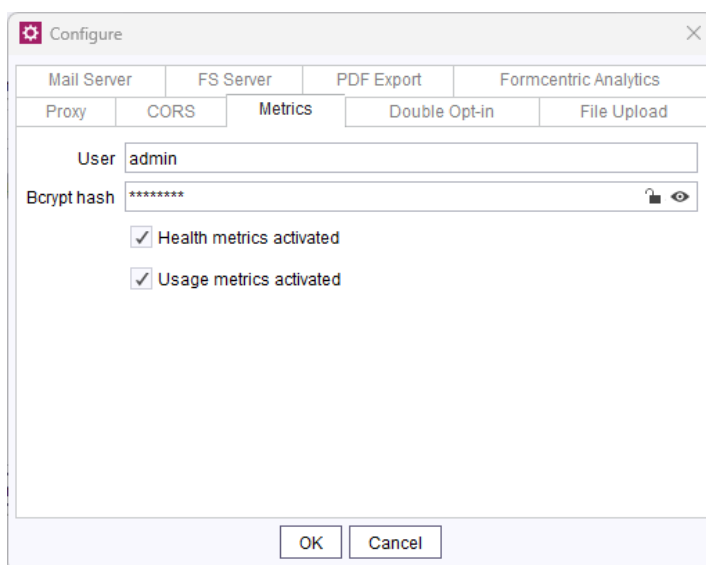


Figure 4.14. Configuring the credentials for displaying the metrics

User: Name of the user who will be granted access to the metrics.

Bcrypt hash: Enter the bcrypt hash of your password here. To generate a bcrypt hash, you can use the online service <https://bcrypt-generator.com>, for example.

System metrics activated: Activates or deactivates the endpoint for system metrics.

Usage metrics activated: Activates or deactivates the endpoint for usage metrics.

Double opt-in tab

When the double opt-in option is used, the user is sent a mail with a confirmation link. The link directs the user to the website on which this user completed and submitted the form. To prevent a situation where the link could be manipulated to redirect the user to a phishing or spam site, the URL entered must be verified. Link verification is completed using the URL pattern as configured in this dialog.

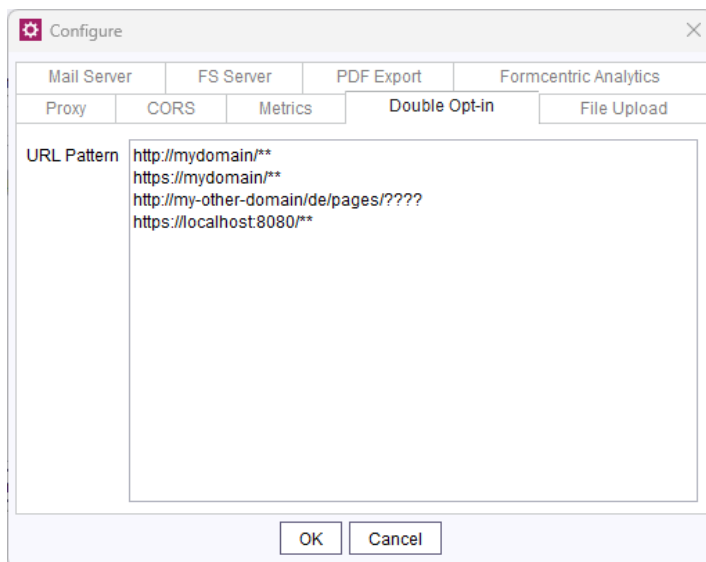


Figure 4.15. Configuration of the double opt-in URL pattern

URL pattern: In this input field, enter a URL pattern for each web page on which a Formcentric form can be embedded. Within the URL pattern, you can use the following wildcards:

1. ? stands for one character
2. * stands for one or more characters
3. ** stands for one or more directories in the path

File upload tab

Files that a user uploads in a form are cached temporarily on the server until form submission is complete. You use this configuration dialog to specify the storage location and maximum permitted size for file uploads.

For the cached storage, you can either use a local directory or MinIO, or an S3-compatible object storage service. MinIO is specially designed for the storage of large, unstructured data and optimised to handle high I/O loads. MinIO also offers strong security features, including server-side and client-side encryption, and can be operated both on-prem as well as in the cloud. Deploying MinIO is especially recommended for installations using multiple instances of the headless web application.



You can only use this configuration dialog for configuring the headless web application.

The screenshot shows a 'Configure' dialog box with several tabs. The 'File Upload' tab is selected. It contains sections for 'Size restrictions' and 'Cache'. Under 'Size restrictions', there are two input fields: 'Maximum file size' and 'Maximum request size', both set to '50MB'. Under 'Cache', there are two radio buttons: 'File system' and 'MinIO object storage'. The 'MinIO object storage' radio button is selected. Below it, there are four input fields: 'MinIO URL' (http://127.0.0.1:9999), 'MinIO bucket' (com.formcentric.headless.upload), 'Access key' (masked with asterisks), and 'Secret key' (masked with asterisks). At the bottom of the dialog are 'OK' and 'Cancel' buttons.

Figure 4.16. Configuring file uploads

Maximum file size: This parameter specifies the maximum size that an uploaded file can have. This is used as a security precaution to avoid large files overloading available server capacity. This setting is a technical limit, so it cannot be overridden by any editorial rules that apply to uploaded content.



For the Spring MVC web application, the maximum file size can be specified in the *web.xml* file.

Maximum request size: This parameter specifies the maximum size that an HTTP request can have. This applies to the size of the uploaded file(s) as well as all associated metadata and headers. As with the maximum file size, this parameter is also intended to reduce server load and improve performance.

File system: This parameter specifies that the uploaded files are cached on a local file system or connected network drive.

Directory: If you have selected *file system* as the cache location, you must specify the exact storage location here. This should be a path to a directory to which the web application has read and write access. You can also use the environment variable `{java.io.tmpdir}` in this field if you want to use the temporary directory from the Java VM.

MinIO Object Storage: This parameter activates the use of MinIO for the cache. Once activated, the corresponding MinIO parameters must also be configured.

MinIO URL: This parameter specifies the URL at which your MinIO storage service is reachable. The URL must include the protocol, hostname and port (if applicable).

MinIO Bucket: The *MinIO Bucket* parameter states the name of the specific container in your MinIO storage that is used to store the uploaded files. In MinIO, a *bucket* is essentially the equivalent of a folder in a conventional file system.

Access Key: The *Access Key* forms part of your login credentials for the MinIO storage. This is used together with the *Secret Key* for MinIO authentication. The access key should be stored securely and must not be generally accessible.

Secret Key: The *Secret Key* is the counterpart to the *Access Key* and is used together with this key for MinIO authentication. As with the access key, the secret key must be handled as strictly confidential and kept in a secure location.

Repeat the steps in sections 4.4.1 and 4.4.2 for the Preview web application.

4.5. Analytics Backend web application

4.5.1. Installation

Create another global web application. This is required to store and process your forms with Formcentric Analytics.

Enter the ID, the name and the context for the web application. Typically, the ID *fc_backend* is used for the Analytics Backend application. Close the dialog box by clicking *OK*.

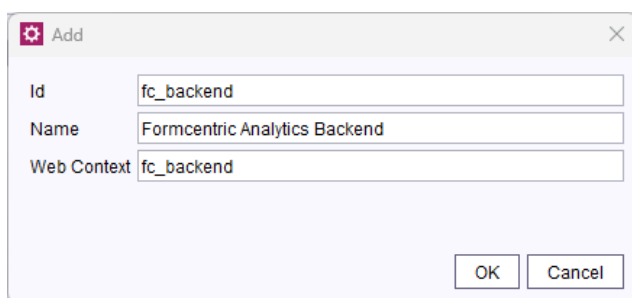


Figure 4.17. Adding a new web application

Add a web component to the web application that you have just created by clicking *Add*. In the following selection dialog, all of the available web components are displayed. Select the *Formcentric Analytics Backend* entry.

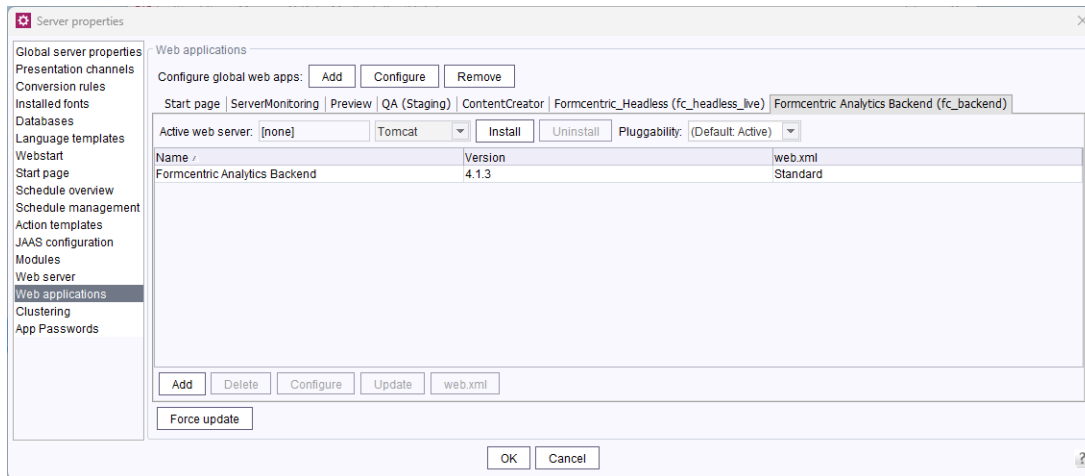


Figure 4.18. Global web application with installed web component

4.5.2. Configuration

Double-click the *Formcentric Analytics Backend* web component to open the configuration dialog, which contains multiple tabs.

General tab

You use the 'General' tab to configure general settings.

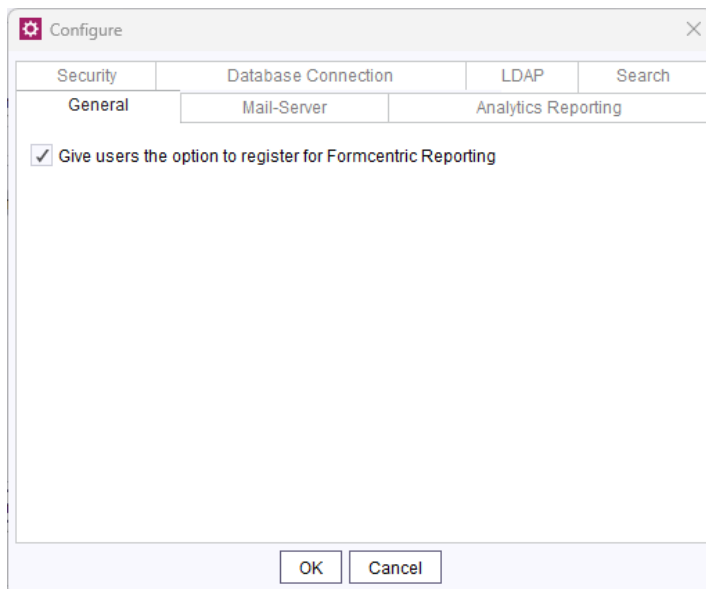
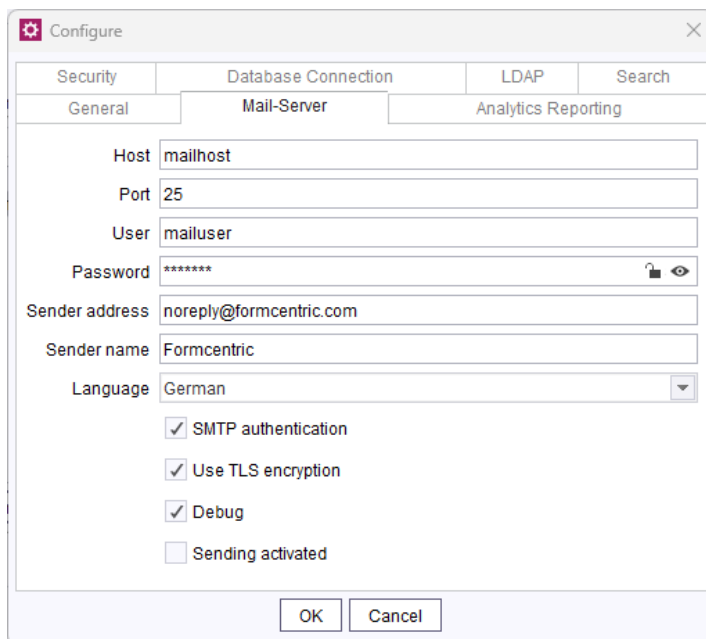


Figure 4.19. Configuring the Analytics Backend web application on the "General" tab

Giving users the opportunity to register for Reporting: Check this check box to specify that a link to the registration screen should be shown on the Formcentric Reporting interface login screen. On the registration screen, users can register for access to Formcentric Analytics (see also section 2, „Login and registration”, in the Formcentric Analytics User Manual).

Mail server tab

You use the 'Mail server' tab to configure the mail server that the Analytics Backend uses to send emails to users.



The screenshot shows a 'Configure' dialog box with the 'Mail-Server' tab selected. The fields are as follows:

- Host: mailhost
- Port: 25
- User: mailuser
- Password: masked with asterisks
- Sender address: noreply@formcentric.com
- Sender name: Formcentric
- Language: German
- SMTP authentication: checked
- Use TLS encryption: checked
- Debug: checked
- Sending activated: unchecked

Figure 4.20. Configuring the Analytics Backend web application on the “Mail server” tab

Host: Name or IP address of the SMTP mail server.

Port: Port number of the mail server.

User: The account used by Formcentric Analytics to authenticate with the SMTP server.

Password: Password for the SMTP user.

Sender address: The email address that Formcentric Analytics uses as the sender when sending email.

Sender name: This property is used to set the name that is shown as the sender for all mail sent by Formcentric Analytics.

Language: Configure this property to set the language used for sending email.

SMTP authentication: Use this property to activate/deactivate use of the AUTH command by the mail user.

Use TLS encryption: Configure this property if STARTTLS must be used.

Debug: Use this property to activate/deactivate extra debugging output while mail is being sent. This information includes all of the body content, metadata and headers, which could contain personal data.

Send activated: Use this property to activate or deactivate the sending of email from Formcentric Analytics.

Analytics Reporting tab

On the *Analytics Reporting* tab, you configure the external URL from which the Reporting interface can be accessed.

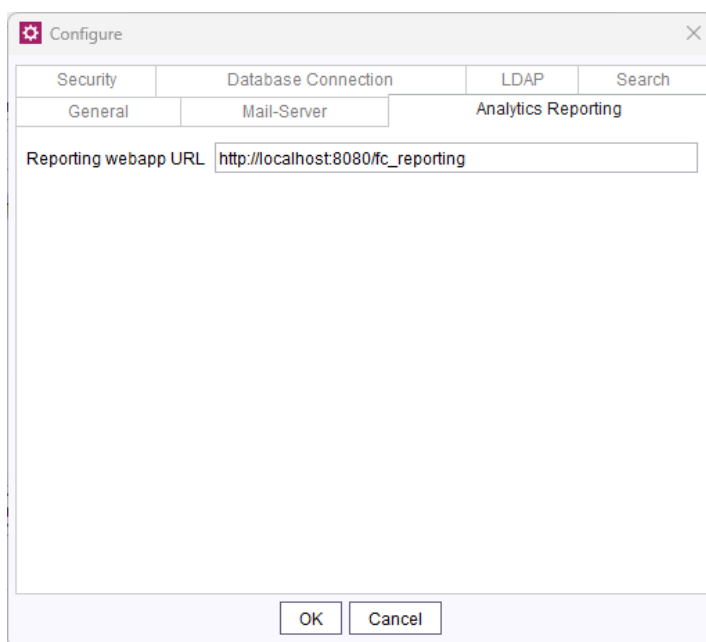


Figure 4.21. Configuring the Analytics Backend web application on the “Analytics Reporting” tab

Reporting web app URL: External URL from which the Reporting web application is reachable.

Security tab

You use the ‘Security’ tab to enter the login credentials for logging into the Analytics Backend.

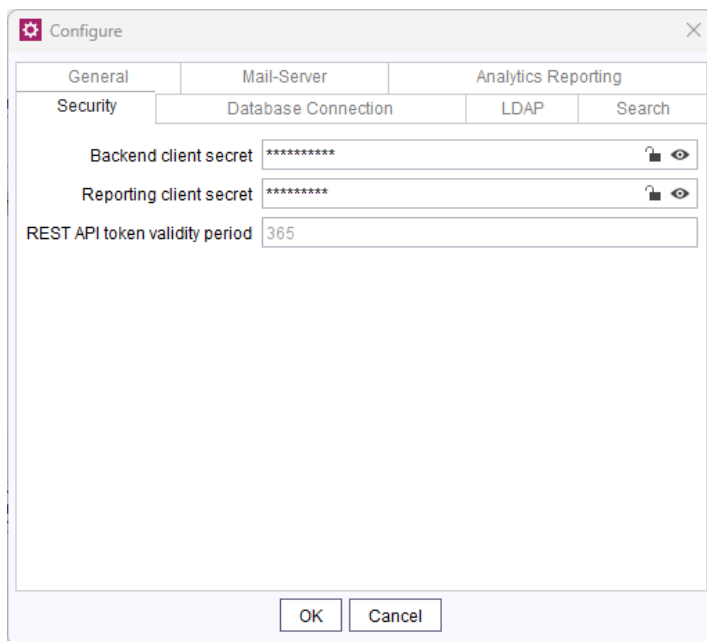


Figure 4.22. Configuring the Analytics Backend web application on the “Security” tab

Backend client secret: Enter your chosen secret here. This will be used by the Formcentric web application to generate an *access token* for the Analytics Backend. The value chosen should be as secure as possible.

Reporting client secret: Enter your chosen password here. This is used for Reporting application authorisation on the Analytics Backend.

API token validity period: Here, enter the number of days for which a REST API token will be valid.

Database connection tab

The database connection tab is used to configure the database connection for the Analytics Backend database. The necessary database tables and indexes are either automatically created or updated when the application is started.

Formcentric uses the Java Database Connectivity API (JDBC) for accessing the Backend database. The JDBC drivers required are not included in the Formcentric FSM module. You therefore need to copy the JDBC driver for the database system used onto the web server, so that the driver is found in the application's classpath. A detailed guide to installing and configuring the Backend web application can be found in the Installation Manual (*formcentric_backend_install_en.pdf*).

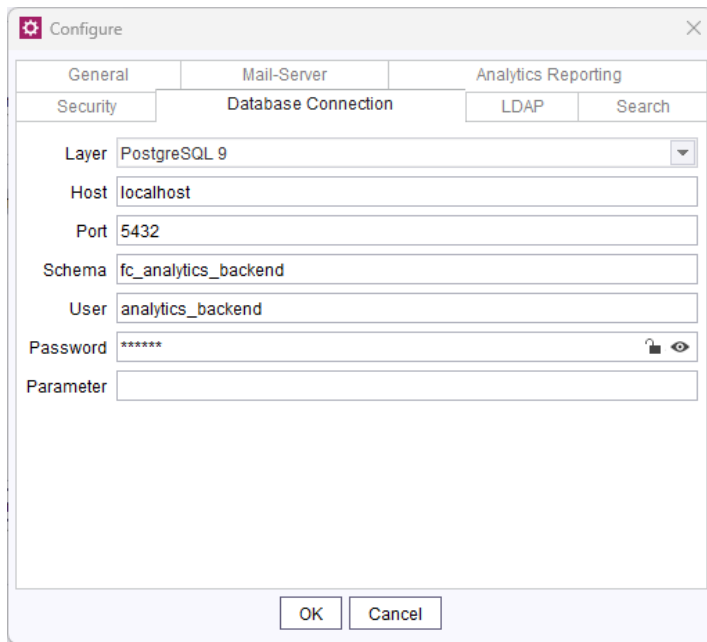


Figure 4.23. Configuring the Analytics Backend web application on the “Database connection” tab

Layer: Select the appropriate driver for your database here.

Host: Enter the name or the IP address of your database server.

Port: Enter the port that the client should use to connect to your database server.

Schema: Enter the database schema that will be used.

User: Enter the username that the Formcentric Analytics Backend uses to log into your database.

Password: Enter the password for the user here.

Parameters: This field gives you the option of specifying additional connection parameters specific to your database. These will be added to the connection URL exactly as you have entered them. The parameter string must begin with the separator that is used by the database system deployed (e.g. `?ssl=true`).

LDAP tab

In addition to the internal user management functionality offered by Formcentric Analytics, you can also integrate an external user directory such as LDAP or Active Directory. You configure the connection and filter settings for your user directory on the ‘LDAP’ tab.

Figure 4.24. Configuring the Analytics Backend web application on the “LDAP” tab

Directory type: Select the type of external user directory that you want to connect to here. Formcentric supports LDAP and Active Directory.

URL: URL of the LDAP server.

Username: Username for access to the LDAP server.

Password: Password for access to the LDAP server.

Base DN: The Distinguished Name (DN) of the base directory that is used to hold the user and group objects (e.g. *dc=mydomain,dc=com*).

Subdomain DNs: Optional specification of subdomains that hold additional user and group objects (e.g. *dc=subdomain1,dc=my-domain,dc=com*). For each additional subdomain, enter a new row with the fully-qualified DN of the subdomain.

Domain name: Human-readable name of the domain used as a default for the user during registration.

Search base for users: Specifies an object in the directory tree under which the user search is executed (e.g. *ou=people* for LDAP or *cn=users* for Active Directory).

Search filter for users: Specifies an LDAP search filter to apply to the user search (which is run using the specified search base) (e.g. *(uid={0})* for LDAP or *(samaccountname={0})* for Active Directory).

The placeholder *{0}* is replaced with the username entered before executing the search.

A general description of the search filter syntax is available from the following link:<http://www.faqs.org/rfcs/rfc2254.html>.

User DN pattern: Pattern that is used to generate the distinguished name (DN) for a user (e.g. *uid={0},ou=people*).

User email: Name of the user object attribute field in which the email address can be found. The email address is saved in the Analytics database.

Search base for groups: Specifies an object in the directory tree under which the group search is executed (e.g. *DC=company,DC=com* for LDAP or *cn=users* for Active Directory).

Search filter for groups: Specifies an LDAP search filter to apply to the group search (which is run using the specified search base) (e.g. *(&(objectclass=groupOfUniqueNames))* for LDAP or *(&(objectclass=group))* for Active Directory).

Groups import filter: All of the groups for which specific permissions will be granted within Formcentric Analytics must first be imported into the internal user directory used by Formcentric Analytics. You can use the *group import filter* field to specify an LDAP search filter that will be applied in order to select the LDAP groups to import from the list of available groups (e.g. *(cn=*AnalyticsUsers*)*). If you do not enter anything into this field, then all of the groups identified by the LDAP search filter field *search filter for groups* will be imported.

Search filter for user groups: This LDAP search filter is used to identify the groups in which a user is a member. You only need to specify this parameter if you have selected the *LDAP* directory type (e.g. *(memberUid={0})*). The placeholder *{0}* is replaced with the username entered before executing the search.

If you want to install the Analytics Backend web application on the *FirstSpirit Jetty Server*, you need to extend the *web.xml* for the web application to include the context parameter *webAppRootKey*. Enter the web application ID as the value.

```
<context-param>
  <param-name>webAppRootKey</param-name>
  <param-value>mwf_backend</param-value>
</context-param>
```

Search tab

The enterprise search engine *Solr* is required for the global full-text form search. Typically, the Solr search engine is run as a self-contained application. For information about installing and configuring Solr, please see the product page <http://lucene.apache.org/solr/>.

To use an external Solr instance with Formcentric Analytics, you first need to access the Solr administration interface and create a new core with the name *mwfanalytics*.

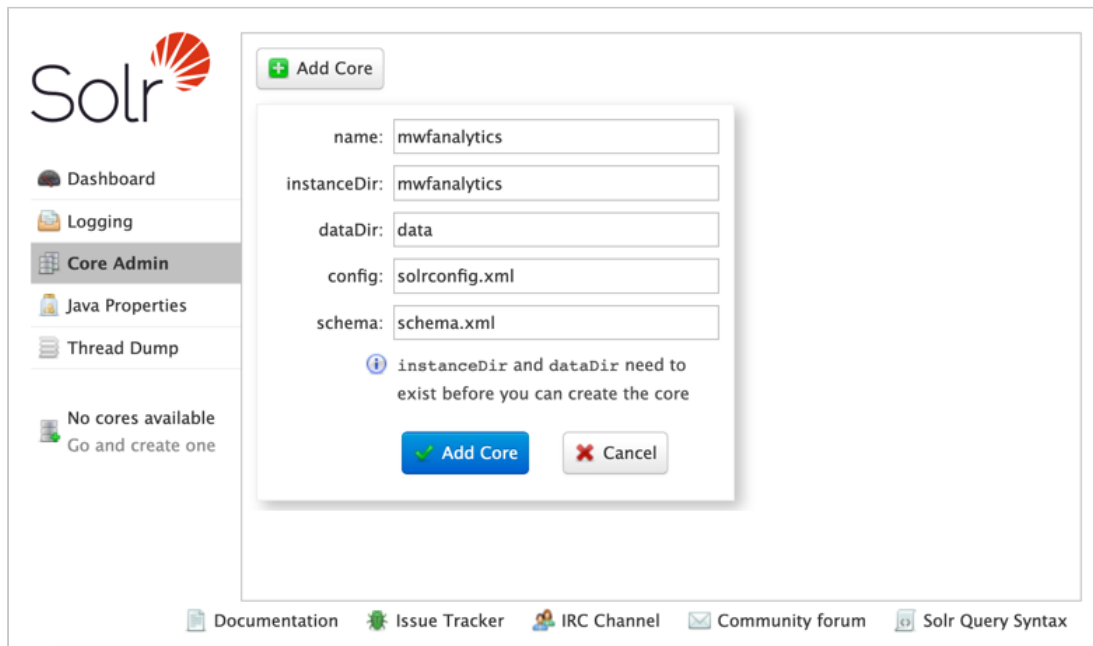


Figure 4.25. Core administration in the Solr administration interface

The required configuration files *solrconfig.xml* and *schema.xml* can be found in the archive file *webforms-backend-solr-config-4.3.1.tar*.

You then configure the Solr search engine on the *Search* tab.

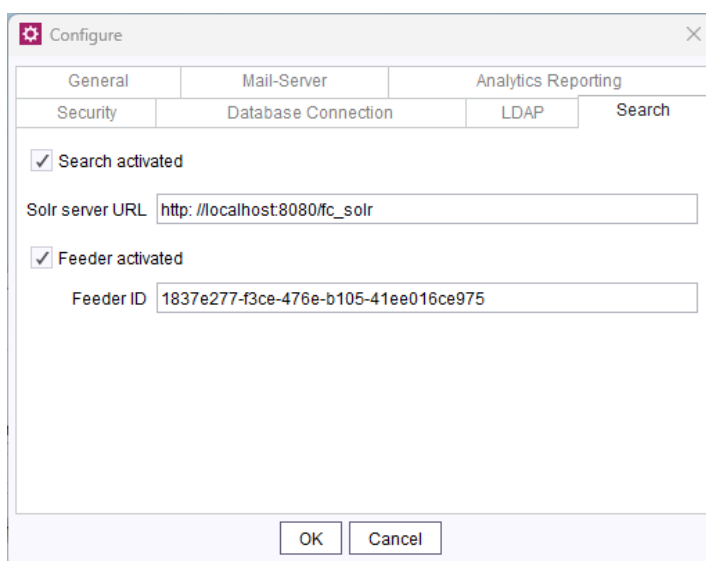


Figure 4.26. Configuring the Solr search engine on the “Search” tab

Search activated: Activates or deactivates the full-text search used in all forms.

Solr server URL: Enter the Solr server URL here.

Feeder activated: Activates or deactivates the feeder that is used to populate the search index. If you are running multiple instances of the Backend application, it is usually adequate to have the search index populated by just one Backend application.

Feeder ID: Assign a unique ID to the feeder here. An arbitrary character string can be used here. If multiple Backend applications are being run in parallel, the feeder IDs for the various Backend applications must be unique.



Instead of running Solr as a standalone application, you also have the option of installing Solr as a global FirstSpirit web app. Formcentric will provide you with a separate FirstSpirit module for this purpose. To install this module, please see the installation guide at Section 4.7, “Solr web application”.

In this scenario, enter the URL of the global web app as the value for the *Solr server URL* parameter.

Repeat the steps in sections 4.5.1 and 4.5.2 if you want to use separate instances of the Analytics Backend for the Preview and Live environments.

4.6. Analytics Reporting web application

4.6.1. Installation

Create another global web application for the Formcentric Analytics Reporting application.

Enter the ID, the name and the context for the web application. Typically, the ID *fc_reporting* is used for the Reporting web application. Close the dialog box by clicking *OK*.

A screenshot of a dialog box titled "Add" with a close button (X) in the top right corner. The dialog contains three text input fields: "Id" with the value "fc_reporting", "Name" with the value "Formcentric Analytics Reporting", and "Web Context" with the value "fc_reporting". At the bottom right of the dialog are two buttons: "OK" and "Cancel".

Figure 4.27. Adding the Reporting web application

Add a web component to the web application that you have just created by clicking *Add*. In the following selection dialog, all of the available web components are displayed. Now select the *Formcentric Analytics Reporting* entry.

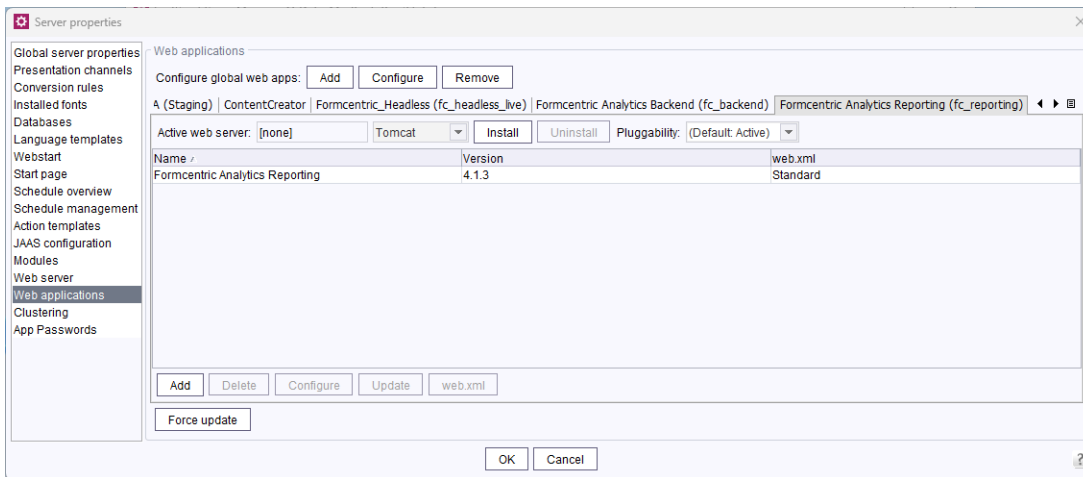


Figure 4.28. Global web application with installed web component

4.6.2. Configuration

Double-click the *Formcentric Analytics Reporting* web component to open the configuration dialog.

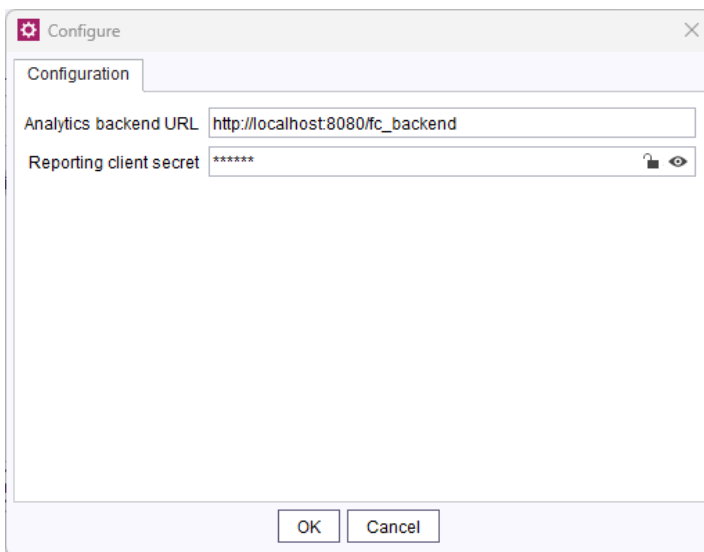


Figure 4.29. Configuring the web application on the “Configuration” tab

Analytics Backend URL: URL for accessing the Formcentric Analytics Backend. If you are using a separate Backend for Live and Preview in your environment, then take care to ensure that you specify the correct URL here.

Reporting client secret: Enter the password used to log into the Analytics Backend here. This must match the *Reporting client secret* that you issued during the configuration of the Analytics Backend (see the section called “Security tab”).

If you want to install the Reporting web application on the *FirstSpirit Jetty Server*, you need to extend the *web.xml* for the web application to include the context parameter *webAppRootKey*. Enter the web application ID as the value.

```

<context-param>
  <param-name>webAppRootKey</param-name>
  <param-value>fc_reporting</param-value>
</context-param>

```

Repeat the steps in sections 4.6.1 and 4.6.2 if you want to use separate instances of the Reporting application for the Preview and Live environments.

4.7. Solr web application

Formcentric provides the Solr search engine as a separate FirstSpirit module for test and development purposes. This can be downloaded from our FTP server.

You install the *solr-module-1.4.0.fsm* module file using FirstSpirit's Server and Project Configuration.

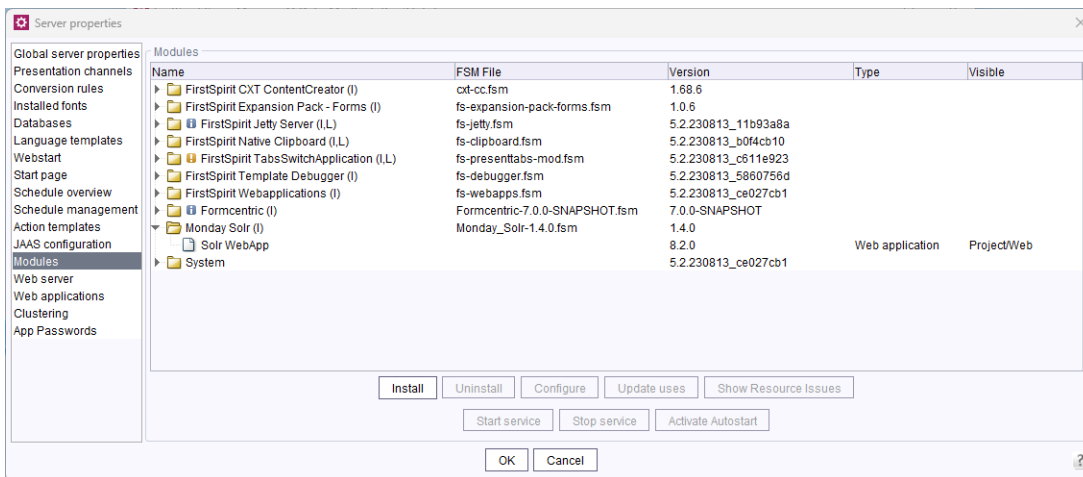


Figure 4.30. Module overview

As a final step, create another global web application.

Enter the ID, the name and the context for the web application. Typically, the ID *fc_solr* is used for the Solr application. Close the dialog box by clicking *OK*.

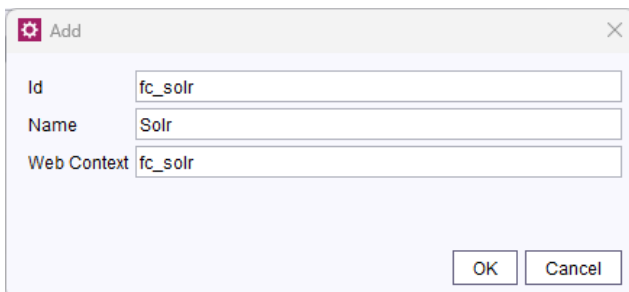


Figure 4.31. Adding a new web application

Add the following web components to the web application that you have just created by clicking *Add*. In the following selection dialog, all of the available web components

are displayed. Select the entries *Solr WebApp* and *Monday Webforms Analytics Solr Core*.

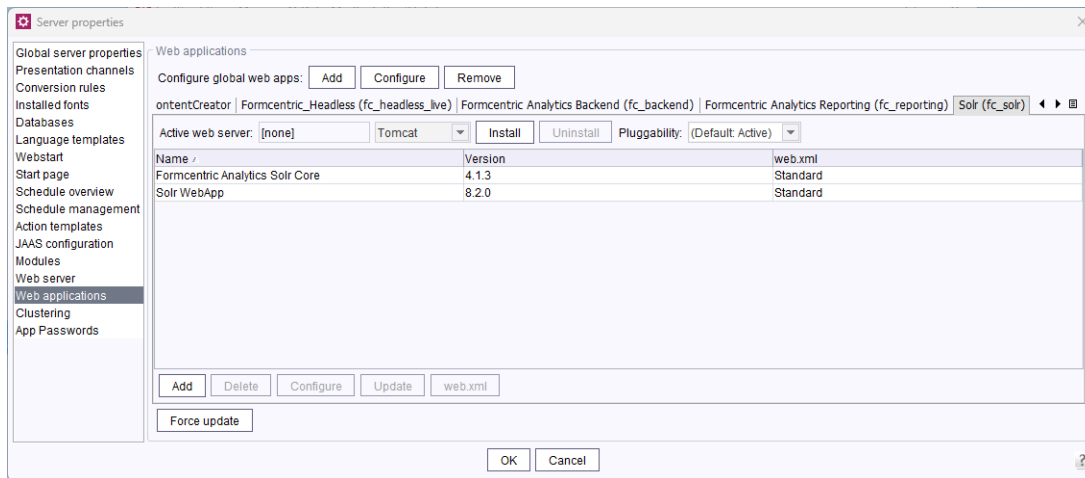


Figure 4.32. Solr web application with installed web components

Install the web application on the web server.

4.8. Installing the project component

To integrate the form extension into a new or existing project, you will need some additional resources (JavaScripts, paragraph style sheets and CSS). These are bundled together as a separate project component, which you can add to your project with the help of the Project Configuration screen.

To do so, open the Project Configuration screen by double-clicking your project in the Project Overview, and select the *Project components* option from the menu on the left. Click the *Add* button to open up a selection dialog box that displays all of the project components installed on the server. Select the *Formcentric resources* entry.

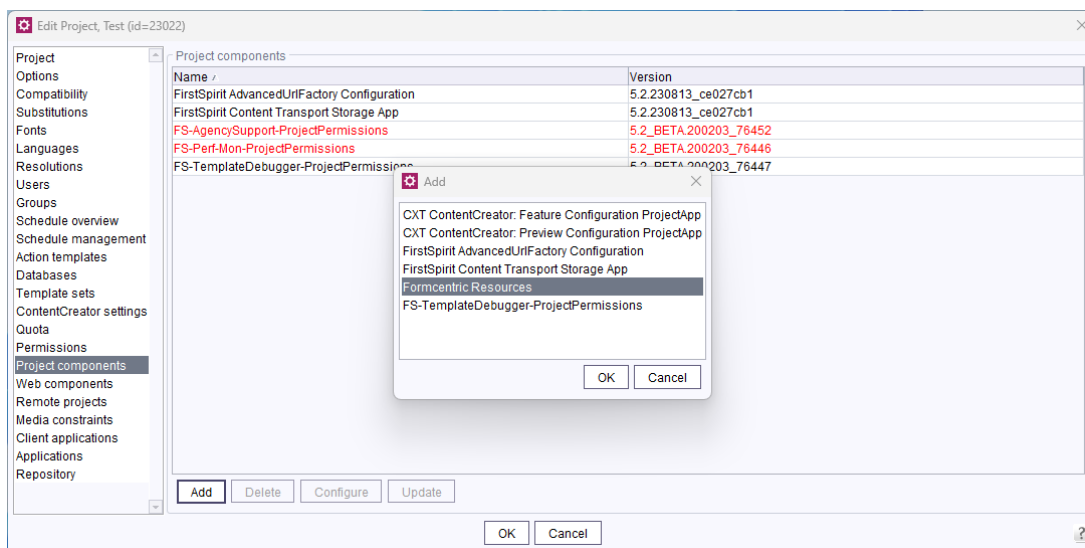


Figure 4.33. Project components in the project properties

Following a successful installation, the project is now extended by the following elements:

- **Paragraph style sheets:** The new style sheets *Form* and *Headless form* have been created in the project's paragraph style sheets.
- **Scripts:** The *Scripts* area in Template Management has now been extended by the *Formcentric* folder, which contains various generation scripts.
- **Media:** Media Management now contains the folders *Formcentric* and *Formcentric Headless*. These contain the required JavaScripts and style sheets in their original and minified forms.



If you remove the project component from the project again, then the resources listed above are also removed automatically. Note that ALL changes that you have made in this area after installing the module will be lost!

When the module is updated to a new version, the project component also updates itself automatically. In the process, the system must also replace the paragraph style sheets (see Section 5.1, "Paragraph style sheet") with new versions. This leads to a situation where you can no longer edit existing forms following the update, since the system can no longer find the associated paragraph style sheet. To avoid this problem, you should work with copies of the paragraph style sheets contained in the project component.

Please note: You can use the *fix_ReferenceNotFoundException* script to repair forms whose paragraph style sheet has been deleted.

4.9. Configuring the publication tasks

You need to perform the configuration of the publication tasks as described below only if the PDF template documents should be loaded from a local directory within the Formcentric web app (see also the section called "PDF export tab").

For the PDF action to work properly, you need to publish your PDF templates to the Formcentric web application. This publication is necessary both for the production web application and for the Preview. The external program *rsync* is used for the publication.

For information about installing and configuring *rsync* in conjunction with FirstSpirit, please see section 10 of the FirstSpirit "Administrator Documentation".

A BeanShell script is provided for publishing the PDF template. The script is supplied to you as part of the Formcentric scope of delivery.

Open the Project Configuration screen by double-clicking your project in the Project Overview, and select the *Action templates* option from the menu on the left. Click the

Add button to open up a selection dialog box that displays various activities. Select the *Execute script* entry.

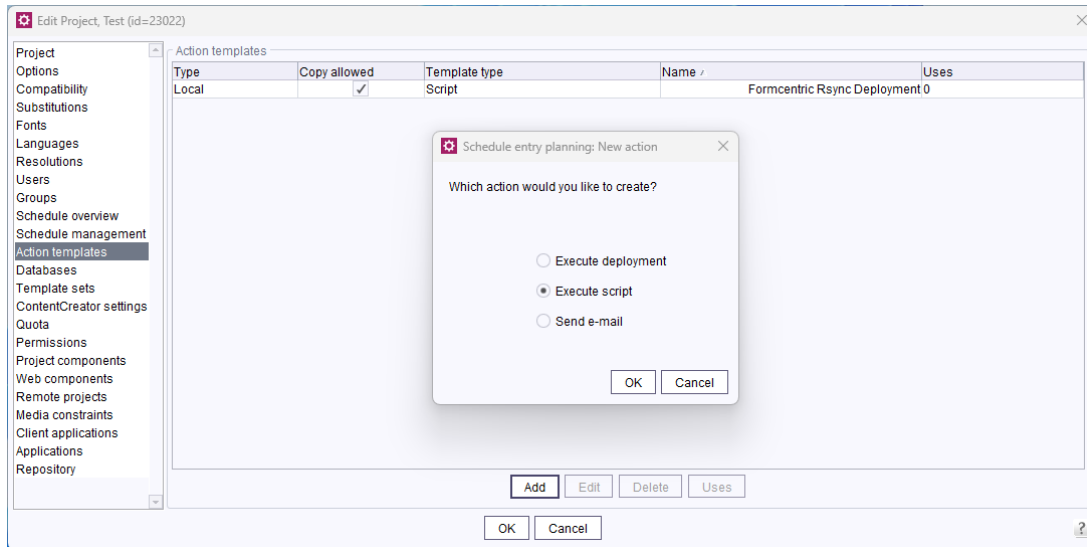



Figure 4.34. Creating a schedule template in Server Manager

In the *Script* dialog box that then appears, enter the name “Formcentric RSync Deployment”.

 You can use a different name, but you will then have to modify the script in the Java client.

Copy the contents of the script provided (*ext-rsync-ssh.bsh*) into the input field below.

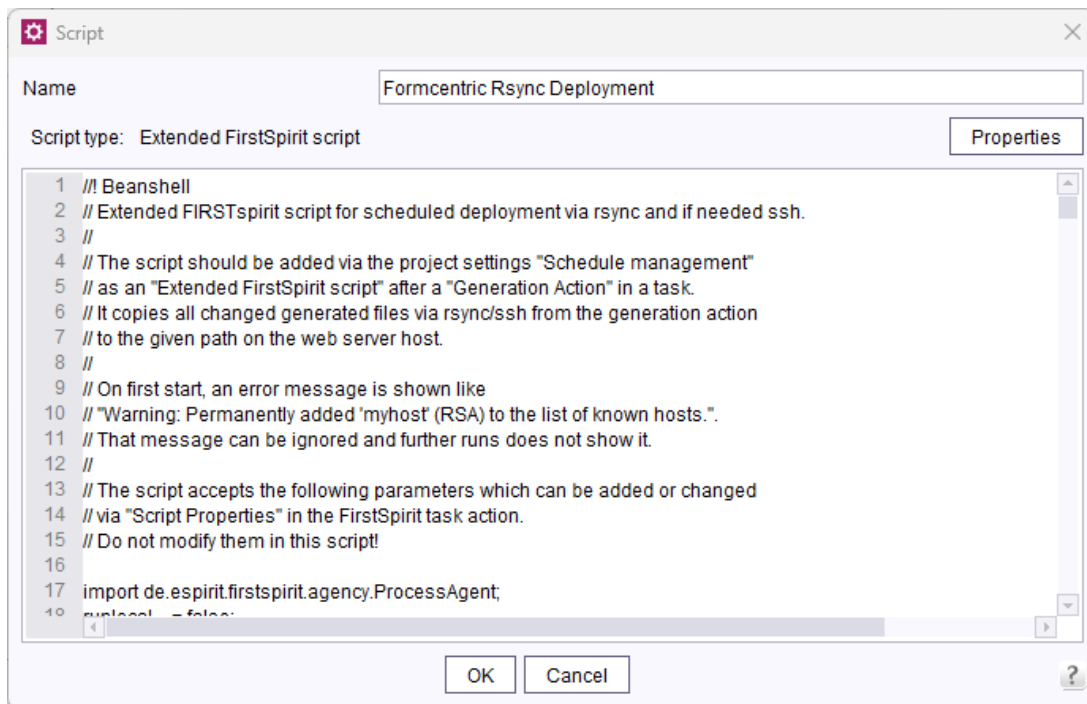


Figure 4.35. PDF action deployment script

As a next step, now configure the following parameters for your Preview or Delivery server, as described in section 10.5 of the FirstSpirit “Administrator Documentation”.

Parameter	Description
rsync	Path to the rsync program (only necessary on Windows systems). Example: <i>C:\cygwin\bin\rsync.exe</i>
runlocal	This parameter is used to specify whether publication is made to a local directory (<i>true</i>) or to a remote server (<i>false</i>).
subfolder	Path to a directory in Media Management in which the PDF templates are to be found. Example: <i>media/pdf</i>
webpath	Path to the directory in which the PDF templates should be stored following publication. Example: <i>/opt/firstspirit5/web/fc_preview/WEB-INF/pdf</i> Ensure that this directory is identical to that specified in the module configuration for the PDF action. The default value is <i>WEB-INF/pdf</i> .

The following parameters only need to be specified if publication is to take place to a remote server (*runlocal=false*).

Parameter	Description
webuser	Username that the system uses to log in to the remote server.
webhost	Hostname or IP address of the remote server.
ssh	Path to the ssh program. Example: <i>C:\cygwin\bin\ssh.exe</i>
privkey	On Windows systems, this parameter can be used to specify the full path to the SSH key file for the user configured by the <i>webuser</i> parameter. Example: <i>c:\User\fs5\.ssh\id_rsa</i>

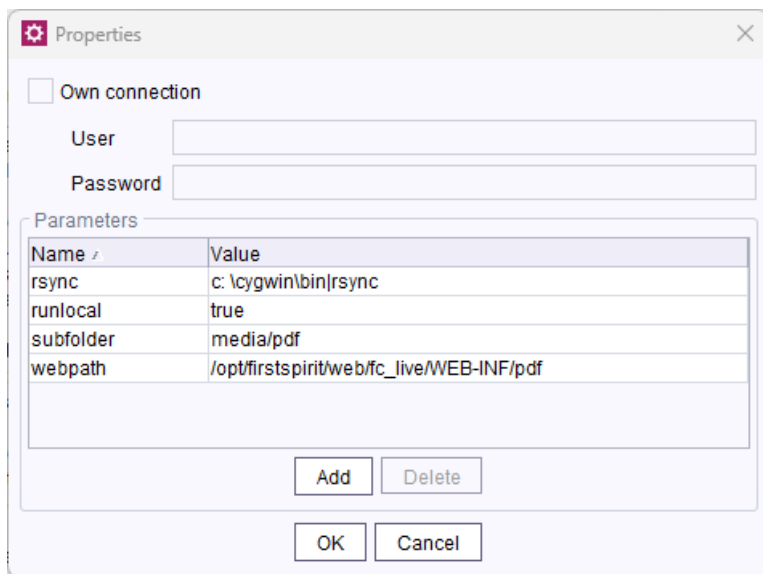


Figure 4.36. Sample configuration for the PDF action deployment script on a Windows server

As a final step, save the schedule template.

Use the new schedule template to create a new publication schedule that the editorial staff can use to transfer the PDF templates to the Preview web app (*fc_preview*) (see also section 3.4.3 in the Formcentric User Manual).

In order for the PDF templates to also be transferred with each deployment, you should include the schedule template in all relevant publication schedules. For further details of schedules and FirstSpirit schedule management, please consult section 7.5 of the FirstSpirit “Administrator Documentation”.

4.10. Password encryption

In the default configuration, login credentials for databases, mail servers, etc. are stored in various configuration files in plaintext. In the event of a security breach affecting the server, attackers would gain access to valid login credentials. For this reason, you are given the option of storing passwords in an encrypted format. In this case, passwords are decrypted only when the application starts, using the stored encryption password. You must ensure the password used for the encryption is stored in an environment variable before the Formcentric web applications start. The default environment variable used by Formcentric for this is *MWF_ENCRYPTION_PASSWORD*.

```
export MWF_ENCRYPTION_PASSWORD=my-encryption-password
```

Within the Formcentric configuration interface, you have the option of encrypting the password that is entered for each password input field.



Figure 4.37. Password field with encryption functionality

To do so, click the lock symbol within the field. This opens a dialog box in which you enter an encryption password with which the password is then encrypted. Please note: you must use the encryption password stored in the system for all of the passwords that are to be encrypted. No check is made to confirm that the password entered in the dialog matches the stored password.



Figure 4.38. Dialog for password encryption/decryption

If you are not using the FirstSpirit administration interface to install Formcentric, you also have the alternative option of using a command line program to encrypt and decrypt your passwords. After encrypting passwords in this way, they must then be entered manually into the corresponding configuration file.

Download the program from the Formcentric Maven repository by executing the following command at the command line. You can obtain the necessary login credentials by contacting our Helpdesk (support@formcentric.com).

```
mvn org.apache.maven.plugins:maven-dependency-plugin:3.0.2:copy \
-Dartifact=com.monday.webforms:encryption-cli:1.0:jar \
-DoutputDirectory=.
```

To encrypt a password, enter the following command at the command line:

```
java -jar encryption-cli-1.0.jar \
-p '<encryption-password>' -e '<password>'
```

Please note that the parameters must be entered in single quotation marks. You can enter the following command line parameters when starting:

Parameter	Description
<i>-p encryption-password</i>	The password to be used for encryption or decryption.
<i>-d</i>	Decrypt password
<i>-e</i>	Encrypt password
<i>-?</i>	Show help

5. Extending the FirstSpirit project

After installing the project component, you need to make a number of manual changes to the imported resources and to existing page templates.

5.1. Paragraph style sheet

At the content level, the system uses a paragraph style sheet to display forms. You can modify this to suit your precise requirements as described below. Out of the box, Formcentric ships with two paragraph style sheets – the paragraph style sheet *Headless form* is envisaged for use with the headless web application, while the paragraph style sheet *Form* is envisaged for use with the Spring MVC web application. In the default setup, the only difference between the two paragraph style sheets is the *Internet (HTML)* section.

5.1.1. Properties tab

One of the configuration options on the *Properties* tab is the option to set default values for new forms. Click the *Default values* button to open the Form Editor in a new window. Here, you can create a form for each project language: from now on, the system uses this form as the template for newly-created forms.

5.1.2. Form tab

For forms, you use the input component *FORMCENTRIC_FORMEDITOR*. You configure this on the *Form* tab. For a detailed description of the component, please consult the user manuals (*formcentric_contentcreator_en.pdf* and *formcentric_sitearchitect_en.pdf*).

```
<?xml version="1.0" encoding="UTF-8"?>
<FORMCENTRIC_FORMEDITOR name="form" expandOnStartup="yes" hFill="yes"
  useLanguages="yes">
  <DATASOURCES>
    <DATASOURCE name="Countries" type="comboBox">
      <LANGINFOS>
        <LANGINFO lang="*" label="English country names"/>
        <LANGINFO lang="DE" label="Englische Ländernamen"/>
      </LANGINFOS>
    </DATASOURCE>
    <DATASOURCE name="Laender" type="comboBox">
      <LANGINFOS>
        <LANGINFO lang="*" label="German country names"/>
        <LANGINFO lang="DE" label="Deutsche Ländernamen"/>
      </LANGINFOS>
    </DATASOURCE>
    <DATASOURCE name="Countries" type="inputField">
      <LANGINFOS>
        <LANGINFO lang="*" label="English country names"/>
        <LANGINFO lang="DE" label="Englische Ländernamen"/>
      </LANGINFOS>
    </DATASOURCE>
  </DATASOURCES>
</FORMCENTRIC_FORMEDITOR>
```

```

</DATASOURCE>
<DATASOURCE name="Laender" type="inputField">
  <LANGINFOS>
    <LANGINFO lang="*" label="German country names"/>
    <LANGINFO lang="DE" label="Deutsche Ländernamen"/>
  </LANGINFOS>
</DATASOURCE>
...
</DATASOURCES>
<FILE_TYPES>
  <FILE_TYPE name="bmp">
    <LANGINFOS>
      <LANGINFO lang="*" label="Windows-Bitmap (*.bmp)"/>
    </LANGINFOS>
  </FILE_TYPE>
  <FILE_TYPE name="gif">
    <LANGINFOS>
      <LANGINFO lang="*" label="CompuServe-Bitmap (*.gif)"/>
    </LANGINFOS>
  </FILE_TYPE>
  <FILE_TYPE name="jpeg">
    <LANGINFOS>
      <LANGINFO lang="*" label="JPEG Image (*.jpeg)"/>
      <LANGINFO lang="DE" label="JPEG Bild (*.jpeg)"/>
    </LANGINFOS>
  </FILE_TYPE>
  ...
</FILE_TYPES>
<FORM_LAYOUTS>
  <FORM_LAYOUT name="mwf-separator">
    <LANGINFOS>
      <LANGINFO lang="*" label="Horizontaal seperator"/>
      <LANGINFO lang="DE" label="Horizontale Trennlinie"/>
    </LANGINFOS>
  </FORM_LAYOUT>
</FORM_LAYOUTS>
<FORM_VARIABLES>
  <FORM_VARIABLE name="date"/>
  <FORM_VARIABLE name="time"/>
  <FORM_VARIABLE name="url"/>
  <FORM_VARIABLE name="language"/>
  <FORM_VARIABLE name="ip"/>
  <FORM_VARIABLE name="userAgent"/>
  <FORM_VARIABLE name="referer"/>
</FORM_VARIABLES>
<INPUT_STYLE_CLASSES>
  <INPUT_STYLE_CLASS name="mwf-s" type="inputField">
    <LANGINFOS>
      <LANGINFO lang="*" label="Small width (mwf-s)"/>
      <LANGINFO lang="DE" label="Kleine Breite (mwf-s)"/>
    </LANGINFOS>
  </INPUT_STYLE_CLASS>
  <INPUT_STYLE_CLASS name="mwf-m" type="inputField">
    <LANGINFOS>
      <LANGINFO lang="*" label="Medium width (mwf-m)"/>
      <LANGINFO lang="DE" label="Mittlere Breite (mwf-m)"/>
    </LANGINFOS>
  </INPUT_STYLE_CLASS>

```



```

    </LANGINFOS>
</INPUT_STYLE_CLASS>
<INPUT_STYLE_CLASS name="mwf-l" type="inputField">
  <LANGINFOS>
    <LANGINFO lang="*" label="Large width (mwf-l)"/>
    <LANGINFO lang="DE" label="Große Breite (mwf-l)"/>
  </LANGINFOS>
</INPUT_STYLE_CLASS>
<INPUT_STYLE_CLASS name="mwf-s" type="checkboxGroup">
  <LANGINFOS>
    <LANGINFO lang="*" label="Small width (mwf-s)"/>
    <LANGINFO lang="DE" label="Kleine Breite (mwf-s)"/>
  </LANGINFOS>
</INPUT_STYLE_CLASS>
<INPUT_STYLE_CLASS name="mwf-m" type="checkboxGroup">
  <LANGINFOS>
    <LANGINFO lang="*" label="Medium width (mwf-m)"/>
    <LANGINFO lang="DE" label="Mittlere Breite (mwf-m)"/>
  </LANGINFOS>
</INPUT_STYLE_CLASS>
  ...
</INPUT_STYLE_CLASSES>
<LANGINFOS>
  <LANGINFO lang="*" label="Form"/>
  <LANGINFO lang="DE" label="Formular"/>
</LANGINFOS>
<MAIL_ACTION>
  <MAIL_FORMATS>
    <MAIL_FORMAT name="text">
      <LANGINFOS>
        <LANGINFO lang="*" label="Plain Text"/>
        <LANGINFO lang="DE" label="Text" />
      </LANGINFOS>
    </MAIL_FORMAT>
    <MAIL_FORMAT name="html">
      <LANGINFOS>
        <LANGINFO lang="*" label="HTML"/>
        <LANGINFO lang="DE" label="HTML"/>
      </LANGINFOS>
    </MAIL_FORMAT>
    <MAIL_FORMAT name="freetext">
      <LANGINFOS>
        <LANGINFO lang="*" label="Freemarker (Plain Text)"/>
        <LANGINFO lang="DE" label="Freemarker (Text)"/>
      </LANGINFOS>
    </MAIL_FORMAT>
    <MAIL_FORMAT name="freehtml">
      <LANGINFOS>
        <LANGINFO lang="*" label="Freemarker (HTML)"/>
        <LANGINFO lang="DE" label="Freemarker (HTML)"/>
      </LANGINFOS>
    </MAIL_FORMAT>
  </MAIL_FORMATS>
</MAIL_ACTION>
<PDF_ACTION templateFolder="pdf,pdf2"/>
<PHONE_NUMBER_TYPES>

```

```

<PHONE_NUMBER_TYPE name="FIXED_LINE">
  <LANGINFOS>
    <LANGINFO lang="*" label="Fixed-line"/>
    <LANGINFO lang="DE" label="Festnetz"/>
  </LANGINFOS>
</PHONE_NUMBER_TYPE>
<PHONE_NUMBER_TYPE name="MOBILE">
  <LANGINFOS>
    <LANGINFO lang="*" label="Mobile"/>
    <LANGINFO lang="DE" label="Mobil"/>
  </LANGINFOS>
</PHONE_NUMBER_TYPE>
<PHONE_NUMBER_TYPE name="FIXED_LINE_OR_MOBILE">
  <LANGINFOS>
    <LANGINFO lang="*" label="Fixed-line or Mobile"/>
    <LANGINFO lang="DE" label="Festnetz oder Mobil"/>
  </LANGINFOS>
</PHONE_NUMBER_TYPE>
<PHONE_NUMBER_TYPE name="TOLL_FREE">
  <LANGINFOS>
    <LANGINFO lang="*" label="Toll-free"/>
    <LANGINFO lang="DE" label="Gebührenfrei"/>
  </LANGINFOS>
</PHONE_NUMBER_TYPE>
<PHONE_NUMBER_TYPE name="PREMIUM_RATE">
  <LANGINFOS>
    <LANGINFO lang="*" label="Premium Rate"/>
    <LANGINFO lang="DE" label="Premium-Tarif"/>
  </LANGINFOS>
</PHONE_NUMBER_TYPE>
<PHONE_NUMBER_TYPE name="SHARED_COST">
  <LANGINFOS>
    <LANGINFO lang="*" label="Shared Cost"/>
    <LANGINFO lang="DE" label="geteilte Kosten"/>
  </LANGINFOS>
</PHONE_NUMBER_TYPE>
<PHONE_NUMBER_TYPE name="VOIP">
  <LANGINFOS>
    <LANGINFO lang="*" label="Voice over IP"/>
    <LANGINFO lang="DE" label="Voice-over-IP"/>
  </LANGINFOS>
</PHONE_NUMBER_TYPE>
<PHONE_NUMBER_TYPE name="PERSONAL_NUMBER">
  <LANGINFOS>
    <LANGINFO lang="*" label="Personal Numbers"/>
    <LANGINFO lang="DE" label="Persönliche Nummer"/>
  </LANGINFOS>
</PHONE_NUMBER_TYPE>
...
</PHONE_NUMBER_TYPES>
<REDIRECT_ACTION httpsOnly="no" supportDynamicUrls="yes">
  <ALLOWED_HOSTS>
    <ALLOWED_HOST name="*" />
  </ALLOWED_HOSTS>
</REDIRECT_ACTION>
<REGEX_PATTERNS>

```

```

<REGEX_PATTERN name="^[0-9]*$">
  <LANGINFOS>
    <LANGINFO lang="*" label="Ciphers"/>
    <LANGINFO lang="DE" label="Nummern"/>
  </LANGINFOS>
</REGEX_PATTERN>
<REGEX_PATTERN name="^[0-9]{3,7}$">
  <LANGINFOS>
    <LANGINFO lang="*" label="3-7 ciphers"/>
    <LANGINFO lang="DE" label="3-7 Nummern"/>
  </LANGINFOS>
</REGEX_PATTERN>
<WEBEDIT_MACROS>
  <WEBEDIT_MACRO>{"name":"File upload", ... }</WEBEDIT_MACRO>
</WEBEDIT_MACROS>

...
</REGEX_PATTERNS>
</FORMCENTRIC_FORMEDITOR>

```

Consult the following section for a description of the various tags and attributes that you can use when configuring the Form Editor.

FORMCENTRIC_FORMEDITOR

Input component for forms (Form Editor)

Attribute	Description
name	You use the <i>name</i> attribute to give the Form Editor a variable name. You can then use this name in the templates in order to output the encrypted presentation text with the help of <code>\$CMS_VALUE()</code> (see Section 5.1.3, "Internet (HTML) tab").
expandOnStartup	You use the <i>expandOnStartup</i> parameter to specify whether or not the form elements are expanded in the tree view when the Editor is opened or the form is saved.
excludeElements	<p>You use the <i>excludeElements</i> parameter to hide form element types completely in the Form Editor. For the parameter value, enter a comma-separated list containing the names of the element types to hide. If the use of captchas and page breaks should be prohibited, for example, then you would enter the value <i>captcha,pageBreak</i>.</p> <p>You can select from the following element types:</p> <p><i>button, checkBoxGroup, comboBox, condition, fileUpload, hiddenField, shortText, emailField, numberField, dateField, phoneField, inputField, layout, fieldSet, pageBreak, paragraph, passwordField, radioGroup, calculatedValue, pageCondition, summary, textArea, captcha, sequenceAction, mailAction,</i></p>

Attribute	Description
	<i>dataSourceAction, datastoreAction, redirectAction, mediaStoreAction, pdfAction, webhookAction</i>
hFill	The Form Editor is displayed with a predefined width. If you want to use the full width available to you from your display, configure the <i>hFill</i> parameter with the value <i>YES</i> . Due to the large amount of space required by this input component, the setting <i>YES</i> is recommended.
useLanguages	You use the <i>useLanguages</i> parameter to specify whether or not the Form Editor should save different form definitions for the various languages (multilingual content management).

You can also specify the standard parameters *hidden, label, noBreak, preset, convertEntities* and *allowEmpty*. For a detailed description of this parameter, please consult the FirstSpirit documentation.

FORM_VARIABLES

You use the *FORM_VARIABLES* tag to define variables that a form author can then use when pre-setting values for input fields in the *Default value* field for single- and multi-line text input boxes. For example, the form author can set the current date (variable *date*) as the default value for an input field.

The standard variables *date, time, serverDate, serverTime, clientDate, clientTime, timezone, url, language, ip, userAgent* and *referer* are defined and can be used. In addition, you can also define your own, custom variables. For further details, see Section 6.5.7, “Adding variables for pre-filling form fields”.

The screenshot shows a form editor interface with an autocomplete list. The list is organized into sections: Value, Max. Length, CSS Class, and Validation. Each section contains a list of variables starting with a dollar sign and curly braces. The variable `${language}` is highlighted in blue. Below the list is a checkbox labeled "Readonly".

Figure 5.1. Autocomplete list with variables for a single-line text field

FORM_VARIABLE

You use the *FORM_VARIABLE* tag to enter the name of a form variable.

Attribute	Description
name	Name of the form variable.

FORM_LAYOUTS

The *FORM_LAYOUTS* tag enables the definition of a list of layout names that the form author can then later select via the layout element.

FORM_LAYOUT

You use the *FORM_LAYOUT* tag to enter the name of a layout.

Attribute	Description
name	Name of the layout.

An optional, language-dependent label for the layout can be configured by specifying a subordinate *<LANGINFOS>* tag (see the section called “LANGINFOS”).

FIELD_WIDTHS

The *FIELD_WIDTHS* tag lets you define a list of CSS classes from which the form author can make a selection later when creating a form element, so as to specify the width of the form element.

Out of the box, the CSS classes *mwf-s*, *mwf-m* and *mwf-l* are already included: these can be used to influence the width of input elements on the form page.

FIELD_WIDTH

Attribute	Description
name	Name of the CSS class.
type	Form element type for which the CSS class should be selectable by the form author. As standard, you can configure CSS classes for the following element types: <i>inputField, textArea, passwordField, checkBoxGroup, comboBox, paragraph, radioGroup, shortText, emailField, numberField, dateField, phoneField, fileUpload, pageBreak.</i>

An optional, language-dependent label for the CSS class can be configured by specifying a subordinate *<LANGINFOS>* tag (see the section called “LANGINFOS”).

INPUT_STYLE_CLASSES

The *INPUT_STYLE_CLASSES* tag lets you define a list of CSS classes, from which the form author can make a selection later when creating a form element.

INPUT_STYLE_CLASS

Attribute	Description
name	Name of the CSS class.

Attribute	Description
type	Form element type for which the CSS class should be selectable by the form author. As standard, you can configure CSS classes for the following element types: <i>form, inputField, textArea, passwordField, checkBoxGroup, comboBox, paragraph, radioGroup, shortText, emailField, numberField, dateField, phoneField, summary, fieldSet fileUpload, pageBreak.</i>

An optional, language-dependent label for the CSS class can be configured by specifying a subordinate <LANGINFOS> tag (see the section called “LANGINFOS”).

DATASOURCES

You use the *DATASOURCES* tag to define dynamic data sources (REST services), which the form author can then select in order to fill a form element with values (see Section 6.5.8, “Implementing a REST service”).

DATASOURCE

Attribute	Description
name	Name of the REST service, as you entered it in the service mapping for the REST controller.
type	Form element type that the form author can use to select the data source. As standard, you can configure data sources only for drop-down lists of the types <i>comboBox, radioGroup, checkBoxGroup, inputField</i> and <i>hiddenField</i> .

An optional, language-dependent label for the data source can be configured by specifying a subordinate <LANGINFOS> tag (see the section called “LANGINFOS”).

FILE_TYPES

You use the *FILE_TYPES* tag to define file types that the form author can select for the file validator.

FILE_TYPE

Attribute	Description
name	Name of the file type, as entered into the MIME type mapping for the file validator. As standard, you can define the following file types: <i>bmp, gif, jpeg, png, wmf, tif, mp3, wav, mp4, avi, mpg, wma, mov, asf, wmv, zip, gz, tar, gtar, 7z, rar, arj, bz, bz2, doc, ppt, pptx, xls, xlsx, docx, mpx, wps, pdf, rtf, flv, txt, dvi, xml, js, css, xhtml, html, svg, psd, rpm, indd, inx</i>

An optional, language-dependent label for the file type can be configured by specifying a subordinate `<LANGINFOS>` tag (see the section called “LANGINFOS”).

PHONE_NUMBER_TYPES

You use the `PHONE_NUMBER_TYPES` tag to define phone number types that the form author can select when using the phone number validator.

PHONE_NUMBER_TYPE

Attribute	Description
name	Name of the phone number type, as entered into the MIME type mapping for the phone number validator. In the default setup, you can configure the following phone number types: <i>FIXED_LINE, MOBILE, FIXED_LINE_OR_MOBILE, TOLL_FREE, PREMIUM_RATE, SHARED_COST, VOIP, PERSONAL_NUMBER, UAN, VOICEMAIL, UNKNOWN.</i>

An optional, language-dependent label for the phone number type can be configured by specifying a subordinate `<LANGINFOS>` tag (see the section called “LANGINFOS”).

REGEX_PATTERNS

You use the `REGEX_PATTERNS` tag to define regular expressions that the form author can select when using the regular expression validator.

REGEX_PATTERN

Attribute	Description
name	The regular expression that you are now adding to the list of examples in the regular expression validator.

An optional, language-dependent label for the regular expression can be configured by specifying a subordinate `<LANGINFOS>` tag (see the section called “LANGINFOS”).

MAIL_ACTION

You use this tag to configure the mail action dialog for editorial staff.

MAIL_FORMATS

The `MAIL_FORMATS` tag enables you to define a list of format identifiers that the form author can subsequently use within the mail action in order to specify the format

of the email sent by the system. If you do not specify any format identifiers, then the default formats of *html* and *text* are offered for selection.

MAIL_FORMAT

Attribute	Description
name	Name of the email format, identical to the entry in <i>bodyRendererMapping</i> that you have made for the mail action (see also the section called “formcentric-actions.xml”).

An optional, language-dependent label for the email format can be configured by specifying a subordinate `<LANGINFOS>` tag (see the section called “LANGINFOS”).

PDF_ACTION

You use this tag to configure the PDF action dialog for editorial staff.

Attribute	Description
templateFolder	All PDF documents that are to be used as a template for the PDF action must be stored in a master folder within Media Management. Enter the reference name of the folder here or use a comma-separated list for multiple folders. In the PDF action dialog for editorial staff, selection is restricted to PDF documents in these folders.

REDIRECT_ACTION

You use this tag to configure the redirect action dialog for editorial staff.

Attribute	Description
httpsOnly	You use the <i>httpsOnly</i> parameter to specify that only secure URLs can be entered as the redirection target.
supportDynamicUrls	You use the <i>supportDynamicUrls</i> parameter to specify whether or not the redirect URL variables can use the <code>\${field name}</code> format.

ALLOWED_HOSTS

This tag gives you the option of restricting the possible redirection targets to certain addresses (hosts).

ALLOWED_HOST

Specification of an allowed host.

If, for example, you want to ensure that only your own company webpages can be entered as the redirection target, then you should enter the domain name of your

company here. Note that you can use placeholders such as *.your-domain.com in order to also allow the use of all subdomains.

Attribute	Description
name	Name of an allowed host.

LANGINFOS

You use this tag to specify language-dependent labels for the attribute *name* in the parent tag; these labels are displayed in the form editing interface. This applies to the tags *DATASOURCE*, *FILE_TYPE*, *INPUT_STYLE_CLASS*, *MAIL_FORMAT*, *FORM_LAYOUT*, *PHONE_NUMBER_TYPE* and *REGEX_PATTERN*.

LANGINFO

This tag is used to set the label for a specific language.

Attribute	Description
lang	Language code for the display language (e.g. <i>DE</i> for German, <i>EN</i> for English).
label	Text of the label in the specified language
description	You can use the <i>description</i> parameter to specify an optional description that is used to show a tooltip (on mouseover).

5.1.3. Internet (HTML) tab

Form presentation is normally dynamic. As a result – and depending on the input made by the user, for example – individual form pages or form fields can be shown or hidden on the fly. For this reason, the HTML form output is created either server-side by the web application or browser-side by a React application (Formcentric client) and is then dynamically embedded into the respective page. For the server-side approach, the web app component is accessed by using an asynchronous JavaScript request (AJAX). The paragraph style sheet generates only the JavaScript for embedding the AJAX response. With the browser-side approach, however, the Formcentric Client is executed in the browser itself. In this case, the paragraph style sheet generates the JavaScript for embedding the client.

When using the Spring MVC web application, the required JavaScript libraries and CSS styles must be loaded in the page template (see Section 5.6, “Page template”). When using the headless web application, these are referenced in the paragraph style sheet and loaded by the Formcentric client.

The following example shows the output template for use with the Spring MVC web application:

```
$CMS_RENDER(script: "formcentric_encrypted_form",
             form: form.XML)$
$CMS_RENDER(script: "formcentric_encrypted_refs",
```

```

        ids: form.internalReferences)$
$CMS_RENDER(script: "formcentric_login_ticket")$

<!-- Formcentric -->
<div class="clearfix module"$CMS_VALUE(editorId())$>
  <div class="mwf-form">
    <div id="ajaxreplace$CMS_VALUE(form.oid)$">
      <script type="text/javascript">
        jQuery(function() {
          jQuery.mwfAjaxReplace({
            uid: '$CMS_VALUE(form.oid)$',
            selector: '#ajaxreplace$CMS_VALUE(form.oid)$',
            url: '$CMS_RENDER(script:"formcentric_url")$/servlet/form',
            appendUrlVars: true,
            data: {
              _view: 'webform',
              _fd: '$CMS_VALUE(fc_encryptedForm)$',
              _refs: '$CMS_VALUE(fc_encryptedRefs)$',
              _lang: '$CMS_VALUE(form.lang())$',
              _ticket: '$CMS_VALUE(fc_loginTicket)$'
            }
          });
        });
      </script>
    </div>
  </div>
</div>

```

On each form submit, the DIV tag with the ID *ajaxreplace<form ID>* is replaced by the AJAX response.

For the headless web application, the output template is presented as follows:

```

$CMS_RENDER(script: "formcentric_encrypted_form",
  form: form.XML)$
$CMS_RENDER(script: "formcentric_encrypted_refs",
  ids: form.internalReferences)$
$CMS_RENDER(script: "formcentric_login_ticket")$

<!-- Formcentric -->
<div class="clearfix module"$CMS_VALUE(editorId())$>
  <div data-fc-id="$CMS_VALUE(form.oid)$"
    data-fc-formapp-url="$CMS_REF(media:"formapp")$"
    data-fc-theme-url="$CMS_REF(media:"formcentric_headless_css")$"
    data-fc-template-url="$CMS_REF(media:"formcentric_templates_js")$"
    data-fc-theme-variable-url="$CMS_REF(media:"formcentric_json")$"
    data-fc-form-definition="$CMS_VALUE(fc_encryptedForm)$"
    data-fc-refs="$CMS_VALUE(fc_encryptedRefs)$"
    data-fc-params='{"ticket": "$CMS_VALUE(fc_loginTicket)$}'
    data-fc-data-url='$CMS_RENDER(script:"formcentric_url")$'
  ></div>
</div>

```

The Formcentric client is configured using data attributes:

data-fc-id: Specifies the form ID.

data-fc-formapp-url: Specifies the URL for the Formcentric client's JavaScript file.

data-fc-theme-url: The URL to the application's CSS styles.

data-fc-template-url: The URL to the JavaScript file for the form templates.

data-fc-theme-variable-url: The URL to the JSON file with the theme variables.

data-fc-form-definition: The encrypted form definition.

data-fc-refs: The form's encrypted references.

data-fc-params: Additional parameters that will be passed to the headless application as a JSON object. For the headless application's login to the Preview, a login ticket is usually passed here.

data-fc-data-url: The URL of the headless web application.

5.2. formcentric_headless_url script

The system uses the generation script *formcentric_headless_url* within the paragraph style sheet to output the context paths of the global headless web applications (see Section 4.1, "Installing the Formcentric module").

Take care to ensure that the values of the script variables *previewWebAppId* and *liveWebAppId* contain the IDs of the global web applications. If necessary, adjust these to match.

```
#!/Beanshell
previewWebAppId = "fc_headless_preview";
liveWebAppId = "fc_headless_live";

agent = context.requestSpecialist(
    de.espirit.firstspirit.agency.LegacyModuleAgent.TYPE);

url = context.isPreview() ? agent.getGlobalWebAppUrl(previewWebAppId) :
    agent.getGlobalWebAppUrl(liveWebAppId);

return url;
```



For the context path output of the Spring MVC web application, please use the generation script *formcentric_url*, which uses the same structure.

5.3. formcentric_encrypted_form script

The form definition must be encrypted before being passed to the Formcentric web app. The *formcentric_encrypted_form* script calls an executable that encrypts the form definition and stores it in the variable *fc_encryptedForm* in the page context. When the script is called, the unencrypted form definition must be passed in the *form* script parameter.

```
$CMS_RENDER(script: "formcentric_encrypted_form", form: form.XML)$
```

5.4. formcentric_encrypted_refs script

References within the form definition to FirstSpirit objects such as images or PDF template documents must, like the form definition, be passed to the Formcentric web app in an encrypted format. You use the *formcentric_encrypted_refs* script to do this. The script calls an executable that identifies the external URLs to the FirstSpirit objects and encrypts these URLs. The encrypted URLs are saved in the *fc_encryptedRefs* variable in the page context. When this script is called, all of the internal references must be passed to the script in the *ids* parameter.

You can use the optional parameter *resolutions* to specify a comma-separated list of resolutions: the image URLs will then be generated at these resolutions. If you do not specify the *resolutions* parameter, the image URLs are generated for all resolutions.

```
$CMS_RENDER(script: "formcentric_encrypted_refs", ids: form.allReferences,  
    resolutions: "47x47, 60x40")$
```

5.5. formcentric_login_ticket script

A FirstSpirit login ticket is required in order to access the Preview web app content. This ticket is passed when the Formcentric web app is called, together with the encrypted form definition and the encrypted internal references. The *formcentric_login_ticket* script generates a login ticket and saves it in the *fc_loginTicket* variable in the page context. If the context is not the Preview context, the variable contains an empty string.

```
$CMS_RENDER(script:"formcentric_login_ticket")$
```

5.6. Page template

To display the forms correctly, additional JavaScripts and CSS styles are required. These will have been imported during the installation of the Formcentric project component.

When using the Spring MVC web application, extend the output of the HTML header in your page templates to include the imported CSS (*formcentric_css*) and the imported JavaScripts. Please take care to ensure that the JavaScripts are loaded in the order as given, since some have dependencies on others.

Alternatively, you can also use the minified version of the scripts (**_min*).

```
...  
<link rel="stylesheet" type="text/css"  
    href="$CMS_REF(media:"formcentric_flex_css_min")$" />
```

```

...
$CMS_FOR(_script, [
  "json2_js_min",
  "jquery_3_6_0_js_min",
  "select2_4_0_13_js_min",
  "jquery_autocomplete_js_min",
  "jquery_format_1_3_js_min",
  "jquery_ui_widget_1_13_2_js_min",
  "load_image_all_js_min",
  "canvas_to_blob_js_min",
  "jquery_xdr_transport_js_min",
  "jquery_iframe_transport_js_min",
  "jquery_fileupload_10_31_0_js_min",
  "jquery_fileupload_process_10_31_0_js_min",
  "jquery_fileupload_image_10_31_0_js_min",
  "jquery_formcentric_1_9_js_min"
])$

<script type="text/javascript" src="$CMS_REF(media:_script)$"></script>
$CMS_END_FOR$
...

```



Please note that the actual reference names may differ from the names as specified here. During content item import, FirstSpirit appends a sequential number to the item's reference name if a content item with this name already exists. In this case, ensure that you use the extended reference name.

For the headless web application, only one JavaScript needs to be loaded in the page template. All other JavaScripts and CSS styles required are referenced in the corresponding *Headless Form* paragraph style sheet and loaded when a form is accessed by the Formcentric Client (see Section 5.1.3, "Internet (HTML) tab").

```

...
<script type="text/javascript"
  src="$CMS_REF(media:"formcentric")$" defer></script>

```

Specifying the *defer* attribute ensures that the JavaScript is loaded asynchronously and not executed until page parsing is complete.

5.7. Themes

For the display of headless forms, Formcentric offers a variety of themes that can be customized to meet your specific needs. Each theme consists of a template file, a CSS file, and a JSON file. The JSON file contains specific CSS parameters that allow you to modify the appearance of the theme to suit your requirements. For more extensive customizations, you have the option to download the source code from the Formcentric npm repository using the following URL:

<https://maven.monday-consulting.com:443/artifactory/formcentric-npm/@formcentric/client/-/@formcentric/client-1.0.7.tgz>

5.8. CSS

Formcentric already includes a simple CSS, which you can use as the basis for designing your own forms. The JSP templates of the Spring MVC application use the CSS classes defined within this CSS. The example below shows you the generated HTML presentation layer (extract) for a simple contact form.

```
<div id="ajaxreplaceDE2790">
  <form accept-charset="utf-8" class="mwf-form" data-mwf-id="DE187126"
    enctype="multipart/form-data" id="commandDE187126"
    method="post" onsubmit="return false;">

    <div class="mwf-layout mwf-layout--default">
      <div class="mwf-textinput" data-mwf-container="fccc48273e2eeb">
        <input id="fccc48273e2eeb" type="text" name="email"
          class="mwf-textinput__input" data-mwf-id="fccc48273e2eeb" />
        <label class="mwf-textinput__label" for="fccc48273e2eeb">Email</label>
      </div>

      <div class="mwf-textinput" data-mwf-container="fc313d3971298a">
        <input id="fc313d3971298a" name="message" type="text"
          class="mwf-textinput__input" data-mwf-id="fc313d3971298a" />
        <label class="mwf-textinput__label" for="fc313d3971298a">Message</label>
      </div>
    </div>
  </form>
</div>
```

The system outputs the form as series of *div* containers, with each form element being embedded within a separate *div* element. The various HTML elements are given specific CSS class syntax, so as to ensure that you can use CSS to style both the entire form and its individual components – such as input fields, labels and error messages. To avoid polluting other CSS classes in the project, all class names begin with the *mwf* prefix.

6. Programming and customisation

This section, written from a software developer's point of view, shows you how you can extend or modify Formcentric. Alongside general knowledge of Java and XML, you will also need to know the fundamentals of the Spring framework and the Maven build system.

6.1. Development workspace

To give you a head start when developing your extensions, Monday provides you with a pre-configured development workspace. The workspace in question is a Maven workspace that contains all of the examples described in this manual.

The workspace consists of nine Maven artefacts:

Artefact directory	Description
/formcentric-admin-customizations	Server and Project Configuration extensions.
/formcentric-editor-customizations	Extensions to input components
/formcentric-module-customizations	Module descriptor and FirstSpirit project resources.
/formcentric-webapp-customizations	Extensions to the Spring MVC web application
/formcentric-webapp-lib-customizations	Java classes for extensions to the Spring MVC web application
/formcentric-headless-webapp-customizations	Extensions to the headless server (WAR Deployment)
/formcentric-headless-server-customizations	Extensions to the headless server (Embedded Web Server Deployment)
/formcentric-headless-first-spirit-customizations	Java classes for extensions to the headless web application
/formcentric-webedit-customizations	Extensions to the web editor component.
/formcentric-webedit-lib-customizations	Java classes for the extended web editor component.

All required JAR archives (dependencies) are downloaded from the Monday Maven server (<http://maven.monday-consulting.com>) when building the workspace. You will need to enter the necessary login details beforehand in the Maven configuration file *settings.xml*. The configuration file can be found in the root directory of the development workspace.

To obtain your personal login details, please contact our Helpdesk (support@formcentric.com).

```

...
<servers>
  <server>
    <id>maven.monday-consulting.com</id>
    <username>my-username</username>
    <password>my-password</password>
  </server>
</servers>
...

```

As a final step, modify the FirstSpirit version specified in the root POM (*pom.xml*) to match the version you have deployed.

```
<cms.version>5.2.190105</cms.version>
```

To build the workspace, switch to the directory `<formcentric-firstspirit-workspace>` and execute the following from the command line:

```
mvn -s ./settings.xml install
```

The FirstSpirit module archive created is stored in the directory `<formcentric-firstspirit-workspace>/formcentric-module-customizations/target`.

6.2. Monday Maven plugin

Each FirstSpirit module contains a specialised module descriptor (*module.xml*), which describes the components and files contained within the corresponding module. For Formcentric, JAR archives are a key part of this, alongside configuration files and output templates. To simplify the creation of the module descriptor, Formcentric has a specialised Maven plugin that is used to generate the FirstSpirit module descriptor. The plug-in parses the dependencies defined in the Maven configurations and uses these to create the `<resource>` entries in the module descriptor.

The plugin is configured in the Maven configuration (*pom.xml*) of the *formcentric-module-customizations* artefact.

You need to specify the following parameters in the plugin's `<configuration>` tag.

Parameter	Description
configXml	References the configuration file for the Maven plugin.
prototypeXml	References the file that serves as the template for the module descriptor to be created.

```

<plugin>
  <groupId>com.monday-consulting.maven.plugins</groupId>
  <artifactId>fsm-maven-plugin</artifactId>
  <configuration>

```



```

<configXml>${basedir}/target/extra-resources/fsm-plugin.xml</configXml>
<prototypeXml>${basedir}/target/extra-resources/
  prototype.module.xml</prototypeXml>
<targetXml>${basedir}/target/extra-resources/module.xml</targetXml>
</configuration>
<executions>
  <execution>
    <id>dependencyToXML</id>
    <phase>package</phase>
    <goals>
      <goal>dependencyToXML</goal>
    </goals>
  </execution>
</executions>
</plugin>

```

To generate the FirstSpirit module descriptor, the template *prototype.module.xml* is used, which is stored in the directory */formcentric-module-customizations/src/non-packaged-resources*. In this file, a number of variables are used, which are then replaced when building the workspace. Modified class names, etc. must be configured here.

```

<module>
  <name>Formcentric</name>
  <version>${project.version}</version>
  <description>Formcentric Form Editor</description>
  <vendor>Formcentric GmbH</vendor>
  <components>
    <public>
      <name>FORMCENTRIC_FORMEDITOR</name>
      <description>Formcentric editor component</description>
      <class>de.espirit.firstspirit.module.GadgetSpecification</class>
      <configuration>
        <gom>com.formcentric.examples.gom.CustomGomFormEditor</gom>
        <factory>com.formcentric.editor.
          gadgets.FormEditorSwingGadgetFactory</factory>
        <value>com.formcentric.editor.
          gadgets.FormValueEngineerFactory</value>
        <scope data="yes" content="yes" link="yes"/>
      </configuration>
      <resources>
        <resource>files</resource>
        <!-- Variable, which is replaced by the Maven plugin. -->
        <dependencies>webforms-editor-resources</dependencies>
      </resources>
    </public>

    <web-app scopes="global">
      <name>Formcentric WebApp</name>
      <description>Formcentric FIRSTspirit integration.</description>
      <configurable>com.formcentric.examples.
        admin.CustomWebAppConfiguration</configurable>
      <class>com.formcentric.admin.webapp.FormcentricWebApp</class>
      <web-xml>web.xml</web-xml>
      <resources>

```

```

    <!-- Variable, which is replaced by the Maven plugin. -->
    <dependencies>formcentric-admin-resources</dependencies>
  </resources>
  <web-resources>
    ..
    <!-- Variable, which is replaced by the Maven plugin. -->
    <dependencies>formcentric-webapp-resources</dependencies>
  </web-resources>
</web-app>

<project-app>
  <name>Formcentric Resources</name>
  <description>Formcentric project resources</description>
  <class>com.formcentric.admin.project.FormcentricProjectApp</class>
  <resources>
    <resource>files</resource>
    <!-- Variable, which is replaced by the Maven plugin. -->
    <dependencies>formcentric-admin-resources</dependencies>
  </resources>
</project-app>

</components>
</module>

```

The configuration file *fsm-plugin.xml* is used to define the *<dependencies>* variable with its corresponding dependencies.

```

<fsm-maven-plugin>
  <scopes>
    <scope>runtime</scope>
    <scope>compile</scope>
  </scopes>
  <modules>
    <module>
      <id>change.this.now.fs5:formcentric-admin-customizations:jar</id>
      <prefix>lib</prefix>
      <dependencyTagValueInXml>formcentric-admin-resources
        </dependencyTagValueInXml>
      <firstSpiritScope>module</firstSpiritScope>
    </module>
    <module>
      <id>change.this.now.fs5:formcentric-editor-customizations:jar</id>
      <prefix>lib</prefix>
      <dependencyTagValueInXml>formcentric-editor-resources
        </dependencyTagValueInXml>
      <firstSpiritScope>module</firstSpiritScope>
    </module>
    <module>
      <id>change.this.now.fs5:formcentric-webapp-lib-customizations:jar</id>
      <prefix>lib</prefix>
      <dependencyTagValueInXml>formcentric-admin-resources
        </dependencyTagValueInXml>
    </module>
  </modules>
</fsm-maven-plugin>

```

Element	Description
<fsm-maven-plugin>	Root element.
<scopes>	Optionally specifies the Maven scopes that should be considered when determining the dependent Maven artefacts. Each scope must be specified separately. If not specified, the default scopes <i>compile</i> and <i>runtime</i> apply.
<scope>	Designation of a maven scope to be considered.
<modules>	Container element for the definition of artefact dependencies.
<module>	Specifies an artefact whose dependencies should be written into the module descriptor.
<id>	The artefact's Maven identifier. Must be specified using the standard Maven notation <i><groupId>:<artifactId>:<type></i> .
<dependencyTagValueInXml>	Name of the <i><dependencies></i> variable that is entered into the module descriptor template.
<firstSpiritScope>	Specifies the FirstSpirit scope (see also section 2.5.1.2, p. 13 in the Developer Manual "MDEVDE_FirstSpirit_ModulDeveloperDoc.pdf").

6.3. Extending the input component in Site Manager

The Monday Form Editor is based on a general framework with which Swing-based XML editors can be developed as required. Within this framework, an XML document is represented by a hierarchical structure of *NodeModel* objects. Each *NodeModel* object corresponds to one or more elements in the XML structure. The framework provides methods for the display, selection and editing of the individual *NodeModel* objects.

Each *NodeModel* object is assigned a *NodeType* object. This contains information about how the associated *NodeModel* is to be initialised, presented, processed and validated. For each *NodeType* object, you can also specify which other XML elements can be assigned beneath the model.

The *NodeType* objects required are generated by the *EditorSetup* class. You therefore use this as your starting point for extending the Form Editor.

Within the form framework, actions encapsulate the actual business logic for form data processing. The following section gives you an example using an action to demonstrate how you can extend the Form Editor to include additional elements.

6.3.1. Developing a NodeEditorPane

For each element type, the system requires a modified editor, which must be derived from the *NodeEditorPane* class. The node editor is used to process a *NodeModel* of the corresponding type. The code snippet given below shows you the implementation of a *NodeEditorPane*.

```
public class CustomActionEditorPane extends TabbedBaseActionEditorPane {

    public static final String PROP_CUSTOM = "customProperty";
    protected JPanel propertiesPanel;
    protected JTextField customPropertyField;
    protected ActionModel model; // extends NodeModel

    PropertiesBundle custombundle =
        PropertiesBundle.getInstance("com/custom/forms/formeditor",
            CustomActionEditorPane.class.getClassLoader());

    protected JTextField getCustomPropertyField() {
        if (customPropertyField == null) {
            customPropertyField = new JTextField();
            customPropertyField.addFocusListener(focusListener);
            customPropertyField.addKeyListener(new KeyAdapter() {
                public void keyReleased(final KeyEvent e) {
                    model.setProperty(PROP_CUSTOM, customPropertyField.getText());
                }
            });
        }
        return customPropertyField;
    }

    protected JPanel getPropertiesPanel() {
        if (propertiesPanel != null) {
            return propertiesPanel;
        }

        propertiesPanel = new JPanel();

        GridBagDesigner layout =
            new GridBagDesigner(custombundle, propertiesPanel);

        layout.row().labelTop("propLabel").expand()
            .add(getCustomPropertyField());

        layout.fill();

        return propertiesPanel;
    }

    public void setEditable(final boolean editable) {
        if (customPropertyField != null) {
            customPropertyField.setEditable(editable);
        }
    }

    public void setModel(NodeModel model) {
```

```

    this.model = (ActionModel) model;
    update();
}

public void update() {
    getCustomPropertyField().setText(
        (String) model.getProperty(PROP_CUSTOM));
}
}

```

The example given above shows you a node editor containing a text field in which the value of the property *customProperty* can be entered. The text field and the corresponding label are arranged in a `GridBagLayout` on the underlying `JPanel`. For the purpose of editor internationalisation, the text of the label is read from a language-dependent resource bundle.

Text input is transferred to the `NodeModel` using an anonymous `KeyListener`.

6.3.2. Extending the `EditorSetup` class

Within the `init` method of the `EditorSetup` class, the `NodeType` objects of the Form Editor are generated. By overriding this method, you can generate and register an additional `NodeType` object for the new `CustomAction`:

```

@Override
public void init() {

    super.init();

    PropertiesBundle custombundle =
        PropertiesBundle.getInstance("com/custom/forms/formeditor"
            this.getClass().getClassLoader());

    // Generate new type
    NodeType customActionType = createCustomActionType(custombundle);

    // Register new type
    registerType(customActionType);

    // Register as possible sub-element of the form element
    NodeType formNodeType = getNodeType(FORM.TYPE.FORM);
    formNodeType.addAllowedChildType(customActionType);
}

```

The `NodeType` objects of the individual form elements are typically generated in separate factory methods. The following example shows you how they are implemented:

```

protected NodeType createCustomActionType(final EditorBundle bundle) {

    final String CUSTOM_ACTION = "customAction";

    NodeType action = new NodeType(CUSTOM_ACTION, ActionModel.class,
        CustomActionEditorPane.class);
}

```

```

action.setTitle(bundle.getString(CUSTOM_ACTION + "Label"));

action.setIcon(bundle.getImageIcon(CUSTOM_ACTION + "Icon"));

action.setToolBarIcon(bundle.getImageIcon(CUSTOM_ACTION + "LargeIcon"));

action.setValidator(new CustomActionValidator());

action.setInitializer(new CustomActionInitializer());

action.setButtonGroup(1);

return action;
}

```

In the example given above, the first step is to create a new *NodeType* instance with the name *customAction*. The name serves to identify this type uniquely in further processing steps. As one example, the associated action implementation is identified using this name in the web application. Accordingly, you can assign type names only once. Following this, a specialised *NodeValidator* and a *NodeInitializer* are defined.



If you want to modify the initialisation or validation of an existing form element, then it is generally sufficient for you to override the associated factory method, so as to set a different validator or initialiser.

6.3.3. Extending the Form Editor GUI object model

The *EditorSetup* class is instantiated via the Form Editor's GUI object model (GOM). By overriding the *createEditorSetup* method in the class *com.formcentric.editor.gadgets.FormEditorSwingGadgetFactory*, you generate the new *EditorSetup* class instead of the existing one.

```

public class CustomFormEditorSwingGadgetFactory extends
    FormEditorSwingGadgetFactory {
    @Override
    public FormEditorSetup createEditorSetup(GomFormEditor gomFormEditor,
        SpecialistsBroker broker, Language lang) {
        return new CustomFormEditorSetup(gomFormEditor, broker, lang));
    }
}

```

Configure the modified *CustomGomFormEditor* class in the *prototype.module.xml* module descriptor in the input component's *configuration* element.

```

<name>MONDAY_FORMEDITOR</name>
<description>Monday Webforms editor component</description>
<class>de.espirit.firstspirit.module.GadgetSpecification</class>
<configuration>
    <factory>com.custom.webforms.CustomFormEditorSwingGadgetFactory</factory>
    ...
</configuration>

```

6.4. Extending the ContentCreator web application

The Formcentric ContentCreator integration is a single-page application that is based on the JavaScript *React* framework. The Form Editor's user interface is generated client-side on the browser using the JSON data sent by the server. The interface layout is specified declaratively using a number of JavaScript configuration files. This approach makes it easy for you to make changes and create extensions to the form editing interface.

The JavaScript configuration files that are stored in the development workspace in the *formcentric-webedit-customizations* module in the directory */src/main/webapp/WEB-INF/formcentric_editor/gadget/formeditor/config/editor* constitute the main starting-point for making changes to the form editing interface. All of the changes described below are made to the files in this directory.

The available form elements and their properties are described as JSON objects. The React application uses these to generate the form editing interface. The following example shows a configuration snippet for the *textArea* form element.

```
{
  icon: 'textarea',
  type: 'textArea',
  properties: {
    general: [
      {
        title: 'name',
        type: 'text',
        properties: {
          required: true
        }
      },
      {
        title: 'label',
        type: 'text'
      },
      {
        title: 'hint',
        type: 'text'
      },
      {
        title: 'value',
        type: 'wysiwyg'
      },
      ...
    ]
  }
}
```

6.4.1. Adding a new form element

Extend the Form Editor to include a new form element by extending the configuration *fields_custom.js*. If you want to extend the Editor to include the form element

termsCheckbox with the properties *name*, *text* and *link*, for example, then add the following object definition to the JavaScript array in the configuration *fields_custom.js*.

```
[
  {
    icon: 'termscheckbox',
    type: 'termsCheckbox',
    properties: {
      general: [
        {
          title: 'name',
          type: 'text',
          properties: {
            required: true
          }
        },
        {
          title: 'text',
          type: 'wysiwyg',
          properties: {
            required: true
          }
        },
        {
          title: 'link',
          type: 'reference',
          properties: {
            refType: 'pageref',
            FS_refType: 'pageref'
          }
        }
      ],
      specialProperties: {
        condition: {
          conditionable: false,
          operators: {}
        }
      }
    }
  }
]
```

Please note: The external JavaScript array already exists and simply needs to be extended by the configuration object.

The table below describes the possible attributes that a field definition can have at the first level.

Attribute	Description
icon	Type: <i>String</i> Name of the icon to load. The name specified must match the filename of the icon without the file extension.
type	Type: <i>String</i>

Attribute	Description
	Form element name
properties	<p>Type: <i>Object</i></p> <p>Defines the properties of a field that can be edited in Content-Creator on the right-hand side, under <i>Field properties</i>. The object properties of <i>properties</i> each correspond to individual Editor tabs. The following JSON snippet configures two tabs with the name <i>general</i> and <i>special</i>, with a total of three properties: <i>name</i>, <i>label</i>, <i>hint</i>.</p> <pre data-bbox="438 622 1279 1238"> properties: { general: [{ title: 'name', type: 'text' }, { title: 'label', type: 'text' }], special: [{ title: 'hint', type: 'text' }] } </pre> <p>To ensure that the field can be uniquely identified during later processing, the property <i>title</i> is required with the value <i>name</i> in the <i>general</i> array.</p> <p>For a list of all available property types, please see Section 6.4.5, “Input elements for element properties”.</p>
specialProperties	<p>Type: <i>Object</i></p> <p>You use the <i>specialProperties</i> attribute to configure properties that are evaluated by the Editor for internal functions. The following JSON snippet defines the usage of the field within a condition.</p> <pre data-bbox="438 1742 1279 2011"> specialProperties: { condition: { conditionable: true, operators: { startswith: { values: [], freeField: true, useChildren: false } } } } </pre>

Attribute	Description
	<pre data-bbox="438 241 1279 712"> }, endswith: { values: [], freeField: true, useChildren: false }, contains: { values: [], freeField: true, useChildren: false } } } } </pre> <p data-bbox="438 741 1279 813">You set <i>conditionable: true</i> to specify that the field can be selected in a condition.</p> <p data-bbox="438 842 1279 954">You specify the operators that are selectable in the condition for this form element type in the <i>operators</i> object. An operator definition always utilises the schema</p> <p data-bbox="438 983 1279 1014"><operator-name>: {<i>values: [], freeField: true, useChildren: false</i>}.</p> <p data-bbox="438 1043 1279 1155">The name of the operator is also used as the translation ID for user interface internationalisation (see Section 6.4.7, “User interface internationalisation”).</p> <p data-bbox="438 1184 1279 1296">In the <i>values</i> attribute, you can specify a string array containing values that can be selected by the form author when defining a condition.</p> <p data-bbox="438 1326 1279 1482">If you specify the attribute <i>freeField: true</i>, this lets form authors enter user-defined values. This option is required for comparison operators, for example, where form authors need to enter their own comparison values.</p> <p data-bbox="438 1512 1279 1624">If the new field type is a list type with predefined options, you can specify the attribute <i>useChildren</i> if you want to make the list options selectable as a value for the condition.</p>

6.4.2. Adding a new validator

To add a new validator to an input field, extend the *format* property of the corresponding input element.

The example below shows the configuration of the email validator for the single-line text field (*inputField*).

```

{
  title: 'format',
  type: 'dropdown_format',

```

```

properties: {
  options: {
    email: {
      enabled: true,
      fields: {
        errormessage: {
          title: 'errormessage',
          type: 'text'
        }
      }
    }
  }
}

```

The specified attribute name (*email* in the example) must match the external name of the validator. The name is also used for user interface internationalisation. In the translation file, the translation ID `<validator-name>Validator` is used to search for a label for the validator.

You can use *fields* to define the required fields for the validator. The available field types are listed in the table under Section 6.4.5, “Input elements for element properties”.

6.4.3. Adding a new action

Extend the Form Editor to include a new action by extending the configuration *actions_custom.js*.

If you want to extend the Editor to include the action *simpleMailAction* with the properties *to*, *subject*, *body* and *note*, for example, then add the following object definition to the JSON array in the configuration *actions_custom.js*.

```

[
  {
    icon: 'simplemailaction',
    type: 'simpleMailAction',
    properties: {
      general: [
        {
          title: 'to',
          type: 'text',
          properties: {
            required: true
          }
        },
        {
          title: 'subject',
          type: 'text',
          properties: {
            required: true
          }
        }
      ]
    }
  }
]

```

```

        {
            title: 'body',
            type: 'wysiwyg'
        },
        {
            title: 'note',
            type: 'wysiwyg'
        },
    ],
    specialProperties: {
        condition: {
            conditionable: false,
            operators: {}
        }
    }
}
]

```

The table below describes the possible attributes that an action definition can have at the first level.

Attribute	Description
icon	Type: <i>String</i> Name of the icon to load. The name specified must match the filename of the icon without the file extension.
type	Type: <i>String</i> Name of the field type.
properties	Type: <i>Object</i> Describes the properties of an action that can be edited in ContentCreator on the right-hand side, under <i>Properties</i> . The object properties of <i>properties</i> each correspond to individual tabs. The following JSON snippet creates two tabs with the name <i>general</i> and <i>special</i> , with a total of three properties: <i>to</i> , <i>subject</i> and <i>hint</i> . <pre> properties: { general: [{ title: 'to', type: 'text', properties: { required: true } }, { title: 'subject', type: 'text', properties: { </pre>

Attribute	Description
	<pre> required: true } }], special: [{ title: 'hint', type: 'text' }] } </pre> <p>For a list of all available property types, please see Section 6.4.5, “Input elements for element properties”.</p>
specialProperties	<p>Type: <i>Object</i></p> <p>You use the <i>specialProperties</i> attribute to configure properties that are evaluated by the Editor for internal functions.</p> <pre>specialProperties: { maxCount: 1 }</pre> <p>You use <code>maxCount: <count></code> to specify how many times the action can be used within a form.</p>

6.4.4. Adding new element properties

Element properties are defined under the *properties* attribute of the parent form element definition (see Section 6.4.1, “Adding a new form element”). You add a new property to the form element (form field, action or validator) by specifying a JSON object with the following structure.

```

{
  title: '<attribute-name>',
  type: '<field-type>',
  value: 'DefaultValue',
  properties: {
    required: true
  }
}

```

The following table describes the attributes of the configuration object.

Attribute	Description
title	Name used to store the field property in the form definition.
type	Property type. The available types are explained in the following list.
value	Optional specification of a default value.
properties	Other type-specific configuration options

Attribute	Description
properties.required	Specifies whether the property is a required field.

6.4.5. Input elements for element properties

The following table describes the configuration objects for the input elements of the available element properties. You can use these when defining the various form element properties. Please note that some types cannot be used with all form elements.

Type	Description
text	<p>Text field</p> <p>Usage: all form elements</p> <pre>{ title: 'label', type: 'text' }</pre>
number	<p>Number field that only allows numeric input.</p> <p>Usage: all form elements</p> <pre>{ title: 'maxlength', type: 'number', properties: { min: 0, max: null } }</pre> <p>Also supports scientific number notation (e.g. <i>10e6</i>).</p> <p>You can set upper/lower limits by specifying <i>properties.min</i> and <i>properties.max</i>. You can reset an existing limit by using <i>properties.min: null</i>, for example.</p>
date	<p>Data selection element.</p> <p>Usage: all form elements</p> <pre>{ title: 'from', type: 'date' }</pre> <p>You can define a default value with <i>value</i>. Uses the standard JavaScript date format.</p>
checkbox	Checkbox

Type	Description
	<p>Usage: all form elements</p> <pre data-bbox="470 315 1279 456"> { title: 'requiredField', type: 'checkbox' } </pre>
dropdown	<p>List with fixed options from which the form author can select a single entry.</p> <p>Usage: all form elements</p> <pre data-bbox="470 645 1279 891"> { title: 'pattern', type: 'dropdown', properties: { options: ['dd.MM.yyyy', 'yyyy-MM-dd'] } } </pre> <p>You can specify the selection options as a string array with <code>properties.options</code>.</p> <pre data-bbox="470 1025 1279 1238"> properties: { options: [{text: 'Value 1', value: 'one'}, {text: 'Example Two', value: two}] } </pre> <p>Options can also be defined as objects with <i>value</i> (option value) and <i>text</i> (display name).</p>
dropdown_format	<p>Drop-down list for field validators</p> <p>Usage: Input element (<i>inputField</i>, <i>passwordField</i>, etc.)</p> <p>During selection, the properties of the selected validator are shown underneath the drop-down list.</p> <p>The following example illustrates the definition of the email validator.</p> <pre data-bbox="470 1686 1279 2016"> { title: 'format', type: 'dropdown_format', properties: { options: { email: { enabled: true, fields: { errorMessage: { title: 'errorMessage', </pre>

Type	Description
	<pre data-bbox="466 248 1279 488"> type: 'text' } } } </pre> <p data-bbox="466 510 1061 584">You can specify the selectable validators with <code>properties.options</code>.</p> <p data-bbox="466 611 1126 685">You can manage the validator properties with <code>properties.options["<validator-name>"].fields</code>.</p> <p data-bbox="466 712 1246 869">You use <code>properties.options["<validator-name>"].enabled=true</code> or <code>properties.options["<validator-name>"].enabled=false</code> to activate the validator or to deactivate it so that it is no longer selectable.</p>
syntax	<p data-bbox="466 884 1182 920">Multi-line JavaScript input field with syntax highlighting.</p> <p data-bbox="466 943 794 978">Usage: all form elements</p> <pre data-bbox="466 1010 1279 1191"> { title: 'script', type: 'syntax', value: 'function calculate() {};' } </pre>
condition	<p data-bbox="466 1209 987 1245">Input element for processing conditions.</p> <p data-bbox="466 1267 687 1303">Usage: <i>condition</i></p> <pre data-bbox="466 1335 1279 1644"> { title: 'conditionContent', type: 'condition', properties: { conditional_fields: [], condition_conjunction: 'true', condition: [] } } </pre>
wysiwyg	<p data-bbox="466 1662 1230 1736">Multi-line text input field that allows formatting syntax to be used.</p> <p data-bbox="466 1758 794 1794">Usage: all form elements</p> <pre data-bbox="466 1825 1279 1973"> { title: 'value', type: 'wysiwyg' } </pre>
element	<p data-bbox="466 1986 1198 2022">Allows the selection of other elements in the same form.</p>

Type	Description
	<p>Usage: all form elements</p> <pre data-bbox="472 309 1276 454"> { title: 'elements', type: 'element' } </pre>
dataSource	<p>Input element for a data source's variable parameter list.</p> <p>Usage: inputField, comboBox, radioGroup, checkBoxGroup, hiddenField</p> <pre data-bbox="472 645 1276 887"> { title: 'datasource', type: 'dataSource', properties: { datasource_params: [] } } </pre>
reference	<p>Element for selecting a FirstSpirit reference (opens the FirstSpirit selection screen).</p> <p>Usage: all form elements</p> <pre data-bbox="472 1077 1276 1319"> { title: 'pictureUrl', type: 'reference', properties: { FS_refType: 'picture' } } </pre> <p>The property <i>properties.FS_refType</i> gives you the option of restricting the selection to a certain type of content (Page, Picture, Folder, etc.). The following items can be specified here: <i>pageref</i>, <i>picture</i>, <i>file</i>.</p>
multi_dropdown	<p>List with fixed options from which the form author can select multiple entries.</p> <p>Usage: all form elements</p> <pre data-bbox="472 1693 1276 1912"> { title: 'numberType', type: 'multi_dropdown', properties: { configuration_name: 'phoneNumberTypes' } } </pre>

Type	Description
	<p>You can define the selection options (as <i>strings</i>) with the <i>properties.options</i> attribute.</p> <p>Alternatively, the values can also be taken from the paragraph style sheet. In this case, specify the name of the corresponding GOM element in the <i>configuration_name</i> attribute.</p> <p>The <i>configuration_name</i> parameter specifies the name that is used to store the value in the form definition.</p>
regEx_dropdown	<p>Drop-down list for regular expressions.</p> <p>Usage: <i>regex</i></p> <pre data-bbox="470 703 1279 1247"> { title: 'mailPattern', type: 'regEx_dropdown', properties: { options: [{ text: '^[\+]{0,1}[0-9\\s-/*]*\$', label: 'phone' }, { text: '^[a-zA-ZÁ-ÿöüÖÄÜß\\s-]*\$', label: 'characters' }] } } </pre>
mediastore_mapping	<p>Element for the assignment of data upload fields to directories in the FirstSpirit media store.</p> <p>Usage: <i>mediaStoreAction</i></p> <pre data-bbox="470 1429 1279 1610"> { title: 'mediastore_mapping', type: 'mediastore_mapping', value: '[]' } </pre>
field_mapping	<p>Element for selecting a PDF template (opens a FirstSpirit selection screen).</p> <p>Usage: <i>pdfAction</i></p> <p>A PDF field from the template can then be assigned to the form fields. If the fields are drop-down lists, their options can be selected and assigned to one another.</p> <p>Usage: <i>pdfAction</i></p> <pre data-bbox="470 1989 1279 2022"> { </pre>

Type	Description
	<pre> title: 'field_mapping', type: 'field_mapping', value: '[]' } </pre>
datasource_mapping	<p>You can use this input element to assign form fields to the columns of a FirstSpirit data source.</p> <p>Usage: <i>dataSourceAction</i></p> <pre> { title: 'datasource_mapping', type: 'datasource_mapping', value: '[]' } </pre>
custom_mapping	<p>You can use this input element to design your own mapping tables.</p> <pre> { title: 'custom_table', type: 'custom_mapping', properties: { mapping: [{ type: 'dropdown', name: 'field', placeholder: 'custom_table.field', selectableFieldTypes: ['inputField', 'radioGroup'] }, { type: 'dropdown', name: 'option', placeholder: 'custom_table.option', connectedField: 'field' }, { type: 'text', name: 'otherValue', placeholder: 'custom_table.otherValue' }, { type: 'dropdown', name: 'attribut', loadRemoteData: 'FS_ServiceField', loadRemoteDataOptions: [{ name: 'connectedField', key: 'task', value: 'someDropdown' }] }] } } </pre>

Type	Description
	<pre data-bbox="472 248 1262 981"> name: 'connectedMapField_type', key: 'type', value: 'field' }, { name: 'connectedMapField', key: 'value', value: 'field' },], placeholder: 'custom_table.attribut', }, { type: 'dropdown', name: 'attributoption', placeholder: 'custom_table.attributoption', loadRemoteData: 'FS_ServiceField', connectedField: 'attribut', },], }, }, </pre> <p data-bbox="464 1003 1286 1120">You define the columns in the <i>properties.mapping</i> attribute. You use the <i>type</i> key to decide whether this is a selection field (drop-down list) or an input field (text).</p> <p data-bbox="464 1142 1286 1220">The <i>name</i> key sets the key for the export of the respective fields in a row.</p> <p data-bbox="464 1243 1286 1321">You use the <i>placeholder</i> key to define the placeholder for the field.</p> <p data-bbox="464 1344 1286 1456">You use the <i>loadRemoteData</i> key to decide, as you can with a drop-down field, if the options should be provided by a FirstSpirit service.</p> <p data-bbox="464 1478 1286 1758">If the options are provided by a FirstSpirit service, you can use the <i>loadRemoteDataOptions</i> key to pass additional attributes, such as values of other fields on the element, for example, or from the mapping itself. The following example creates this object, so as to pass it with the <i>loadRemoteData</i> call. <code>{task: <ValueOfFieldsomeDropdown>, type: <TypeVonFormElementAusgewähltInField>, value: '<ValueVonMappingDropdownField>'}</code></p> <p data-bbox="464 1780 1286 1859">If you want to select existing form fields in a drop-down, you can pass these by using the <i>selectableFieldTypes</i> key.</p> <p data-bbox="464 1881 1286 2000">If you want to access nested options from a form field or access <i>loadRemoteData</i> from the options themselves, then you can use <i>connectedField</i> and specify the name of a</p>

Type	Description
	mapping field to access these and make them available for selection.

6.4.6. Editing existing form elements

To modify an existing form element, you copy its full element definition from the corresponding default configuration (*fields_default.js* or *actions_default.js*) into the corresponding configuration file (*fields_custom.js* or *actions_custom.js*).

You can then change or add to the element properties according to your requirements as has been described above.

Please note: Changes made to the default configurations in the development workspace have no effect on the Form Editor.

6.4.7. User interface internationalisation

For the internationalisation of the user interface, the language-dependent labels are read from master language files. Out of the box, Formcentric supports the languages English and German.

To modify or add labels for existing or new form elements, you need to extend or modify the *formeditor_de.json* and *formeditor_en.json* language files, which you will find in the development workspace.

Each label is stored in the language files with a unique translation ID. Typically, the translation IDs of the element properties are each made up of the internal element name and the name of the respective property. For the *placeholder* property of the password field, the entry is as follows:

```
"passwordField.placeholder": "Placeholder"
```

You can add a label for a new element property by adding the corresponding entry to each language file.

6.5. Extending the Spring MVC web application

6.5.1. Spring configurations

The web app module itself is a Spring-based web application. Most of the functionality is encapsulated in specialised beans, which are instantiated and initialised by using Spring's dependency injection mechanism. By swapping out individual beans, you can augment the application with new or modified functionality. In general, however, you will merely need to override individual methods of existing beans.

Out of the box, the Spring MVC web application includes the configurations as described below, which are stored in the web application's */WEB-INF/spring* directory.

formcentric-application.xml

This is used to aggregate the various Spring XML files and also for configuration of the *PropertySourcesPlaceholderConfigurer*. This offers a quick overview of all of the relevant Spring files that are required for the web application.

formcentric-actions.xml

Configures the required actions. Ensure that the action beans listed here are also entered into the action mapping (see below). The required properties are read from several property files and configured appropriately. If you would like to change values, you will need to modify these files accordingly.

```
<bean name="mailAction" class="com.formcentric.actions.mail.MailAction">
  <property name="mailer" ref="mailer" />
  <property name="successView" value="success" />

  <!-- Mapping of format identifiers to MailBodyRenderer. -->
  <property name="bodyRendererMapping">
    <map>
      <entry key="html" value-ref="htmlBodyRenderer"/>
      <entry key="text" value-ref="textBodyRenderer"/>
      <entry key="freetext" value-ref="freeTextBodyRenderer"/>
      <entry key="freehtml" value-ref="freeHtmlBodyRenderer"/>/>
    </map>
  </property>
</bean>
...

```

formcentric-captcha.xml

The open source JCAPTCHA framework is used to generate captchas. The configuration here is a standard JCAPTCHA configuration, which can be used to influence the appearance and behaviour of individual captchas. For a detailed description of configuration options, please visit the project website at <https://jcaptcha.atlassian.net/wiki/display/general/Home>.

formcentric-services.xml

Configures the REST services. Enter the *RestService* beans listed here into the REST controller's service mapping as additional entries (see the section called "formcentric-controllers.xml").

```
<bean id="deCountriesRestService"
  class="com.formcentric.rest.CountriesRestService">
  <property name="lang" value="de"/>
</bean>

<bean id="enCountriesRestService"
  class="com.formcentric.rest.CountriesRestService">
  <property name="lang" value="en"/>
</bean>

```

formcentric-controllers.xml

Configures the form controller, the REST controller and the *FormCommandBeanFactory* that is used to create the *FormCommandBean*. The *FormCommandBean* calls the configured initialiser, validators and actions, and generates the form model.

You configure new actions, validators and form elements using the *CommandBean* factory:

```
<bean id="defaultFormCommandBeanFactory"
      class="com.formcentric.logicbeans.DefaultFormCommandBeanFactory">

  <property name="formElementClassMapping">
    <map>
      <entry key="inputField" value="java.lang.String"/>
      <entry key="passwordField" value="java.lang.String"/>
      <entry key="hiddenField" value="java.lang.String"/>
      <entry key="textArea" value="java.lang.String"/>
      <entry key="radioGroup" value="java.lang.String[]"/>
      <entry key="comboBox" value="java.lang.String[]"/>
      <entry key="checkboxGroup" value="java.lang.String[]"/>
      <entry key="fileUpload" value="com.formcentric.model.FileHolder"/>
      <entry key="captcha" value="java.lang.String"/>
    </map>
  </property>

  <property name="validatorMapping">
    <map>
      <entry key="notempty" value-ref="notemptyValidator"/>
      <entry key="jcaptcha" value-ref="captchaValidator"/>
      <entry key="email" value-ref="emailValidator"/>
      <entry key="date" value-ref="dateValidator"/>
      <entry key="number" value-ref="numberValidator"/>
      <entry key="javascript" value-ref="javascriptValidator"/>
      <entry key="regex" value-ref="regexValidator"/>
      <entry key="length" value-ref="lengthValidator"/>
      <entry key="zipcode" value-ref="zipCodeValidator"/>
      <entry key="phone" value-ref="phoneValidator"/>
      <entry key="password" value-ref="passwordValidator"/>
      <entry key="bic" value-ref="bicValidator"/>
      <entry key="iban" value-ref="ibanValidator"/>
      <entry key="file" value-ref="fileValidator"/>
      <entry key="equal" value-ref="equalValidator"/>
    </map>
  </property>

  <property name="actionMapping">
    <map>
      <entry key="mailAction" value-ref="mailAction"/>
      <entry key="compositeAction" value-ref="compositeAction"/>
      <entry key="dataSourceAction" value-ref="dataSourceAction"/>
      <entry key="mediaStoreAction" value-ref="mediaStoreAction"/>
      <entry key="pdfAction" value-ref="pdfAction"/>
      <entry key="datastoreAction" value-ref="datastoreAction"/>
      <entry key="redirectAction" value-ref="redirectAction"/>
    </map>
  </property>
</bean>
```

```

    </map>
  </property>
</bean>

<bean id="formController"
  class="com.formcentric.controllers.FormController">

  <property name="prefix" value="/form" />
  <!-- form controller specific part -->
  <property name="formCommandBeanFactory"
    ref="defaultFormCommandBeanFactory" />
  <property name="bindOnNewForm" value="true" />
  <property name="commandNamePrefix" value="command" />
  <property name="restoreInitialPage" value="true" />
  <property name="trimSpaces" value="false" />
  <property name="supportedMethods" value="POST" />
</bean>

<bean id="restController"
  class="com.formcentric.controllers.RestController">

  <property name="prefix" value="/rest" />
  <property name="commandNamePrefix" value="command" />

  <!-- Service-Mapping -->
  <property name="restServiceMapping">
    <map>
      <entry key="Länder" value-ref="deCountriesRestService"/>
      <entry key="Countries" value-ref="enCountriesRestService"/>
    </map>
  </property>
</bean>

```

formcentric-resourcebundle.xml

Adds the Formcentric resource bundle to the existing *messageSource* bean.

```

<bean id="messageSource"
  class="org.springframework.context.support.ResourceBundleMessageSource">
  <property name="basenames">
    <list>
      <value>com.formcentric.resourcebundle.messages</value>
    </list>
  </property>
</bean>

```



Individual Formcentric messages can be overwritten by using a project-specific resource bundle. We recommend including this bundle in the list of *basenames* at a position before the Formcentric resource bundle. For further information, please see the Spring configuration Javadoc.

formcentric-views.xml

Generates two new Spring *ViewResolver* instances for resolving the JSP views and the bean views.

```
<bean id="jspViewResolver"
      class="com.formcentric.view.JspViewResolver">

  <property name="prefix" value="/WEB-INF/templates/" />
  <property name="viewNames" value="webform,default,optin,exit,success,
    pdf,error,methodNotSupported,uploadSizeExceeded,sessionExpired,
    invalidLicense"/>
</bean>

<bean id="jsonView" class="com.formcentric.view.JsonView"/>

<bean id="fileDownloadView"
      class="com.formcentric.view.FileDownloadView"/>

<bean id="fileInfoView" class="com.formcentric.view.FileInfoView"/>

<bean id="thumbnailView" class="com.formcentric.view.ThumbnailView">
  <property name="thumbnailWidth" value="90"/>
  <property name="thumbnailHeight" value="90"/>
  <property name="crop" value="true"/>
</bean>

<bean id="beanViewResolver"
      class="com.formcentric.view.BeanViewResolver">
  <property name="order" value="1"/>
  <property name="views">
    <map>
      <entry key="json" value-ref="jsonView"/>
      <entry key="file" value-ref="fileDownloadView"/>
      <entry key="fileInfo" value-ref="fileInfoView"/>
      <entry key="thumbnail" value-ref="thumbnailView"/>
      <entry key="captcha" value-ref="captchaView"/>
    </map>
  </property>
</bean>
...
```

formcentric-mail.xml

You use this configuration to specify the mail server connection settings. The configured *mailer* bean is used by the Mail action. If you want to change the underlying properties of the *mailer* bean, you will find all of the values you require in the *formcentric-mail.properties* file.

```
<bean id="mailer" class="com.formcentric.mail.SpringMailer">
  <constructor-arg ref="mailProperties"/>
</bean>
```



The configuration settings for the mailer bean are read with the help of a *PropertiesFactoryBean* from a separate property file, generated by Server and Project Configuration (see Section 4.4.2, “Configuration”).

formcentric-network.xml

Configures the *HttpClient*, which is used to download required web resources such as the PDF template documents. Environment-neutral configuration parameters can be found in the *formcentric-network.properties* file.

```
<!-- Proxy configuration -->
<bean id="proxy" class="com.formcentric.http.ProxyConfig">
  <property name="host" value="{proxy.host}"/>
  <property name="port" value="{proxy.port}"/>
  ...
</bean>

<!-- HttpClient configuration -->
<bean id="httpClient" class="com.formcentric.http.HttpClientFacade">
  <!-- Browser identification string -->
  <property name="userAgent" value="Mozilla/5.0 Firefox/26.0"/>

  <!-- Proxy configuration -->
  <property name="proxyConfig" ref="proxy" />
  ...
</bean>
```

formcentric-connection.xml

You use this configuration to specify the FirstSpirit server connection settings. The configured *ConnectionFactory* bean is used by the FirstSpirit action. The factory can be modified by setting properties in the associated *formcentric-connection.properties* file.

```
<bean id="firstSpiritConnection"
  class="com.formcentric.firstspirit.DisposableConnection" >
  <property name="protocol" value="{connection.transport.protocol}" />
  <property name="host" value="{connection.host}" />
  <property name="port" value="{connection.port}" />
  <property name="user" value="{connection.user}" />
  <property name="password" value="{connection.password}" />
  <property name="useHttps" value="{connection.https:false}"/>
</bean>
```



The configuration settings for the *ConnectionFactory* bean are read with the help of a *PropertyPlaceholderConfigurer* from a separate property file, generated by Server and Project Configuration (see Section 4.4.2, “Configuration”).

formcentric-analytics.xml

You use this configuration to specify the connection settings to the Formcentric Backend. The configured *BackendApiClient* bean is used by the Analytics action, the *BackendFormStateStore* and the *TrackingCommandBean*.

For authentication against the Formcentric Backend, an *access token* is required. If this token should be generated automatically at runtime, the *ClientSecretCredentialsAuthProvider* can be used. This requires the client secret that was issued during Backend configuration. Alternatively, you can create the token manually and use the *ApplicationAuthProvider*. For more information about generating an access token, please see the Installation Manual for the Formcentric Backend.

You can configure the associated properties individually in the *formcentric-analytics.properties* file.

```
<!--
  METHOD 1: Use a pre-generated token.
-->
<bean id="analyticsAuthenticator"
      class="com.formcentric.backend.api.auth.ApplicationAuthProvider">
  <constructor-arg index="0" value="${analytics.apiAuthentication}" />
</bean>

<!--
  METHOD 2: Pass the full client credentials and request a token at runtime.
-->
<bean id="analyticsAuthenticator" class="com.formcentric.backend
    .api.auth.WebClientSecretCredentialsAuthProviderformsCredentialsAuthProvider">
  <constructor-arg index="0" value="${analytics.backendUrl}" />
  <constructor-arg index="1" value="${analytics.apiAuthentication}" />
</bean>

<bean id="analyticsApiClientBuilder"
      class="com.formcentric.backend.api.ApiClientBuilder">
  <constructor-arg index="0" value="${analytics.backendUrl}" />
  <constructor-arg index="1" ref="analyticsAuthenticator" />
</bean>

<bean id="fcBackendApiClient"
      factory-bean="analyticsApiClientBuilder" factory-method="backendApiClient" />
```

6.5.2. Usage without Formcentric Analytics

The Spring configuration files *formcentric-actions.xml*, *formcentric-controllers.xml* and *formcentric-analytics.xml* contain components that can only be used when Formcentric Analytics is deployed. If you want to use Formcentric without Analytics, then the following beans and bean references must be removed from the Spring configuration files listed:

formcentric-actions.xml: *datastoreAction*

formcentric-controllers.xml: *formStateStore* (*BackendFormStateStore*), *defaultTrackingCommandBean*, *trackingController*

formcentric-analytics.xml: None of the beans in this file are required.

6.5.3. Formcentric licence file

The *formcentric-license.xml* file configures the LicenseLoader from Formcentric. You use the corresponding *formcentric-license.properties* to specify the path to the licence file.

Example (Linux/Unix): */path/to/formcentric-license*

Example (Windows): *C:/path/to/formcentric-license*



Paths that do not start with a / are resolved relative to the web app.

6.5.4. Web security

Formcentric contains a security servlet filter as a safeguard against cross-site scripting (XSS) attacks and cross-site request forgery (XSRF) attacks. This filter removes illegal HTML tags, CSS and scripts from the form data submitted. The filter also checks to confirm that the form data contains a valid XSRF token.

As a safeguard against XSRF attacks, each form can be given an additional XSRF token as a hidden parameter: this is then submitted to the web application along with the normal form data. The security filter verifies that the token submitted matches the token stored in the user's session. If this is the case, the request is forwarded to the web application. If not, a 401 error message is returned to the calling client and the failed access is logged in the web application log using the *warn* log level with the following information:

- URL accessed
- Form data submitted (POST parameter)
- IP address of the accessing client
- Fully-qualified name of the accessing client or the last proxy used

The following example shows you how to insert the XSRF token into the form document's output template:

```
<%@ taglib prefix="form"
    uri="http://www.springframework.org/tags/form"%>
<%@ taglib prefix="fcs"
    uri="http://www.formcentric.com/web-security-1.0" %>
<%@ taglib prefix="c"
    uri="http://java.sun.com/jsp/jstl/core" %>

<c:url value="/servlet/form/${self.uid}?view=ajax" var="ajaxUrl"/>
<form:errors path="command${self.uid}" cssClass="error" />

<div id="ajaxreplace${self.uid}" class="ajax_box">
```

```

<form:form name="${self.uid}" commandName="command${self.uid}"
           enctype="multipart/form-data">

  <!-- include XSRF token&#8211;->
  <fcs:xsrftoken/>
  ...
</form:form>
</div>

```

In addition to the form template, the XSRF token must also be inserted into all URLs that reference a Formcentric controller. Currently, these are *FormController*, *FileUploadController* and *RestController*.

The following example shows you how to insert the XSRF token in the *fileUpload.jsp* template into the upload URL as a URL parameter.

```

<%@ taglib prefix="form"
      uri="http://www.springframework.org/tags/form"%>
<%@ taglib prefix="fcs"
      uri="http://www.formcentric.com/web-security-1.0" %>
<%@ taglib prefix="c"
      uri="http://java.sun.com/jsp/jstl/core" %>

<!-- Upload-URL zusammenbauen und in Variable speichern -->
<c:url value="/servlet/upload" var="uploadUrl">
  <c:param name="_uid" value="${form.uid}"/>
  <c:param name="_lang" value="${form.lang}"/>
  <fcs:xsrftokenParam method="POST"/>
</c:url>

<c:set var="id" value="${self.id}_${form.uid}"/>

<div class="mwf-upload"
      data-mwf-fileupload='{
        "url": "${uploadUrl}",
        "id": "${id}",
        "name": "${self.name}",
        "autoUpload": ${self.properties['auto_upload']},
        "labels": ${rowLabels},
        "previewMaxWidth": "90",
        "previewMaxHeight": "90"
      }'>
  ...

```

The upload URL is created with the help of the JSP `<c:url>` tag. You use the `<mwf:xsrftokenParam>` tag to add an XSRF token parameter to the URL. As with the JSP tag `<c:param>`, this can be used together with the `<c:url>` tag (see the section called "fcs:xsrftokenParam").

You configure the security filter in the web application's deployment descriptor (web.xml). In the mapping rules, specify all of the Formcentric controllers for which the filter should be used.

```

<filter>
  <filter-name>webSecurityFilter</filter-name>
  <filter-class>com.formcentric.security.
    SecurityServletFilter</filter-class>
  <init-param>
    <param-name>xsrpPrevention</param-name>
    <param-value>>true</param-value>
  </init-param>
  <init-param>
    <param-name>xsrpMethods</param-name>
    <param-value>POST</param-value>
  </init-param>
  <init-param>
    <param-name>xsrpSessionBased</param-name>
    <param-value>>true</param-value>
  </init-param>
  <init-param>
    <param-name>xsrpTokenName</param-name>
    <param-value>_mwfToken</param-value>
  </init-param>
  <init-param>
    <param-name>xssPrevention</param-name>
    <param-value>>true</param-value>
  </init-param>
</filter>

<filter-mapping>
  <filter-name>webSecurityFilter</filter-name>
  <url-pattern>/servlet/form/*</url-pattern>
</filter-mapping>

<filter-mapping>
  <filter-name>webSecurityFilter</filter-name>
  <url-pattern>/servlet/rest</url-pattern>
</filter-mapping>

<filter-mapping>
  <filter-name>webSecurityFilter</filter-name>
  <url-pattern>/servlet/upload</url-pattern>
</filter-mapping>

```

The servlet filter has the following configuration settings.

Parameter	Description
xsrpPrevention	Enables XSRF protection (true, false).
xsrpMethods	Comma-separated list of HTTP methods (GET, POST) that should be secured using an XSRF token.
xsrpSessionBased	Use this parameter to specify whether the XSRF token should be renewed on each page reload (true) or just once per user session (false).
xsrpTokenName	Base name of the request parameter in which the current XSRF token will be passed. In the default

Parameter	Description
	configuration, the name <i>com.formcentric.XSRFToken</i> is used. The base name is automatically extended by the ID of the associated form.
xssPrevention	Enables XSS protection (true, false).



The security servlet filter can be configured only once per web application.

6.5.5. Saving the form status

As standard, all of the data entered by the user is saved in the user session on the server. If forms are complex, however, the session may expire before the user has finished completing and sending the form. In this case, the data items stored in the session are lost.

Formcentric therefore offers you the option of saving the entered data for a longer period of time. This means users can take a break from form entry and continue filling out the form at a later point in time.

To activate this function, configure a *FormStateStore* in the *formcentric-controllers.xml* Spring configuration file.

```
<bean id="defaultFormCommandBeanFactory"
  class="com.formcentric.logicbeans.DefaultFormCommandBeanFactory">
  ...
  <property name="formStateStore" ref="formStateStore"/>
</bean>
```

Formcentric offers you two separate store implementations:

BackendFormStateStore

This implementation stores the form data in the Analytics Backend database. For each data record, a unique ID is generated and stored both in the database and in a cookie file. The cookie's lifetime, domain and path can be specified in the Spring configuration.

```
<bean id="formStateStore"
  class="com.formcentric.store.BackendFormStateStore">

  <property name="cookiePath" value="/" />
  <property name="cookieDomain" value="my-domain.com" />
  <property name="cookieMaxAge" value="604800" />
  <property name="backendClient" ref="backendClient" />
</bean>
```

FileFormStateStore

This implementation stores the form data in an encrypted file on the server. The associated file name is stored in a cookie. The cookie's directory, encryption password, lifetime, domain and path can all be specified in the Spring configuration.

```
<bean id="formStateStore"
  class="com.formcentric.store.FileFormStateStore">

  <property name="cookiePath" value="/" />
  <property name="cookieDomain" value="my-domain.com" />
  <property name="cookieMaxAge" value="604800" />
  <property name="secret" value="change-this-now" />
  <property name="storageDir" value="/var/webforms" />
</bean>
```

The user account used to start the application server must possess write permissions for this directory. If not otherwise specified, the directory configured in the system variable *java.io.tmpdir* is used. In a clustered environment, the directory must also be accessible to all instances of the web application.

6.5.6. Implementing an action

As already described, the business logic for form data processing is encapsulated by actions in the web application. From a technical perspective, these are classes that implement the *com.formcentric.actions.Action* interface. Additional business beans can be injected into an action via Spring. In this way, data access objects (DAOs) can be made available in order to access external databases, for example. The actions are injected into the form controller via Spring when the application starts.

Variable action parameters, which must be entered by the form author when creating the form (such as the target address for the mail action), are passed to the action implementation via the properties map of the *ActionNode* bean.

The following example shows you how you can implement and configure the *CustomAction* described in section Section 6.3.1, "Developing a *NodeEditorPane*".

```
public class CustomAction extends BaseAction<WebForm> {

  public static final String PROP_CUSTOM = "customProperty";

  @Override
  public ModelAndView execute(ExecutionContext<WebForm> context, Map<String,
    Object> formData) throws Exception {

    WebForm formDefinition = context.getFormDefinition();
    ActionNode action = context.getAction();

    String customParam = action.getPropertyAsString(PROP_CUSTOM);

    // Business logic
    ...
  }
}
```



```

// ModelAndView
ModelAndView mv = new ModelAndView("success");
mv.addObject(Constants.ATTRIBUTE_SELF, formDefinition);
return mv;
}

@Override
public boolean isExecutable(ExecutionContext<WebForm> context,
    Map<String, Object> formData) throws Exception {
    return true;
}
}

```

When called via the *execute* method, the action is given all of the available data. In addition to the actual form data, *ExecutionContext* passes the form definition, the action definition, the form variables (see Section 6.5.7, “Adding variables for pre-filling form fields”) and the request object. The *parameters* map contains only the values of the visible form elements. Access to all form data is provided by calling the method *getRawFormData()* on the *ExecutionContext* bean.

The *execute* method must return an object of the *ModelAndView* type. This is used in order to present the results page, as shown to the user after data has been submitted.

Typically, the *ModelAndView* object is generated with the form bean and a specialised view (such as *success*).

In addition, however, there is also the option of redirecting the request to another page. In this case, the *ModelAndView* object can be generated as follows:

```
ModelAndView mv = ControllerUtils.redirectTo(renderBean, viewName);
```

The *renderBean* object and the view name can be generated by the specialised business logic of the action implementation.

In some application scenarios, errors in the input data are discovered only during processing by the associated backend. In this case, the user should not be presented with the results page but should be given the form again, along with an error message. To achieve this, the action – in the same way as with validators – should create an error on the *Errors* bean passed in the *ExecutionContext*.

```

public ModelAndView execute(ExecutionContext<WebForm> context, Map<String,
Object> formData) throws Exception {
    ...
    context.getErrors().rejectValue("username", DUPLICATE_USER_ERROR,
        "That user name is already being used.");

    ModelAndView mv = new ModelAndView("success");
    mv.addObject("self", formDefinition);
    return mv;
}

```

Enter the action into the Spring configuration *formcentric-actions.xml*:

```
<bean name="customAction" class="com.custom.forms.web.CustomAction">
```

```

    <!-- Required properties -->
    ...

</bean>

```

In the Spring configuration *formcentric-controllers.xml*, also add the following to the action mapping:

```

<bean id="formCommandBeanFactory"
    class="com.formcentric.logicbeans.DefaultFormCommandBeanFactory">

    <!-- Mapping of action names to action implementations -->
    <property name="actionMapping">
        <map>
            <entry key="mailAction" value-ref="mailAction"/>
            <entry key="customAction" value-ref="customAction"/>
        </map>
    </property>
    ...
</bean>

```

6.5.7. Adding variables for pre-filling form fields

To pre-fill form fields, the form author can make use of a range of predefined variables. As standard, the form author can use the variables *date*, *time*, *language*, *ip*, *remoteUser*, *principal*, *userAgent* and *referer*.

To provide custom variables, you need to override the method *getVariables()* on the *FormCommandBean*.

```

public class CustomFormCommandBean extends DefaultFormCommandBean {

    @Override
    protected Map<String, Object> getVariables(HttpServletRequest request,
        WebForm formDefinition) {

        Map<String, Object> variables =
            super.getVariables(request, formDefinition);

        // Add your variables to the result map
        ...

        return variables;
    }
}

```

To instantiate the new *CustomFormCommandBean*, you will also need to override the *FormCommandBean* factory and enter this into the configuration *formcentric-controllers.xml*.

```

public class CustomCommandBeanFactory extends DefaultFormCommandBeanFactory {

```

```

@Override
public CustomFormCommandBean createBeanFor(WebForm formDefinition) {

    CustomFormCommandBean commandBean = new CustomFormCommandBean();
    initCommandBean(commandBean, formDefinition);

    return commandBean;
}
}

```

Replace the *DefaultFormCommandBeanFactory* in the Spring configuration *formcentric-controllers.xml* with the *CustomCommandBeanFactory*:

```

<bean id="formCommandBeanFactory"
      class="com.custom.forms.web.CustomCommandBeanFactory"
      ...

```



The form fields are initialised with the predefined pre-filled values once only, when the form is called for the first time. This also replaces the variables with their values. Accordingly, subsequent changes to variable values are not applied to an already-initialised form.

6.5.8. Implementing a REST service

Formcentric includes a REST interface, which you can use to fill drop-down lists or input fields at runtime with data from external systems. The data concerned can be static, dynamic or specific to the user. All of the interface's specialised functions are encapsulated in classes of the *com.formcentric.rest.RestService* type. By implementing your own REST service, you can extend the interface to include additional functionality. The following example shows you a REST service that generates a map with static key/value pairs.

```

public class CustomRestService extends BaseRestService {

    @Override
    public Object invoke(ServiceContext<WebForm> context, Map<String,
        Object> remoteFormData, Map<String, Object> localFormData) {

        String myCustomParam =
            context.getConfigParameterMap().get("myCustomParam");
        ...

        HashMap<String, String> data = new HashMap<String, String>();

        // Fill the map
        data.put("key1", "value1");
        data.put("key2", "value2");
        data.put("key3", "value3");

        return data;
    }
}

```

```
}  
}
```

By calling the *invoke* method, the *RestService* is passed both the *ServiceContext* as well as the user input already sent (*remoteFormData* parameter) and the user input not yet sent (*localFormData* parameter). This enables you to react directly to user input, regardless of whether or not this input has already been sent.

The *ServiceContext* gives you access to the form definition, the input element, the configuration parameters for the *RestService* and the request object.

Enter the REST service into the Spring configuration *formcentric-services.xml*:

```
<bean name="customRestService"  
      class="com.custom.forms.web.CustomRestService">  
  
  <!-- Required properties -->  
  ...  
  
</bean>
```

Also add the service to the REST controller's service mapping in the Spring configuration file *formcentric-controllers.xml*:

```
<bean id="restController"  
      class="com.formcentric.controllers.RestController">  
  <property name="prefix" value="/rest" />  
  <property name="commandNamePrefix" value="command" />  
  
  <property name="restServiceMapping">  
    <map>  
      <entry key="Example" value-ref="exampleRestService"/>  
      ...  
    </map>  
  </property>  
</bean>
```

The service is accessed via the URL:

```
<context-path>/servlet/rest?_service=Example&_uid=<Dokument-ID>&  
  _input=<Input-Name>
```

The following JSON string is returned as the response to this call:

```
[  
  {  
    "k": "key1",  
    "v": "value1",  
    "i": "mwf6aab0bb24033",  
    "h": "8d0c3e13950d86c1a7383f066105f78c"  
  },  
  {  
    "k": "key2",  
    "v": "value2",
```

```

    "i": "mwf06a7a0930d37",
    "h": "d22d445101243a5f616cfd64c765e399"
  },
  {
    "k": "key3",
    "v": "value3",
    "i": "mwf1674ffb0a121",
    "h": "c0ad1fa77bb1b79ca757ee1ffce9f416"
  }
]

```

To prevent manipulation of the JSON data so transmitted, the individual key/value pairs are secured using an additional hash value that is validated on the server during form submission.

This security mechanism means that no calls may be made to external REST services, since their data does not contain the required hash values. If you need to access external services, however, you can implement your own proxy REST service, which in turn accesses the external REST service.

From version 2.3 of Formcentric onwards, REST service calls within JSP templates are made using the HTML attribute *data-mwf-datasource*. In the attribute value, you must specify a JSON object that contains the URL of the REST service, the usage type (*checkbox*, *radio*, *selection*, *suggestion* or *hidden*) and any other parameters.

As standard, you can specify a REST service for the following input elements:

inputField:

```

<c:url value="/servlet/rest" var="restUrl">
  <c:param name="_service" value="${self.properties['datasource']}" />
  <c:param name="_uid" value="${form.uid}" />
  <c:param name="_input" value="${self.name}" />
  <fcs:xsrftokenParam />
</c:url>

<c:set var="params" value="${self.properties['datasource_params']}" />

<form:input data-mwf-id="${self.id}"
  data-mwf-datasource='{
    "type" : "suggestion",
    "url" : "${restUrl}",
    "data" : {},
    "params" : ${params}
  }' ... />

```

hiddenField:

```

<c:url value="/servlet/rest" var="restUrl">
  <c:param name="_service" value="${self.properties['datasource']}" />
  <c:param name="_uid" value="${form.uid}" />
  <c:param name="_input" value="${self.name}" />
  <fcs:xsrftokenParam />
</c:url>

```

```

<c:set var="params" value="${self.properties['datasource_params']}" />

<form:hidden path="${self.name}" data-mwf-id="${self.id}"
    data-mwf-datasource='{
        "type" : "hidden",
        "url" : "${restUrl}",
        "data" : {},
        "params" : ${params}
    }' ... />

```

comboBox:

```

<![CDATA[<c:url value="/servlet/rest" var="restUrl">
    <c:param name="_service" value="${input.properties['datasource']}" />
    <c:param name="_uid" value="${form.uid}" />
    <c:param name="_input" value="${input.name}" />
    <fcs:xsrftokenParam />
</c:url>

<fc:valueOut var="userValue" name="${input.name}" />
<c:set var="strUserValue" value="${fn:join(userValue, ', ')}" />
<c:set var="params" value="${input.properties['datasource_params']}" />

<form:select data-mwf-id="${self.id}"
    data-mwf-datasource='{
        "type" : "selection",
        "url" : "${restUrl}",
        "preselected" : "${strUserValue}",
        "data" : {},
        "params" : ${params}
    }' ... >
    ...
</form:select>

```

checkboxGroup:

```

<c:url value="/servlet/rest" var="restUrl">
    <c:param name="_service" value="${input.properties['datasource']}" />
    <c:param name="_uid" value="${form.uid}" />
    <c:param name="_input" value="${input.name}" />
    <fcs:xsrftokenParam />
</c:url>

<fc:valueOut var="userValue" name="${input.name}" />
<c:set var="strUserValue" value="${fn:join(userValue, ', ')}" />
<c:set var="params" value="${input.properties['datasource_params']}" />

<fieldset data-mwf-id="${self.id}"
    data-mwf-datasource='{
        "name" : "${input.name}",
        "type" : "checkbox",
        "url" : "${restUrl}",
        "preselected" : "${strUserValue}",
        "data" : {},

```

```

        "params" : ${params}
    }' ... >
    ...
</fieldset>

```

radioGroup:

```

<c:url value="/servlet/rest" var="restUrl">
  <c:param name="_service" value="${self.properties['datasource']}" />
  <c:param name="_uid" value="${form.uid}" />
  <c:param name="_input" value="${self.name}" />
  <fcs:xsrftokenParam />
</c:url>

<fc:valueOut var="userValue" name="${self.name}" />
<c:set var="strUserValue" value="${fn:join(userValue, ', ')}" />
<c:set var="params" value="${self.properties['datasource_params']}" />

<fieldset data-mwf-id="${self.id}"
  data-mwf-datasource='{
    "name" : "${self.name}",
    "type" : "radio",
    "url" : "${restUrl}",
    "preselected" : "${strUserValue}",
    "data" : {},
    "params" : ${params}
  }' ... >
  ...
</fieldset>

```



Since the double quotation mark must be used within the JSON string, you must use the single quotation mark for the HTML attribute.

As described previously, both the form input that has been sent and the form input not yet sent is available to you within the RestService. As one example of how to use this function, you could implement a RestService that takes a postcode entered by the user and returns a drop-down list of locations matching the postcode.

In this example, it would be advisable to update the drop-down list automatically if the user changes the postcode, since other locations may be referenced by the changed postcode. This can be achieved by using the parameter *dependsOn*. In the form editing interface, this can be entered into the parameter list of a RestService (see also section 3.2.5 in the User Manual). The value to be entered here must specify the name of the input element on which the result of the selected RestService depends. Every change made to one of the input elements specified results in another call to the RestService.

6.5.9. Template development

The output of the forms and form elements is handled by FreeMarker or JSP templates within the web application.

In the standard configuration, Formcentric is set up to use JSP templates. To reconfigure the web app to use FreeMarker templates, you need to import the *formcentric-views-freemarker.xml* configuration into the Spring configuration *formcentric-views.xml* instead of the *formcentric-views-jsp.xml* configuration.

At the data level (model), all form elements are represented by an object of the *com.formcentric.model.xml.InputNode* type. To access the properties *id*, *type*, *name*, *label*, *value*, *parent* and *children*, you can use the corresponding getter methods on the *InputNode* bean. Access to all other properties is performed using the properties map from the *InputNode* bean.

The following table shows you all of the form element types and their properties. The properties shown in square brackets must be read from the properties map.

Element	Properties
form	name, [style_class, form_style_class, next_label, submit_label, cancel_label, script, description, save_state, save_statistics, doi_to, doi_note, doi_message, doi_from, doi_sender, doi_subject, doi_enabled, doi_format, doi_condition, doi_condition_conjunction]
inputField	name, label, value, [hint, placeholder, field_width, style_class, readonly, maxlength, datasource, datasource_params]
shortText	name, label, value, [hint, placeholder, field_width, style_class, readonly, maxlength]
emailField	name, label, value, [hint, placeholder, field_width, style_class, readonly]
numberField	name, label, value, [hint, placeholder, field_width, style_class, readonly, maxlength]
dateField	name, label, value, [hint, placeholder, field_width, style_class, readonly, maxlength]
phoneField	name, label, value, [hint, placeholder, field_width, style_class, readonly, maxlength]
textArea	name, label, value, [hint, placeholder, field_width, style_class, readonly, maxlength, rows, cols]
passwordField	name, label, [hint, placeholder, field_width, style_class]
button	name, label, [hint, field_width, style_class, onclick]
checkBoxGroup	name, label, children, [pictureUrl, pictureFileName, hint, field_width, style_class, datasource, dynamic, datasource_params]
comboBox	name, label, value, children, [pictureUrl, pictureFileName, hint, field_width, style_class, datasource, dynamic, datasource_params]

Element	Properties
picture	name, [pictureUrl, pictureFileName, alt, field_width, style_class]
pageBreak	name, label, [style_class, condition, next_label, back_label, script]
paragraph	name, value, [bold, italic, field_width, style_class]
captcha	name, label, [field_width, hint]
radioGroup	name, label, children, [pictureUrl, pictureFileName, hint, field_width, style_class, datasource, dynamic, datasource_params]
summary	label, [style_class, elements, hide_empty_fields]
hiddenField	name, value, [datasource, datasource_params]
fileUpload	name, [multiple, auto_upload, hint, field_width, style_class]
condition	[condition, condition_conjunction, conditional_fields]
pageCondition	[condition, condition_conjunction, next_page, script]
layout	label, [layout]
fieldSet	name, label, [style_class]
calculatedValue	name, label, [script, visible, clientside, style_class]
mailAction	[subject, to, cc, bcc, from, sender, body, format, note, replyto, send_hidden_fields, condition, condition_execute, condition_conjunction, redirect_url, hide_empty_fields]
dataSourceAction	[note, schema, table, datasource_mapping, commit_message, release, condition, condition_execute, condition_conjunction]
mediaStoreAction	[note, mediastore_mapping, commit_message, release, condition, condition_execute, condition_conjunction]
pdfAction	[note, pdfFileName, pdfPath, pdfUid, pdfUrl, field_mapping, linktext, readonly, condition, condition_execute, condition_conjunction]
datastoreAction	[note, condition, condition_execute, condition_conjunction]
redirectAction	[note, condition, condition_execute, condition_conjunction, url, content, delay]
webhookAction	[note, condition, condition_execute, condition_conjunction, url, fields, url_parameters, custom_headers, content_type]
sequenceAction	–

In addition to the InputNode beans described above, the system passes other objects in the request to the form templates. The following table gives you an overview of all objects passed.

Parameter name	Type	Description
self	com.formcentric.model.WebForm	Current form document bean
input	com.formcentric.model.xml.InputNode	Current form element bean
pageElements	java.util.List	List with the elements of the current form page
pageCount	java.lang.Integer	Number of form pages
form	com.formcentric.model.WebForm	Form definition
currentPage	java.lang.Integer	Page number of the current form page
currentPageNode	com.formcentric.model.xml.InputNode	Current form page bean
formdata	java.util.Map	Map containing the form data entered by the user

FreeMarker templates

The FreeMarker templates are stored in the directory */WEB-INF/templates/ftl*. Alongside the parent form template, each form element type is also assigned its own template. The example below shows you the *textArea.ftl* template for the multi-line text input field:

```
<li data-mwf-container="${self.id}"
    class="mwf-field ${self.properties['style_class']}!"!""">
  <label class="mwf-label" for="${self.id}">${self.label!"!"""}
    <#if self.required><span class="mwf-required">*</span></#if>
  </label>
  <div class="mwf-input">

    <@spring.bind mwf.bind(self)/>
    <#assign hasErrors=spring.status.error />
    <textarea id="${self.id}"
      class="mwf-text ${self.properties['style_class']}!"!"""}
      name="${spring.status.expression!"!"""}"
      ${self.properties['readonly']?boolean?then("readonly", "")}
      maxlength="${self.properties['maxlength']}!"!"""}
      spellcheck="true"
      placeholder="${self.properties['placeholder']}!"!"""}
      rows="${self.properties['rows']}!"!"""}
      cols="${self.properties['cols']}!"!"""}
      data-mwf-id="${self.id}">${spring.status.value!"!"""}</textarea>
    <#if self.properties['hint']?has_content>
      <div class="mwf-hint">
```

```

        <@fc.markdown value=(self.properties['hint']!"")?string />
    </div>
</#if>
<@spring.showErrors separator="<p>" classOrStyle="mwf-error"/>
</div>
</li>

```

FreeMarker functions and macros

Formcentric provides you with a FreeMarker library that contains specialised functions for displaying the forms.

To utilise these functions, insert the following instruction into the FreeMarker templates:

```
<#import "/lib/formcentric.com/webforms.ftl" as mwf>
```

The following section gives you a description of the functions contained in this library.

mwf.forEachPageElement

List function that contains the elements on the current page.

```

forEachPageElement(boolean layoutFacets, boolean removeEmptyFacets,
    final String exclude, final String include)

```

Parameter	Description
layoutFacets	If this value is set to <i>true</i> , the elements will be split across layoutFacets (optional).
removeEmpty-Facets	Specifies whether empty layouts should be ignored when creating the list (optional). Default value: <i>false</i>
exclude	Comma-separated list of the element types that should be ignored when creating the list. If nothing is specified here, then all element types – with the exception of excluded types – are included (optional).
include	Comma-separated list of element types that should be included when creating the list (optional).

```

<#list mwf.forEachPageElement(true, false, "condition", "") as layout>
    <ul class="{layout.properties['layout']!""}">

        <#if layout_index == 0 && currentPage == 0 && self.label?has_content>
            <li class="mwf-field"><h3>{self.label!""}</h3></li>
        </#if>

        <#list layout.items as input>
            <@fc.include self=input view=input.type />
        </#list>

```

```
</ul>
</#list>
```

mwf.forEachPage

List function that returns a list of the collected pages.

```
forEachPage(final boolean compact)
```

Parameter	Description
compact	Specifies whether form pages with the same title should be consolidated together (optional). Default value: <i>false</i>

mwf.include

Macro that includes a bean with a certain template.

```
<@fc.include self=input view=input.type />
```

Attribute	Description
self	Bean that is included.
view	Name of the template to be included.
params	An extended hash that you can use to pass additional parameters to the included template (optional).

mwf.url

Function that can be used to generate absolute URLs on the Formcentric controller.

If nothing is specified in the *baseUrl* parameter, then the generated URL points to the Formcentric web app in which the FreeMarker template was called.

This function is helpful if you want to install the Formcentric web application on a host separate to the one hosting the surrounding web page, since in this case you are unable to use relative URLs.

Parameter	Description
base	Specifies the Formcentric controller.
params	An extended hash that you can use to pass additional parameters to the URL (optional).
baseUrl	Fully-qualified hostname, such as <i>https://your.domain.com:8000</i> (optional).

```
<#assign restUrl=fc.url("/servlet/rest", {
```

```

    "_uid": form.uid,
    "_input": self.name,
    "_service": self.properties['datasource'],
    fc.xsrfTokenName(form.uid): fc.xsrfTokenValue(form.uid)
  }) />

```

mwf.responseHeader

Macro that you can use to set a header in the response.

```

<@fc.responseHeader name="Content-Type"
  value="text/html; charset=UTF-8"/>

```

mwf.summary

Function that returns a list of all elements as a *com.formcentric.model.InputBean* for the form. This can also be used to query the data entered by the user.

This can be used to query the following properties:

name	Form element name
label	Form element label.
type	Form element type.
object	Form element value bean.
value	String representation of the value bean.
valueLabels	String array containing the labels of the options chosen in the selection field (<i>comboBox</i> , <i>radioGroup</i> , <i>checkboxGroup</i>). If the associated input element is not a selection, then the value of the element is returned in the array.
page	Number of the page on which the element is located.
pageLabel	Label of the page on which the element is located.
layout	Name of the layout in which the element is located.
input	<i>InputNode</i> of the element.

```

summary(InputNode self, String elements,
  final String include, final String exclude,
  final boolean hideEmptyFields, final String excludeIfEmpty)

```

Parameter	Description
self	<i>InputNode</i> of a form element. If this value is set, then the iteration is interrupted at the specified element.
elements	Comma-separated list containing the names of the form elements that should be shown in the summary. If this

Parameter	Description
	attribute contains a value, then the attribute <i>self</i> is ignored (optional).
include	Comma-separated list of element types that should be considered during iteration. If nothing is specified here, then all element types – with the exception of excluded types – are included (optional).
exclude	Comma-separated list of element types that should be ignored during iteration (optional). Default value: <i>button, hiddenField, condition, pageCondition, pagebreak, captcha, passwordField</i>
hideEmptyFields	Specifies that all empty fields should be ignored. Default value: <i>false</i>
excludeEmpty-Fields	Specifies that empty fields should be ignored (optional). Default value: <i>false</i>

```
<#list mwf.summary(self, self.getPropertyAsString('elements'),
  self.getPropertyAsBoolean('hide_empty_fields', false)) as item>
  <tr>
    <#if item.input.type == "paragraph">
      <td colspan="2">
        <@fc.markdown>${item.input.value}</@fc.markdown>
      </td>
    <#else>
      <td>${item.label?has_content?then(item.label, item.name!"")}</td>
      <td>${(item.valueLabels![])?join(", ")}</td>
    </#if>
  </tr>
</#list>
```

mwf.captcha

Template that you can use to generate a captcha image.

Attribute	Description
url	URL of the captcha servlet.
id	ID of the captcha InputNode.
linkClass	CSS class(es) that is/are applied to the link to the captcha image (optional). Default value: ""
imgClass	CSS class(es) that is/are applied to the captcha image (optional). Default value: ""

Attribute	Description
title	The title attribute for the captcha image (optional). Default value: ""
alt	The alt attribute for the captcha image (optional). Default value: <i>Captcha</i>

```
<#assign captchaUrl=mwf.url("/servlet/captcha/captcha.jpg")!"" />
<@fc.captcha url=captchaUrl id=self.id linkClass="css-class__link"
  imgClass="css-class__img" title="A title" alt="Captcha" />
```

mwf.ifCaptcha

Boolean function that evaluates whether the captcha *name* has **not** been entered correctly.

Parameter	Description
name	Name of the captcha element.

```
<#if fc.ifCaptcha(self.name! "")>
  <#assign captchaUrl=mwf.url("/servlet/captcha/captcha.jpg")!"" />
  <@fc.captcha url=captchaUrl id=self.id />
</#if>
```

mwf.getStandardButton

Function that supplies the standard form button determined by the *buttonType* attribute.

Parameter	Description
buttonType	Standard button type. Can be set to the following values: <ul style="list-style-type: none"> <code>_next</code> Button that takes the user to the next page of the form. <code>_back</code> Button that takes the user to the previous page of the form. <code>_cancel</code> Button that cancels form data entry. <code>_finish</code> Button that submits the form. <code>_exit</code> Button that can be used to exit from the form.

```
<#assign finishButton=mwf.getStandardButton("_finish") />
<li data-mwf-container="{finishButton.id}" class="mwf-button mwf-next">
  <input type="button" value="{submitLabel}"
    data-mwf-submit='{"type": "finish",
      "query": "navigationId={cmpage.navigation.contentId}"}' />
```

```
</li>
```

mwf.valueOut

Function that can be used to output the current value of a form field.

```
valueOut(String name, boolean preferLabel)
```

Parameter	Description
name	Form field name
preferLabel	Specifies that the value's label should be output instead of the value itself (optional).

```
${fc.valueOut(self.name!"", true)!""}
```

mwf.conditions

Function with which the JavaScript definitions can be generated for conditional elements. Place this function at the end of the form template.

```
<#assign conditions=mwf.conditions() />
```

mwf.calculatedValues

Function that generates the JSON definitions for the *calculated values*.

```
<#assign calculatedValues=mwf.calculatedValues() />
```

mwf.markdown

Macro that can be used to output a value interpreted using markdown. This macro can handle both a passed value (see *value* parameter) or a body.

Parameter	Description
value	Value to be interpreted using markdown (optional).
inline	Specifies whether the output HTML should be restricted to inline elements (optional). Default value: <i>false</i>

```
<@fc.markdown value=self.value!"" inline=false />  
<!-- or -->  
<@fc.markdown inline=false>${self.value!""}</@fc.markdown>
```

mwf.vars

Macro that can be used to replace variables from the form context in the output.

Parameter	Description
map	Map with the form data.

```
<@fc.vars map=formdata>${action.properties['note']!""}</@fc.vars>
```

mwf.bind

Function that returns the path to which the node is bound.

Parameter	Description
node	InputNode whose path is being queried.

```
<@spring.bind mwf.bind(self) />
```

mwf.encodeUrl

Function that encodes the URL passed in UTF-8.

Parameter	Description
url	URL that is to be encoded.

mwf.hasValidator

Function that checks whether the validator specified by the *name* parameter is present in the InputNode *node*.

Parameter	Description
node	InputNode that is to be evaluated.
name	Name of the validator.

mwf.validatorByName

Function that returns the validator specified in the *name* parameter of the InputNode *node*.

Parameter	Description
node	InputNode that is to be evaluated.
name	Name of the validator.

mwf.elementByName

Function that returns the InputNode specified in the *name* parameter for the form *form*.

Parameter	Description
form	Form that contains the InputNode.
name	Name of the element.

Security library

Formcentric includes a security library for the generation and output of XSRF tokens (see Section 6.5.4, “Web security”). This security library is also included by the integration of the FreeMarker functions.

The FreeMarker functions that are included are described below.

mwf.xsrfToken

Macro that generates a hidden form field with an XSRF token.

```
<@fc.xsrfToken />
```

mwf.xsrfTokenName

Function that generates an xsrfTokenName from the form ID.

Parameter	Description
formId	ID of the form for which the token should be generated. If this parameter is empty, the form ID passed in the request is used (optional).

```
<#assign restUrl=mwf.url("/servlet/rest", {"_uid": form.uid, ...,  
    "tokenName": fc.xsrfTokenName(), "tokenValue": fc.xsrfTokenValue()})!"/>
```

mwf.xsrfTokenValue

Function that generates an xsrfTokenValue from the form ID.

Parameter	Description
formId	ID of the form for which the token should be generated. If this parameter is empty, the form ID passed in the request is used (optional).

```
<#assign restUrl=mwf.url("/servlet/rest", {"_uid": form.uid, ...,  
    "tokenName": fc.xsrfTokenName(), "tokenValue": fc.xsrfTokenValue()})!"/>
```

JSP templates

The JSP templates are stored in the directory */WEB-INF/templates/jsp*. Alongside the parent form template, each form element type is also assigned its own template. The example below shows you the *textArea.jsp* template for the multi-line text input field:

```
<%@ page contentType="text/html; charset=UTF-8" language="java"%>  
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>  
<%@ taglib prefix="spring" uri="http://www.springframework.org/tags" %>  
<%@ taglib prefix="form" uri="http://www.springframework.org/tags/form" %>
```

```

<jsp:useBean id="self" type="com.formcentric.model.xml.InputNode"
  scope="request"/>
<jsp:useBean id="form" type="com.formcentric.model.WebForm"
  scope="request"/>

<c:set var="id" value="${self.id}_${form.uid}"/>
<mf:hasErrors var="hasErrors" path="${self.name}"/>
<li data-mwf-container="${id}" class="mwf-field">
  <label class="mwf-label" for="${id}">
    <c:out value="${self.label}"/>
    <c:if test="${self.required}">
      <span class="mwf-required">*</span>
    </c:if>
  </label>
  <div class="mwf-input">
    <form:textarea id="${id}" path="${self.name}"
      cssClass="mwf-text ${self.properties['field_width']}"
      readOnly="${self.properties['readonly']}"
      maxLength="${self.properties['maxLength']}"
      spellcheck="true"
      rows="${self.properties['rows']}"
      cols="${self.properties['cols']}"
      data-mwf-id="${id}"
      placeholder="${self.properties['placeholder']}"/>
    <c:if test="${not empty self.properties['hint']}">
      <p class="mwf-hint">
        <small><c:out value="${self.properties['hint']}"></small>
      </p>
    </c:if>
    <form:errors path="${self.name}" cssClass="mwf-error" element="p"/>
  </div>
</li>

```

Taglib webfoms-1.0

Formcentric provides you with a tag library that contains specialised tags for displaying the forms.

To utilise this taglib, insert the following instruction into the JSP template:

```

<%@taglib prefix="fc" uri="http://www.monday-consulting.com/webfoms-1
  .0"%>

```

The following section gives you a description of the tags contained in this library.

mwf:forEachPageElement

Iterator tag, which you can use to iterate over the individual elements in a form page. In the process, the current form element is passed in the control variable.

Attribute	Description
var	Name of the control variable that contains the current form element.

Attribute	Description
varStatus	Iteration status.
exclude	Comma-separated list of element types that should be ignored during iteration. If nothing is specified here, then all element types – with the exception of excluded types – are considered.
include	Comma-separated list of element types that should be considered during iteration.
layoutFacets	<p>Specifies whether form elements should be grouped by layout for output. In this case, the control variable contains the current layout. Note that this is a bean of the type <i>PageIterateTag.LayoutFacet</i>. This can be used to query the following properties:</p> <p>layout Layout descriptor</p> <p>items List containing form elements assigned to this layout.</p> <p>Default value: <i>false</i></p>
removeEmpty-Facets	<p>Specifies whether empty layouts should be ignored during iteration.</p> <p>Default value: <i>false</i></p>

```
<mwf:forEachPageElement var="input"
  exclude="pageBreak,calculatedValue,condition">
  <mwf:include self="${input}" view="${input.type}"/>
</mwf:forEachPageElement>
```

```
<mwf:forEachPageElement var="layout" varStatus="s" layoutFacets="true"
  exclude="pageBreak,calculatedValue,condition">

  <c:choose>

    <!-- Single-column layout -->
    <c:when test="${empty layout.layout or layout.layout eq '1'}">
      <fieldset class="onecolumn">
        <c:forEach items="${layout.items}" var="input" varStatus="i">
          <mwf:include self="${input}" view="${input.type}"/>
        </c:forEach>
      </fieldset>
    </c:when>

    <!-- Multi-column layout -->
    <c:when test="${not empty layout.layout and layout.layout eq '2'}">
      <fieldset class="twocolumn">
        <c:forEach items="${layout.items}" var="input" varStatus="i">
          <mwf:include self="${input}" view="${input.type}"/>
        </c:forEach>
      </fieldset>
    </c:when>

  </c:choose>
```

```
</c:choose>
</mwf:forEachPageElement>
```

mwf:forEachPage

Iterator tag, which you can use to iterate over the pages in a form. The page title and other items of status information are passed in the control variable.

Attribute	Description
var	Name of the control variable, which contains a bean of the type <i>ForEachPageTag.Item</i> . This can be used to query the following properties: <ul style="list-style-type: none"> label Form page label index Page number selected Specifies whether the associated form page is currently being shown. finished Specifies whether the associated form page has already been filled out.
varStatus	Iteration status.
compact	Specifies whether form pages with the same title should be consolidated together. Default value: <i>false</i>

```
<mwf:forEachPage var="page" varStatus="status">
  <c:set var="class" value="${entry.selected ? 'selected' : ''}"/>
  <span class="${class}">
    <c:out value="${status.index}"/>. <c:out value="${page.label}"/>
  </span>
</mwf:forEachPage>
```

mwf:summary

Tag that you can use to iterate over all of the elements in a form. Alongside the element properties *name*, *type* and *label*, the value entered by the user is also passed in the control variable.

Attribute	Description
var	Name of the control variable, which contains a bean of the type <i>com.formcentric.model.InputBean</i> . This can be used to query the following properties: <ul style="list-style-type: none"> name Form element name label Form element label

Attribute	Description
	<p>type Form element type</p> <p>object Form element value bean</p> <p>value String representation of the value bean</p> <p>valueLabels String array containing the labels of the options chosen in the selection field (<i>comboBox</i>, <i>radioGroup</i>, <i>checkboxGroup</i>). If the associated input element is not a selection, then the value of the element is returned in the array.</p> <p>page Number of the page on which the element is located.</p> <p>pageLabel Label of the page on which the element is located.</p> <p>layout Name of the layout in which the element is located.</p> <p>input <i>InputNode</i> of the element.</p>
varStatus	Iteration status.
elements	Comma-separated list containing the names of the form elements that should be shown in the summary. If this attribute contains a value, then the attribute <i>self</i> is ignored.
self	<p><i>InputNode</i> of a form element.</p> <p>If this value is set, then the iteration is interrupted at the specified element.</p>
include	Comma-separated list of element types that should be considered during iteration. If nothing is specified here, then all element types – with the exception of excluded types – are considered.
exclude	<p>Comma-separated list of element types that should be ignored during iteration.</p> <p>Default value: <i>button, hiddenField, condition, pageCondition, pagebreak, captcha, passwordField</i></p>
hideEmptyFields	<p>Specifies that all empty fields should be ignored.</p> <p>Default value: <i>false</i></p>
excludelfEmpty	Enter the types of form fields here that should be ignored if they are empty.

```
<table>
  <fc:summary exclude="hiddenField" var="item"
    elements="${self.properties['elements']}"
    hideEmptyFields="${self.properties['hide_empty_fields']}">
```

```

<tr>
  <td><c:out value="\${empty item.label ? item.name : item.label}"/></td>
  <td><c:out value="\${fn:join(item.valueLabels, ', ')}"/></td>
</tr>
</fc:summary>
</table>

```

mwf:captcha

Tag that you can use to generate a captcha image.

Attribute	Description
url	URL of the captcha servlet.
alt	Alternative text for the captcha image

```

<mwf:captcha url="/servlet/captcha/captcha.jpg" path="\${self.name}"
  title="Click for new Captcha"/>

```

mwf:captchaLink

Tag that you can use to generate a reload link for the embedded captcha image. The generated link is provided with an *onClick* event handler that replaces the captcha image with a new image when clicked by the user.

Attribute	Description
url	URL of the captcha servlet.

```

<mwf:captchaLink url="/servlet/captcha/captcha.jpg" path="\${self.name}">
  <mwf:captcha url="/servlet/captcha/captcha.jpg" path="\${self.name}"
    title="Click for new captcha"/>
</mwf:captchaLink>

```

mwf:ifCaptcha

Conditional tag whose body is **not** executed if the captcha specified in the *name* attribute has been entered correctly.

Attribute	Description
name	Name of the captcha element.
var	Name of the variable that is used to store a successful captcha entry.

```

<mwf:ifCaptcha name="\${self.name}">
  <mwf:captchaLink url="/jcaptcha/captcha.jpg" path="\${self.name}">
    <mwf:captcha url="/jcaptcha/captcha.jpg" path="\${self.name}"
      title="Click for new captcha"/>
  </mwf:captchaLink>

```

```
</mwf:ifCaptcha>
```

mwf:getStandardButton

Tag that can be used to access the form's standard buttons.

Attribute	Description
var	Name of the variable in which the button is stored.
buttonType	Standard button type. Can be set to the following values: _next Button that takes the user to the next page of the form. _back Button that takes the user to the previous page of the form. _cancel Button that cancels form data entry. _finish Button that submits the form. _exit Button that can be used to exit from the form.

```
<fc:getStandardButton buttonType="_finish" var="finishButton" />  
<li data-mwf-container="{finishButton.id}" class="mwf-button mwf-next">  
  <input type="button" value="{fn:escapeXml(submitLabel)}"  
    data-mwf-id="{finishButton.id}" data-mwf-submit="{type: 'finish'}" />  
</li>
```

mwf:forEachCondition

Tag that you can use to iterate over the individual rules of a condition. During the iteration, the input element, the operator and the value used for comparison are made available in the control variable.

Attribute	Description
var	Name of the control variable, which contains a bean of the type <i>ForEachConditionTag.Rule</i> . This can be used to query the following properties: input Input element to which the condition refers. operator Logical operator for the condition value Value used by the condition for comparison
self	Condition element
varStatus	Iteration status

```
<mwf:forEachCondition self="{self}" var="cond">  
  <c:out value="{cond.input.name}" />  
  <c:out value="{cond.operator}" />  
  <c:out value="{cond.value}" />
```



```
</mwf:forEachCondition>
```

mwf:valueOut

Tag that can be used to output the current value of a form field.

Attribute	Description
name	Form field name
var	Name of the variable in which the value of the field should be stored.
preferLabel	Specifies that the value's label should be output instead of the value itself.
escapeXml	A Boolean flag that specifies whether the standard XML entities, such as "<" or "&", should be converted to their entity codes.

```
<mwf:valueOut name="${input.name}"/>
```

mwf:conditions

Tag that generates the JSON definitions for the conditional elements.

Attribute	Description
var	Name of the variable in which the JSON data should be stored.

```
<mwf:conditions var="conditions"/>
```

mwf:calculatedValues

Tag that calculates the JSON definitions of the *calculated values*.

Attribute	Description
var	Name of the variable in which the JSON data should be stored.

```
<mwf:calculatedValues var="calculatedValues"/>
```

mwf:markdown

Tag that searches through the text passed in the body for markdown syntax and converts this into HTML.

Attribute	Description
inline	Specifies whether the output HTML should be restricted to inline elements (optional).

Attribute	Description
	Default value: <i>false</i>

```
<mwf:markdown inline="false">**This text  
will be output in bold.**</mwf:markdown>
```

mwf:hasGlobalBindErrors

This tag can be used to check whether global errors were identified while validating the command bean specified in the *name* attribute. The HTML code contained within the tag is output only if the validation resulted in global errors. The errors identified are provided in the request scope for further processing in the variable *errors*, which has the variable type *org.springframework.validation.Errors*.

Attribute	Description
name	Name of the command bean

```
<mwf:hasGlobalBindErrors name="command${self.uid}">  
  <ul>  
    <li class="mwf-error">  
      <form:errors cssClass="mwf-error" element="p" />  
    </li>  
  </ul>  
</mwf:hasGlobalBindErrors>
```

mwf:vars

Tag that searches through the text passed in the body for variables in the format *#{name-der-variable}* and replaces them with their corresponding values.

Attribute	Description
map	Map with the variable values (key, value). Please note: A map with the form variables (<i>formVariables</i>) and a map with the form values (<i>formdata</i>) are passed by default in the page scope.
var	Name of the page scope variable in which the filtered body text should be stored.

```
<fc:vars map="${formdata}">  
  <fc:markdown><c:out value="${action.properties['note']}" /></fc:markdown>  
</fc:vars>
```

You can specify dynamic tag attributes to extend the list of variables.

```
<fc:vars map="${formdata}" my-variable="${any-page-scope-variable}">  
  <p>${my-variable}</p>  
</fc:vars>
```

mwf:url

Tag that can be used to generate absolute URLs on Formcentric controllers.

If nothing is specified in the *baseUrl* attribute, then the URL generated points to the Formcentric web app in which the JSP template was called.

This tag is helpful if you want to install the Formcentric web application on a host separate to the one hosting the surrounding web page, since in this case you are unable to use relative URLs.

Attribute	Description
value	Specifies the Formcentric controller.
var	Name of the variable in which the generated URL should be stored.
scope	Scope in which the generated URL is stored. If you do not enter anything here, the page scope is used.
context	Name of a local web application.

```
<fc:url value="/servlet/rest" var="restUrl">
  <fc:param name="_service" value="${self.properties['datasource']}" />
  <fc:param name="_uid" value="${form.uid}" />
  <fc:param name="_input" value="${self.name}" />
  <fcs:xsrftokenParam />
</fc:url>
```

Taglib web-security-1.0

Another tag library is available for generating and issuing XSRF tokens. To utilise this taglib, insert the following instruction into the JSP template:

```
<%@ taglib prefix="fcs"
  uri="http://www.formcentric.com/web-security-1.0"%>
```

The following section gives you a description of the tags contained in this library.

fcs:xsrftoken

Tag that creates a hidden field with an XSRF token.

Attribute	Description
formId	ID of the form for which the XSRF token should be generated. If nothing is entered here, then the ID of the form passed in the <i>from</i> request attribute is used.
varName	Name of the variable in which the XSRF token should be stored. If this attribute contains a value, then no hidden field is created.

```
<fcs:xsrftoken/>
```

fcs:xsrftokenParam

This tag can be used together with the `<c:url>` tag in order to add an XSRF token to the URL as a parameter.

Attribute	Description
formId	ID of the form for which the XSRF token should be generated. If nothing is entered here, then the ID of the form passed in the <i>from</i> request attribute is used.
method	Specifies the HTTP method (GET, POST) used to access the corresponding URL. Default value: <i>GET</i>

```
<c:url value="/servlet/upload" var="uploadUrl">
  <c:param name="_uid" value="${form.uid}"/>
  <c:param name="_lang" value="${form.lang}"/>
  <fcs:xsrftokenParam method="POST"/>
</c:url>
```

6.5.10. JavaScript

Out of the box, Formcentric ships with the JavaScript files as described below. These are stored in the development workspace, in the directory `/formcentric-webapp-customizations/src/main/webapp/js` and the export file `/formcentric-module-customizations/resources_export.zip`.

jQuery-File-Upload

For file uploading, Formcentric uses the Blueimp jQuery-File-Upload plugin. Depending on the browser used, files are either transferred using AJAX or within a hidden iframe. The plugin comprises the following JavaScript files.

- jquery_ui_widget_1_13_2.js
- jquery_iframe_transport.js
- jquery_xdr_transport.js
- load-image-all-min.js
- canvas-to-blob.min.js
- jquery_fileupload_10_31_0.js
- jquery_fileupload_process_10_31_0.js
- jquery_fileupload_image_10_31_0.js

Since some of the JavaScripts have dependencies on one another, they must be loaded in the order specified here.

jquery-autocomplete.js

This JavaScript contains a jQuery plugin that can be used to add autocomplete functionality to input fields. Values for the autocomplete function are loaded asynchronously from the specified REST service.

jquery-format-1.3.js

This JavaScript contains a jQuery plugin that enables the formatting or analysis of dates and numbers. Note that this is a JavaScript alternative to the Java classes *SimpleDateFormat* and *NumberFormat*.

json2.js

Formcentric uses the native *JSON* object supplied by modern browsers to parse and construct JSON objects. For older browsers that do not support the *JSON* object, the object is provided by this JavaScript.

Select2

Formcentric uses the Select2 jQuery plugin to offer a configurable selection field that supports functions such as search, selection, user-defined options and many other features. Once the plugin has been loaded by the browser, all drop-down lists are automatically shown as Select2 selection fields. Select2 offers a wide range of plugin-specific configuration parameters, which you can specify in the JSP or FreeMarker template as a JSON object in the data attribute *data-mwf-select* within the `<form:select>` tag. For further information about the Select2 plugin, see <https://select2.org>.

```
<form:select id="${self.id}"  
  
    <!-- Select2 config -->  
    data-mwf-select='{  
        "placeholder": "${placeholder}",  
        "width": "100%",  
        "tags": "${customInput}"  
    }'  
  
    ...
```

jquery-formcentric-1.9.js

This JavaScript contains a jQuery plugin that provides the JavaScript functions required by Formcentric. As shown below, an instance of the plugin can be generated in the JSP template *webforms.jsp*.

```
<mwf:calculatedValues var="calculatedValues"/>  
<mwf:conditions var="conditions"/>  
  
<script type="text/javascript">
```

```

$('#command${form.uid}').webforms({
  "calculatedValues" : ${calculatedValues},
  "conditions" : ${conditions}
});
</script>

```

When calling the plugin method *webforms*, you have the option of passing a configuration object that may contain the following options.

Option	Description
conditions	Type: <i>JSON</i> JSON definition of the client-side conditions to be evaluated.
calculatedValues	Type: <i>JSON</i> JSON definition of the client-side <i>calculated values</i> to be calculated.
appendUrlVars	Type: <i>Boolean</i> You use this parameter to specify that the URL parameters in the host page will be appended to the AJAX request sent to the form controller.
trimSpaces	Type: <i>Boolean</i> You use this parameter to specify that leading and trailing spaces will be trimmed from the form data.
createOption	Type: <i>Function(\$form, entry, selected)</i> Function that creates an option element in a dynamic drop-down list (<i>comboBox</i>).
createRadio	Type: <i>Function(\$form, name, entry, checked)</i> Function that creates a radio button in a dynamic Radio Button Select field (<i>radioGroup</i>).
createCheckBox	Type: <i>Function(\$form, name, entry, checked)</i> Function that creates a check box button in a dynamic Check Box Select field (<i>checkBoxGroup</i>).
createUploadFileRow	Type: <i>Function(Object \$form, Object attr, file)</i> This function creates a new entry in the file list of the File-Upload element before the file is uploaded.
createDownload-FileRow	Type: <i>Function(Object \$form, Object attr, file)</i> This function creates a new entry in the file list of the File-Upload element after the file has been uploaded.
updateCalculatedValue	Type: <i>Function(\$form, id, value)</i>

Option	Description
	Function that updates the display of a <i>calculatedValue</i> when this has been re-calculated.
updateFormValue	Type: <i>Function(\$form, \$elem, name, l)</i> This function updates the value of a form field in the summary, if this value has been entered or changed by the user. If the corresponding form field is a selection (list) field, the labels of the options selected are passed in the <i>l</i> parameter. Otherwise, the text value entered is passed.
onFillDropdown	Type: <i>Function(Object \$form, Object \$elem)</i> Callback function that is called after a dynamic dropdown list has been filled.
onFillSelection	Type: <i>Function(Object \$form, Object \$elem)</i> Callback function that is called after a dynamic drop-down list has been filled.
onInit	Type: <i>Function(Object \$form)</i> Callback function that is called after the jQuery plugin has been initialised. Use this function to perform your own initialisations.
onSubmit	Type: <i>Function(Object \$form, String url, String query)</i> Callback function that is called when the form is submitted. The form is submitted only if the function returns the Boolean value <i>true</i> .
onSuccess	Type: <i>Function(Object \$form, Object data, String status, Object jqXHR)</i> Callback function that is called after the form has been submitted successfully.
onRedirect	Type: <i>Function(Object \$form, Object data, String status, Object jqXHR)</i> The <i>onRedirect</i> function is a callback function that is called when the server response to the form submission includes the <i>X-Redirect-Location</i> header. If the <i>onRedirect</i> function returns the boolean value <i>true</i> , it indicates that the redirect has already been processed within the <i>onRedirect</i> callback function. If the function returns <i>false</i> or if there is no return, the redirect is automatically carried out to the URL specified in the server response (<i>X-Redirect-Location</i> header).
onAjaxError	Type: <i>Function(jqXHR jqXHR, String status, String error)</i>

Option	Description
	Function that is called when an error occurs during an AJAX request. As standard, this function creates an entry in the browser's error log.
operations.visible	Type: <i>Function(Object \$form, Object field)</i> Function with which input fields can be made visible.
operations.hidden	Type: <i>Function(Object \$form, Object field)</i> Function with which input fields can be hidden.
operations.writeable	Type: <i>Function(Object \$form, Object field)</i> Function with which input fields can be changed from write-protected to editable.
operations.readonly	Type: <i>Function(Object \$form, Object field)</i> Function with which input fields can be changed from editable to write-protected.
operations.enabled	Type: <i>Function(Object \$form, Object field)</i> Function with which input fields can be changed from deactivated to activated.
operations.disabled	Type: <i>Function(Object \$form, Object field)</i> Function with which input fields can be changed from activated to deactivated.
operations.optional	Type: <i>Function(Object \$form, Object field)</i> Function with which input fields can be marked as optional.
operations.mandatory	Type: <i>Function(Object \$form, Object field)</i> Function used to mark input fields as mandatory fields.

The default implementations of the JavaScript functions listed can be found in the JavaScript *jquery-webforms-1.9.js*.

In the following example, the *operations.mandatory* function is replaced by a modified version.

```
<fc:calculatedValues var="calculatedValues"/>
<fc:conditions var="conditions"/>

<script type="text/javascript">
    $('#command${form.uid}').webforms({
        "calculatedValues" : ${calculatedValues},
        "conditions" : ${conditions},
        "operations": {
            "mandatory": function ($form, field) {
                var $label = $form.find('label[for="' + field.input + '"]');
```



```

        $label.children('em').remove();
        $label.append('<em>*</em>', '');
    }
}
});
</script>

```

Event reference

The Formcentric jQuery plugin makes a series of events available that enable you to respond to scenarios that match the various events. The corresponding event handler must be registered on the *document* object.

Detailed kinds of event-dependent information such as the associated form element, for example, are passed to the event handler in the *event.details* event object.

```

document.addEventListener("mwf-fill-selection",
    function(event) {
        console.log(event.detail.$form);
        console.log(event.detail.$elem);
    }
);

```

The following table describes the events that you can monitor and program specific responses to:

Event name	Detailed information	Is sent when
mwf-ajax-finished	<i>\$dest, \$content</i>	the function <i>mwfAjaxReplace</i> has been executed successfully.
mwf-ajax-error	<i>\$dest, jqXHR, textStatus, errorThrown</i>	the asynchronous call (AJAX call) has an error.
mwf-fill-dropdown	<i>\$form, \$elem</i>	a drop-down list has been filled by a data source.
mwf-fill-selection	<i>\$form, \$elem</i>	a radio button or check box select field has been filled by a data source.
mwf-fill-hidden	<i>\$form, \$elem</i>	a hidden field has been filled by a data source.
mwf-suggestion-selected	<i>\$form, \$elem, id, selection, params</i>	an autocomplete item has been selected for an input field.
mwf-value-changed	<i>\$form, \$elem, name, value</i>	the value of a form field has changed.

6.6. Extending the headless web application

The Formcentric headless web application is a Spring Boot application that provides a REST interface with various end points for form processing. For client-side connec-

tivity to the headless application, a ready-to-use React client is provided, which you can also configure to suit your requirements (see also Section 6.7, “Formcentric client”).

The following section describes how to extend the functionality of the headless application. Please note: some of the names of the framework classes are the same as those from the Spring MVC web application but are located outside of the *com.formcentric.headless.rest* package.

6.6.1. Implementing an action

Similarly to the Spring MVC web application, the headless application also uses actions that encapsulate the business logic for the form data processing. These classes implement the interface *com.formcentric.headless.actions.Action*. You can then integrate any backend systems you need to by developing a custom action. These actions are Spring beans: as a result, configuration parameters can be passed to the action by using the standard Spring mechanisms. The following example shows you how to implement and configure a *CustomAction*.

```
import com.formcentric.headless.actions.*;

public class CustomAction extends BaseAction {

    public static final String PROP_CUSTOM = "anyCustomActionPropertyName";

    @Override
    public ActionResult execute(ExecutionContext context, Map<String,
        Object> formData) throws ActionException {

        WebForm formDefinition = context.getFormDefinition();
        ActionNode action = context.getAction();

        String customParam = action.getPropertyAsString(PROP_CUSTOM);

        // Business-Logic
        ...

        ActionResult actionResult = new ActionResult();
        actionResult.setView("success");
        return actionResult;
    }

    @Override
    public boolean isExecutable(ExecutionContext context,
        Map<String, Object> formValues) throws ActionException;
    return true;
}

public String name() {
    return "customAction";
}
}
```

To instantiate and configure your own actions in the headless application, create a configuration class as shown in the following code example. Label the configuration class with the annotation `@Configuration`. In your configuration class, you define a method that instantiates an object of your action class. This method must be annotated with `@Bean` in order to document the fact that this method provides a bean definition.

If your action class has dependencies on other beans, use the method parameters to inject these beans. Use the annotation `@Value` to inject configuration values or pass other beans directly as parameters.

```
package com.example.myapp.config;

import com.example.myapp.actions.MyCustomAction;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

@Configuration
public class MyActionsConfiguration {

    @Bean
    public MyCustomAction myCustomAction() {
        // Configuration and initialization of the MyCustomAction
        return new MyCustomAction();
    }

    // Additional bean definitions can be added here
}
```

If you place your configuration class in the base package `com.formcentric.headless` or a sub-package, Spring Boot will find and load your class automatically. If you use different package names, you must extend the `scanBasePackage` parameter for the `@SpringBootApplication` annotation in the main application class.

In the development workspace, the application class `CustomHeadlessWebApplication` is already created, which you can modify for this purpose.

6.6.2. Adding variables for pre-filling form fields

Alongside the predefined variables, you also have the option of adding your own variables for pre-filling form fields. To do this, register a Spring bean of the `VariablesService` type in the application context of the headless application.

```
import com.formcentric.headless.services.VariablesService;
import com.formcentric.headless.model.WebForm;
import org.springframework.stereotype.Service;
import jakarta.servlet.http.HttpServletRequest;

@Service
public class CustomVariablesService implements VariablesService {
    @Override
    public final Map<String, Object> getVariables(HttpServletRequest request,
```

```

WebForm formDefinition) {

    Map<String, Object> vars = new HashMap<>();

    // Add custom variables to the variables Map
    vars.put("custom_var", "custom_value");
    return vars;
}
}

```

This *VariablesService* bean is a Spring bean, which means you can also access external systems or services when creating the variables.

In some application scenarios, you will need to pre-fill form fields with values from the web page or client application into which the form is embedded. For this use case, it is sufficient to specify the variables in the data attribute *data-fc-vars* from the *div* tag with which the form is associated.

```

<div
  data-fc-id="1249010"
  ...
  data-fc-vars='{ "custom_var": "custom_value" }'
></div>

```

6.6.3. Implementing a REST service

All of the REST services described in section Section 6.5.8, “Implementing a REST service” are also available to you when deploying the headless application. The following example shows you a REST service that generates a map with static key/value pairs.

```

package com.formcentric.headless.examples.rest;

import com.formcentric.headless.rest.BaseRestService;
import com.formcentric.headless.rest.ServiceContext;
import org.jetbrains.annotations.NotNull;
import org.springframework.stereotype.Component;

import java.util.HashMap;
import java.util.Map;

@Component
public class CustomRestService extends BaseRestService {

    @Override
    public Object invoke(ServiceContext context, Map<String, Object> formData) {

        HashMap<String, String> data = new HashMap<>();

        // fill the map
        data.put("key1", "value1");
        data.put("key2", "value2");
        data.put("key3", "value3");
    }
}

```

```

        return data;
    }

    @NotNull
    @Override
    public String name() {
        return "customRestService";
    }
}

```

By calling the *invoke* method, the *RestService* is passed both the *ServiceContext* as well as the current user input (*formData* parameter). This lets you respond directly to user input.

The *ServiceContext* gives you access to the form definition, the input element, the configuration parameters for the *RestService* and the request object.

All form elements, which also includes the REST services, must have their own unique name with which they can be referenced within the form definition. The name is determined when starting the application by calling the method *name()*.

Spring uses the *@Component* annotation to instantiate your REST service automatically and register it using the specified name.

6.7. Formcentric client

The NPM module *@formcentric/client* (<https://www.npmjs.com/package/@formcentric/client>) is required to present Formcentric forms in the browser. This applies both for projects based on HTML only plus JavaScript as well as for projects that utilise frontend frameworks or frontend libraries.

The installed package includes various variants of modules for a wide range of applications. The files required are installed using NPM, which itself has no dependencies, however, and can also be used without any bundlers.

For installation, execute the following command:

```
npm install @formcentric/client
```

Or alternatively:

```
pnpm install @formcentric/client
```

The following items must be present in order for a form to be displayed correctly:

1. A *div* tag with an *fc-id* data attribute, into which the form will be rendered.
2. A loaded theme, consisting of CSS, templates and CSS custom properties, if these are being used in the CSS file.
3. To be able to be embedded as a script tag, *formapp.js* must also be accessible.

4. To be able to be embedded as a link tag, *formcentric_component_style.css* must also be accessible, if internal components like DatePicker or FileUploader are being used.

```
<head>
  {...}
  <link rel="stylesheet" href="/example-url/formcentric_component_style.css"/>
</head>
<div
  data-fc-id="1249010"
  data-fc-formapp-url="/example-url/formapp.js"
  data-fc-theme-url="/example-url/formcentric.css"
  data-fc-template-url="/example-url/formcentric_templates.js"
  data-fc-theme-variable-url="/example-url/formcentric.json"
  data-fc-form-definition="K82AClxH1YpNGtKt ... ffUuAm40yEQsC9"
  data-fc-refs="ffUuAm40yEQsC9 ... 2AClxH1YpNGtKt"
  data-fc-vars='{}'
  data-fc-params='{}'
  data-fc-data-url='https://example-url-to-formcentric-headless-server.com'
></div>
```

```
<script
  src="./formcentric.js"
  defer
></script>
```

6.7.1. Theme

The theme CSS must be loaded to ensure that the form can be displayed correctly. This can be achieved by using a link tag in the HTML head and the use of custom properties. If available, these must be set in the HTML code.

Each input field has its own template, which can be modified. These templates are defined on the *Window* object in a JS file called *formcentric_templates.js*. This ensures that they can be found later when rendering the form. The templates are required in order to present the form correctly (see Section 6.7.3, “Templates”).

6.7.2. Initialisation

To start the client, either the script *formcentric.js* can be loaded or, after this has been loaded, *window.formcentric.initFormcentric()* can be called at a later point in time. You use the data attribute *fc-data-url* to configure the URL for accessing the Formcentric headless server.

6.7.3. Templates

Templates always consist of a function whose return value is used by the Formcentric Client to render HTML code. To achieve this, the Formcentric client passes two parameters (*html* and *props*) to a template function. The exact structure and the parameters used will depend on the specific usage of the template.

html: A template literal tag, which is used to render HTML code. This parameter enables the embedding of HTML into the template's JavaScript code.

props: An object that contains the properties of the form field. These consist of calculated values from the Formcentric Client as well as field data supplied by the Editor. The specific properties vary according to the form field.

The final HTML is created from a combination of static HTML code and the values from the *props* properties. This can be achieved by combining strings together or using functions to render HTML. The resulting HTML created is then passed back as a template function return value to the Formcentric Client, which then renders it in the DOM.



All template functions can also be processed asynchronously by the Formcentric Client as a promise.

Template properties

The Formcentric client passes template properties to the corresponding template function as parameters (*props*). These include all of the information required for the presentation and behaviour of the respective form fields.

The following properties are passed to the templates:

Property	Description
key	The element's unique ID.
oninput	(event: InputEvent) => void Updates the field value.
onfocus	(event: FocusEvent) => void If present, returns any validation errors for the field.
onclick	A function that evaluates the <i>onClick</i> functions defined by the editor.
fieldSuccess	A Boolean value that specifies whether the element was successfully validated and has no errors.
fieldError	An object that contains information about an error in the element.
properties	An object that contains the properties of the field element.
components	An object that contains components for certain field types. These components are used to render the element in the template. The object contains components for <i>captcha</i> , <i>fileUploader</i> , <i>comboBox</i> , <i>suggestions</i> , <i>hint</i> , <i>datePicker</i> and <i>markdown</i> .
fieldSetFields	An array that contains the fields from a <i>FieldSet</i> .

Property	Description
layoutFields	An array that contains the fields from a <i>Layout</i> .
summaryFields	An array that contains the information from fields as specified in a <i>SummaryField</i> .
fieldEmptyText	A piece of boilerplate displayed if the <i>SummaryField</i> contains no values.
contentMarkup	A content component that is returned by a function specified by the <i>form</i> div.
hasService	A Boolean value that specifies whether the element has a REST service.
setRESTParams	(params: Record<string,any>) => void: A function that is used to specify the parameters for the element's REST service.

Element: The properties also contain all of the information about the element to be shown.

```
interface fcElement {
  id: string // ID des Elements
  name: string // Technischer Name des Elements
  type: fcFieldTypes // Elementtyp
  fieldSetId?: string // ID des FieldSets, in dem das Element enthalten ist
  layoutId?: string // ID des Layout Elements, in dem das Element enthalten ist
  label?: string // Label des Elements
  value?: string | string[] // Wert des Elements
  validators?: fcElementValidator[] // Validatoren
  children?: {
    id: string
    type: fcFieldTypes
    name: string
    label?: string
    value?: string | string[]
    checked?: boolean
  }
}
```



```
    properties?: fcProperties
    validators?: fcElementValidator[]
  }[]
  properties?: fcProperties // Properties des Elements (siehe Properties)
}
```

Field types:

```
type fcFieldTypes =
| 'error'
| 'success'
| 'formHeader'
| 'formFooter'
| 'inputField'
| 'button'
| 'form'
| 'layout'
| 'condition'
| 'passwordField'
| 'textArea'
| 'radioGroup'
| 'comboBox'
| 'checkBoxGroup'
| 'fileUpload'
| 'calculatedValue'
| 'hiddenField'
| 'paragraph'
| 'summary'
| 'dateField'
| 'numberField'
```

```
| 'emailField'  
| 'phoneField'  
| 'shortText'  
| 'captcha'  
| 'content'  
| 'option'  
| 'fieldSet'
```

Validators:

```
interface fcElementValidator {  
    id: string  
    name: string  
    properties?: {  
        errorMessage?: string  
        from?: string  
        to?: string  
        days_from?: string  
        days_to?: string  
        pattern?: string  
        max_files?: string  
        max_size?: string  
        file_types?: string  
    }  
}
```

Properties: All HTML attributes and field properties from the form definition are contained in *props.properties*. These are calculated as a result of conditions by the Formcentric Client, for example, or configured by the form author for the corresponding form field. The following table gives an overview of possible properties:

Property	Description
hint	An optional note text that gives the user more information or provides instructions.
placeholder	An optional placeholder text that is displayed in an input field if no value has been entered.

Property	Description
selected	A Boolean value that specifies whether the element is selected by default.
errormessage	An optional error message that is displayed if the element is invalid.
multiple	A Boolean value that specifies whether multiple values can be selected for this element.
auto_upload	A Boolean value that specifies whether an automatic upload function is activated.
datasource	A character string that specifies a data source.
datasource_params	A character string that specifies the parameters for the data source.
dynamic	A Boolean value that specifies whether the element is dynamic and has properties that can be changed at runtime.
visible	A Boolean value that specifies whether the element should be visible.
hidden	A Boolean value that specifies whether the element should be hidden.
writable	A Boolean value that specifies whether the element should be writable.
readonly	A Boolean value that specifies whether the element should be read-only.
optional	A Boolean value that specifies whether the element is optional.
mandatory	A Boolean value that specifies whether the element is required.
disabled	A Boolean value that specifies whether the element should be deactivated.
enabled	A Boolean value that specifies whether the element should be activated.
type	A character string that specifies the element type.

Components

Internal components are provided for selection by *props.components*. This is intended to simplify work with individual form fields if there is no need to modify the functionality provided by these fields. The following components are available:

Property	Description
captcha	Loads captcha images from the headless server

Property	Description
combobox	Displays drop-down lists
datePicker	Displays a date picker
fileUploader	Displays an upload dialog
hint	Displays note text
markdown	Displays markdown as HTML elements
suggestions	Displays autocomplete items from REST services as a drop-down list under input fields

The following properties can be passed to the components named above:

captcha:

Property	Description
buttonText	An optional character string for the refresh button on the captcha component. If this is not specified, the button shows an icon instead.

combobox:

Property	Description
all	All properties from the template's parameter must be passed.

datepicker:

Property	Description
all	All properties from the template's parameter must be passed.

fileUploader:

Property	Description
trigger	The class or ID of the trigger element
inline	An optional Boolean value that specifies whether the element should be displayed inline.

hint:

Property	Description
all	All properties from the template's parameter must be passed.

markdown:

Property	Description
markdown	The <i>markdown</i> property accepts stringified markdown as a value.

suggestions:

Property	Description
All	All properties from the template's parameter must be passed.

The components are executed within the HTML template literal tag in the templates:

```
`${ props.components.captcha( {...} ) }`
```

Modifying and extending templates

To extend the templates, you can modify the HTML elements and classes inside the templates. This gives you the option of modifying the appearance and behaviour of the components.

You can add or remove classes to modify the styling, or add additional HTML elements in order to provide additional functionality.

The templates can also be executed asynchronously: this means that you can access and display data that is not passed directly to the templates by the Formcentric Client. This gives you the option of integrating with APIs or other external data sources.

To use asynchronous data in the templates, you can use JavaScript functions like *fetch* to request data from a server. You can then display the data received in the templates by utilising the corresponding variables or placeholders.

Extension options:

1. Support for one or more user-defined CSS classes. Optional CSS classes can be added, so as to modify the styling of the input element. For this, a custom class can be used in the *className* definition:

```
input className="customClass" />
```

2. Modifying the markup: New markup elements can easily be added and existing elements modified:

```
inputField: (html, props) => html`<div className="fc-field
  ${props?.properties?.hidden ? 'fc-hiddenField' : ''}
  ${props?.properties?.hint ? 'fc-field--has-hint' : ''}
  ${props?.fieldError ? 'fc-field--has-error' : ''}
  ${props?.fieldSuccess ? 'fc-field--is-valid' : ''}`>
```

```

    ${customFunction(html, props)}

    <div className="fc-textinput">
      <div className="fc-textinput__input">
        <input
          id=${props.id}
          name=${props.name}
          value=${props.value}
          oninput=${props.oninput}
          onfocus=${props.onfocus}
          onblur=${props.onblur}
          type=${props.properties.type || 'text'}
          autocomplete=${props.properties?.autocomplete}
          maxLength=${props.properties?.maxLength}
          disabled=${props.properties?.disabled}
          placeholder=${props.properties?.placeholder || ''}
          readOnly=${props.properties?.readOnly}

          {...customProperties}
        />

        ${props.components.suggestions(props)}
      </div>

      ${label(html, props)} ${hint(html, props)} ${error(html, props)}
    </div>
  </div>

```

6.7.4. Special integration scenarios

For most use cases, the *@formcentric/client* script is simply loaded and then executed. However, there are some special scenarios, such as single-page applications (SPA), in which the script must not be executed until the DOM tree has been fully constructed. In such cases, it is useful to be able to import the script dynamically and execute it at the exact moment when the virtual DOM has been fully constructed. This point in time will depend on the SPA framework.

```

function App() {
  const ref = useRef(null);

  const formDef = "TGU5Kmx4svPaahc2aSm-4PHzoKWWtvC ... D-ZwC6MPQRWA==";

  useEffect(() => {
    if (!ref) return;

    import("@formcentric/client/dist/formcentric");
  }, [ref]);

  return (
    <div
      ref={ref}
      data-fc-id="<<id>>"
      data-fc-form-definition={formDef}
    ></div>

```

```
);  
}
```

If no dynamic import is possible, the function *initFormcentric* from the *window.formcentric* object can be called after loading the script.

```
window.formcentric.initFormcentric()
```

6.7.5. Troubleshooting

Always check the browser log. If no client output can be found there, then the *formcentric.js* script was not loaded and/or executed.

There are two reasons for a message stating that the form *div* could not be found:

1. No *div* tag with the data attribute *fc-id* was found
2. The script *formcentric.js* was loaded without specifying the *defer* attribute

Several issues may cause a situation where no form is displayed although a form *div* was found:

1. No *div* tag with the data attribute *fc-id* was found
2. The script *formapp.js* was not loaded

Internal components like DatePicker or FileUploader are rendered without styling:

1. *@formcentric/client/dist/formcentric_component_style.css* was not found in the page

Adjusting the log level

If necessary, the log level for the Formcentric Client can be extended to include warnings and other information by making the following entries in the local storage of the browser dev tool for the embedding page.

Key	Value
FC-logLevel-[WRAPPER]	Info
FC-logLevel-[FORMAPP]	Info