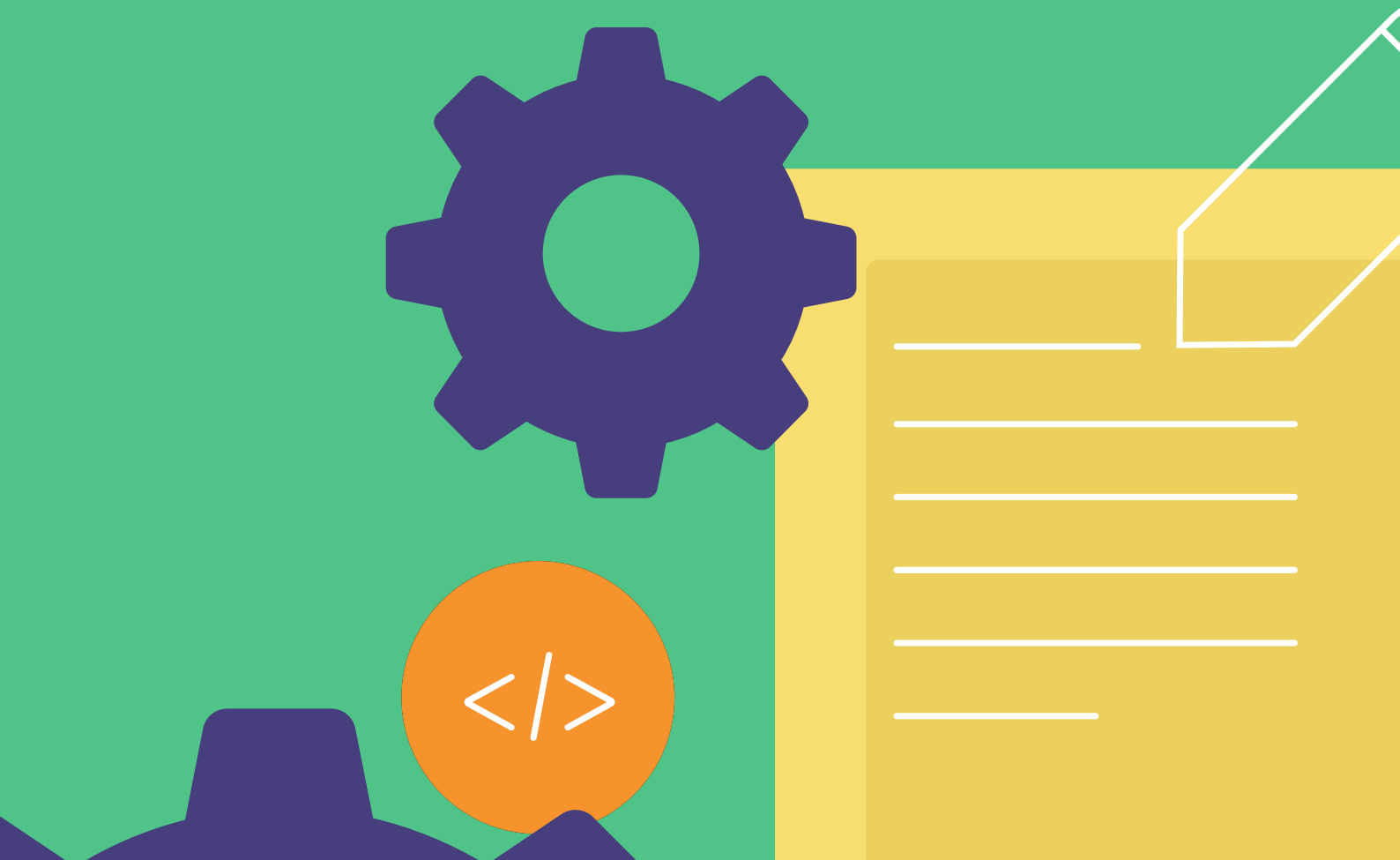
FORMCENTRIC

Entwicklerhandbuch

Version 2310.1.0



Formcentric for CoreMedia: Entwicklerhandbuch

Breite Str. 61, 22767 Hamburg
Deutschland

1. Einleitung

Dieses Handbuch beschreibt, wie die Formularmanagererweiterung Formcentric installiert, konfiguriert und erweitert wird. Es richtet sich an Administratoren und Entwickler. Für das vollständige Verständnis des Textes benötigen Sie Kenntnisse im Bereich der Administration und Bedienung von CoreMedia sowie im Bereich der Java-Softwareentwicklung.

Kapitel 4, *Integration* **:** beschreibt die Schritte, die Sie bei der Installation von Formcentric ausführen müssen.

Kapitel 5, *Konfiguration* **:** beschreibt, wie Sie die verschiedenen Komponenten von Formcentric konfigurieren.

Kapitel 6, *Programmierung und Anpassung* **:** zeigt Ihnen, wie Sie Formcentric um zusätzliche Funktionen erweitern können.

1.1. Begriffsdefinitionen

In diesem Handbuch werden folgende Begriffe verwendet:

Begriff	**Beschreibung**
Redakteur	Person, die Formulare erstellt und bearbeitet.
Benutzer	Person, die ein Formular ausfüllt.
Formular	Im Browser dargestelltes HTML-Webformular.
Formularelemente	Alle Elemente, aus denen ein Formular zusammengesetzt wird (Eingabefelder, Auswahllisten, Checkboxen, et cetera).
Editor	CoreMedia Studio
Formulareditor	Erweiterung des CoreMedia Studios, mit dem Formulare angelegt und bearbeitet werden können.
Formulardaten	Die vom Benutzer in das Formular eingegebenen Daten.

2. Übersicht

Auf der Redaktionsseite erweitert Formcentric das CoreMedia-Studio um einen grafischen Property-Editor, mit dem Redakteure beliebige Webformulare erstellen und bearbeiten können.

Für die Darstellung der Formulare und die Verarbeitung der abgesendeten Formulardaten stellt Formcentric Ihnen eine Integration in die CoreMedia CAE oder alternativ eine eigenständige Server-Anwendung zur Verfügung.

Bei Verwendung der CAE-Integration wird die HTML-Ausgabe serverseitig über Content-Beans und Freemarker-Templates erzeugt. Bei Einsatz des Formcentric Headless-Servers erfolgt das Rendering der Formulare browser-seitig durch einen von Formcentric bereitgestellten React-Client.

Beide Integrationen enthalten verschiedene Spring-Controller für die Verarbeitung der Daten. Ein Formular-Controller validiert die empfangenen Daten und reicht sie an spezifische Actions weiter, die die abschließende Verarbeitung durchführen. Auf diese Weise können verschiedene Backend-Systeme wie Mail-Server, Formcentric Analytics oder Datenbanken angebunden werden.

Die in Formcentric enthaltene Analytics-Komponente ermöglicht es die abgesendeten Formulardaten zu speichern und auszuwerten. Formcentric Analytics besteht aus zwei Web-Applikationen. Die Backend-Applikation ist für das Speichern der Daten in einer relationalen Datenbank zuständig. Hierfür stellt es eine REST-Schnittstelle zur Verfügung, über die Clients mit dem Backend kommunizieren können. Neben den reinen Formulardaten speichert das Backend auch die Formular-Sessions, sofern dies für das jeweilige Formular aktiviert ist.

Die Reporting-Applikation ist eine moderne Single-Page-Anwendung, mit der die im Backend gespeicherten Formulardaten angezeigt, gelöscht und exportiert werden können.

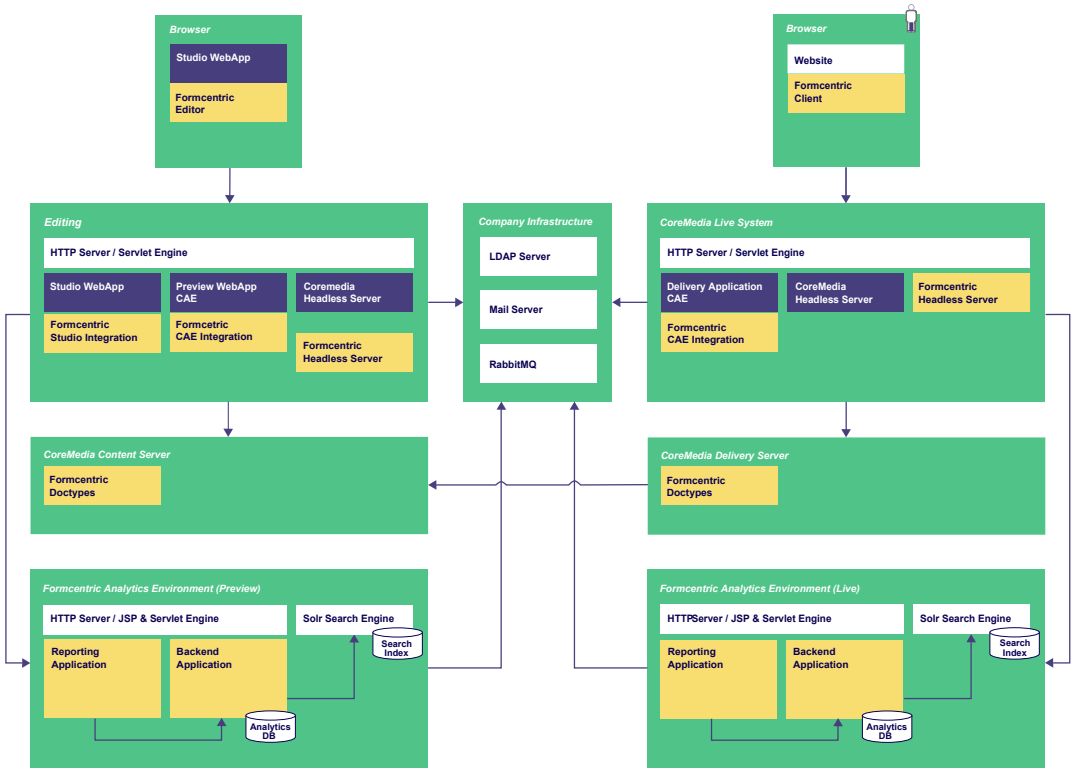


Abbildung 2.1. Architekturübersicht

3. Systemvoraussetzungen

Formcentric 2310.1.0 ist nur für die Verwendung mit der aus der Version hervorgehenden CoreMedia Content Cloud Version vorgesehen.

Formcentric benötigt das JavaScript-Framework „jQuery“ ab Version 1.12.4 und Java 11.

Darüber hinaus gelten dieselben Systemanforderungen, wie sie auch für die eingesetzte CoreMedia Content Cloud-Version gelten.

4. Integration

In diesem Abschnitt finden Sie aus Entwicklerperspektive beschrieben, wie Sie Formcentric in den CoreMedia Blueprint integrieren.

Sollte Ihre Lösung nicht auf dem CoreMedia Blueprint basieren, können Sie die Formcentric Extension genauso integrieren wie in der Anleitung beschrieben. Sie müssen lediglich die verwendeten Dependencies aus den CoreMedia Blueprint entfernen.

4.1. Maven Repository und npm Registry hinzufügen

Ergänzen Sie Ihre Maven *settings.xml* um das Formcentric Artifactory:

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

Um Ihre persönlichen Zugangsdaten zu erhalten, wenden Sie sich bitte an unseren Helpdesk (helpdesk@formcentric.com).

Konfigurieren Sie npm für den Zugriff auf die Packages im @formcentric Scope in dem Formcentric Artifactory:

```
pnpm config set @formcentric:registry \
  https://maven.monday-consulting.com/artifactory/api/npm/formcentric-npm/
```

Melden Sie sich mittels npm an der npm-Registry an:

```
npm login --scope=@formcentric --registry=\
  https://maven.monday-consulting.com/artifactory/api/npm/formcentric-npm/
```

Es ist unter https://maven.monday-consulting.com möglich, Artefakte zu suchen, im Browser herunterzuladen oder über neue Releases informiert zu werden.

4.2. Formcentric Extensions Archiv herunterladen

Das Formcentric Extensions Archiv enthält alle Dateien, die Sie für die Integration von Formcentric als CoreMedia Extension in den Standard CoreMedia Blueprint in Version benötigen. Dieses stellen wir Ihnen als tar- und zip-Datei zur Verfügung. Beide Archive sind inhaltsgleich. Sie können sich frei für das für Sie passende Format entscheiden.

```
mvn dependency:copy -Dartifact=com.formcentric.coremedia:
        formcentric-blueprint-extension:2310.1.0:tar
    -DoutputDirectory=.
```

Die Ordnerstruktur des Archivs ist an den Blueprint Workspace angepasst: die diversen *formcentric* Extension-Ordner für die verschiedenen CoreMedia Komponenten unter jeweils *modules/extensions* müssen entsprechend in die zugehörigen Verzeichnisse (z.B. *apps/cae/modules/extensions*) im Blueprint Workspace verschoben werden.

Durch die angeglichene Ordnerstruktur ist das Entpacken und Verschieben der Erweiterung mithilfe eines einzigen Befehls im Root-Verzeichnis des Blueprints möglich:

```
tar -xvf formcentric-blueprint-extension-*.tar
```

4.3. Formcentric Extensions integrieren

In Abschnitt 4.2, „Formcentric Extensions Archiv herunterladen“ haben Sie die Formcentric Extensions heruntergeladen und an die korrekte Stelle im Blueprint entpackt. Als nächster Schritt passen Sie nun den Maven-Parent in allen Formcentric Extensions (also z.B. *apps/cae/modules/extensions/formcentric/pom.xml*) und in allen Untermodulen an die GroupId und Version Ihres Blueprints an:

```
<parent>
    <groupId>your.blueprint.groupId</groupId>
    <artifactId>component.extensions</artifactId>
    <version>your.blueprint.version</version>
</parent>
```

Sie können hierzu die sich bereits im Blueprint Workspace befindlichen Scripte *set-blueprint-version.sh* und *set-blueprint-groupId.sh* nutzen, da die Formcentric Extensions mit den Blueprint Standardwerten für GroupId und Version ausgeliefert werden.

Die Formcentric Extensions sind nun in Ihrem Blueprint integriert. Abschließend müssen Sie die CoreMedia Extensions synchronisieren und aktivieren. Folgen Sie hierfür der CoreMedia Dokumentation zum *CoreMedia Maven Extension Plugin* und aktivieren Sie die Extension mit dem Namen *formcentric*:

```
mvn -f workspace-configuration/extensions extensions:sync
                    -Denable=formcentric
```

4.4. Formcentric Studio-App einbinden

Bevor die Formcentric Editor App im Studio genutzt werden kann, muss diese noch im Studio hinzugefügt werden. In *apps/studio-client/global/studio/jangaroo.config.js* muss in den *appPath* folgende Zeile hinzugefügt werden:

```
"@formcentric/studio-app.customizations": {
```

```
                buildDirectory: "dist",
                },
```

In *apps/studio-client/global/studio/package.json* müssen die *dependencies* um diese Zeile erweitert werden:

```
"@formcentric/studio-app.customizations": "<WORKSPACE VERSION>"
```

Wobei *WORKSPACE VERSION* durch die Version aus der *package.json* ersetzt werden muss.

Anschließend muss die Formcentric App noch zu den *packages* in *apps/studio-client/pnpm-workspace.yaml* hinzugefügt werden.

```
  - "apps/formcentric/app"
```

4.5. Formcentric Frontend-Archiv herunterladen

Das Formcentric Frontend Archiv enthält alle Dateien, die Sie für die Integration des Formcentric Bricks in den Standard CoreMedia Blueprint (oder Frontend) Workspace in Version benötigen. Dieses stellen wir Ihnen als tar- und zip-Datei zur Verfügung. Beide Archive sind inhaltsgleich. Sie können sich frei für das für Sie passende Format entscheiden.

```
mvn dependency:copy -Dartifact=com.formcentric.coremedia:
        formcentric-blueprint-frontend:2310.1.0:tar
    -DoutputDirectory=.
```

Das Integrieren des Formcentric Bricks für die Erweiterung des verwendeten Themes verläuft analog. Verschieben Sie den Ordner *formcentric* unter *frontend/bricks* in den Ordner *frontend/bricks* im Blueprint Workspace oder alternativ in den Ordner *bricks* des Frontend Workspaces.

Durch die angeglichene Ordnerstruktur ist das Entpacken und Verschieben der Erweiterung mithilfe eines einzigen Befehls im Root-Verzeichnis des Blueprints möglich:

```
tar -xvf formcentric-blueprint-frontend-*.tar
```

bzw. für den Frontend Workspace:

```
tar -xvf formcentric-blueprint-frontend-*.tar --strip=1
```

4.6. Formcentric Brick integrieren

Für die Erweiterung des Themes muss nun noch der Formcentric Brick, der bereits in *frontend/bricks/formcentric* bzw. in *bricks/formcentric* liegt, integriert werden.

Diese Anleitung orientiert sich am *chefcorp-theme*, dem einem der Beispiel-Themes des Blueprints. Die Integration in projektspezifische Themes verläuft nach demselben Prinzip.

1. In *frontend/themes/chefcorp-theme/package.json* werden alle verwendeten Bricks aufgelistet. Fügen Sie hier *@formcentric/formcentric-coremedia-frontend* hinzu:

```
{
  "name": "@coremedia/corporate-theme",
  ...
  "dependencies": {
    ...,
    "@formcentric/formcentric-coremedia-frontend": "workspace:*"
  },
  ...
```

Vermeiden Sie Versionskonflikte und passen Sie die jQuery Version, die in der package.json des Formcentric Bricks verwendet wird, an die im Theme verwendete jQuery Version an, sofern hier eine vorhanden ist.

Um Autovervollständigung der Formcentric FreeMarker Makros nutzen zu können, kann die formcentric.ftl zu den impliziten Imports in der Datei *frontend/src/main/resources/freemarker_implicit.ftl* hinzugefügt werden:

```
...
[#-- asset management download portal --]
[#import "/lib/coremedia.com/blueprint/am.ftl" as am]
[#-- Formcentric --]
[#import "/lib/formcentric.com/formcentric.ftl" as fc]
...
```

4.7. Workspace bauen

Formcentric ist nun vollständig als Blueprint Extensions integriert und das Theme um den Formcentric Brick erweitert. Bauen Sie jetzt den Workspace wie gewohnt mit Maven durch.

5. Konfiguration

Für die Darstellung der Formulare und die Verarbeitung der abgesendeten Formulardaten stehen Ihnen zwei Alternativen zur Verfügung.

5.1. CoreMedia Headless-Server

Die GraphQL-Erweiterung von Formcentric für den CoreMedia Headless-Server ergänzt das Schema für den standard Formular-Dokumenttyp *Form* um zwei Properties, die für die Verwendung mit dem Formcentric-Client und dem Formcentric Headless-Server benötigt werden.

Hierbei handelt es sich um:

formDefinition (**String**)**:** Verschlüsselte Form-Definition zur Verwendung mit dem Formcentric Client.

formReferences (**String**)**:** Verschlüsselte Referenzen, die in der Formulardefinition auf andere CoreMedia-Inhalte verweisen und nur vom Formcentric Headless Server verwendet werden.

formReferencedContent: Alle aus der Formulardefinition verlinkten Content-Dokumente in Form eines Array aus CMTeasable-Objekten

Eine minimale GraphQL-Beispielabfrage kann wie folgt aussehen.

```
{
   content {
      content(id: "<replace with FORM ID>") {
         ... on Form {
            formDefinition
            formReferences
            formReferencedContent {
               ... on CMTeasable {
                  type
               }
            }
         }
      }
   }
}
```

5.2. CAE-Erweiterung

Die Spring Konfigurationsdateien entnehmen Sie bitte dem oben genannten Formcentric Blueprint Workspace. Diesen stellen wir Ihnen als ZIP-Archiv zur Integration in den CoreMedia Workspace zur Verfügung.

Sie finden die Dateien unter *formcentric-blueprint-cae/src/main/resources/META-INF/*

Die Ordnerstruktur stellt sich wie folgt dar:

Datei / Verzeichnis	Beschreibung
coremedia/	Spring-Konfigurationen
resources/WEB-INF/	Lizenz- und Properties-Dateien

5.2.1. Spring-Konfigurationen

Die Konfiguration der Formularerweiterung innerhalb der Webanwendung erfolgt über Spring-Konfigurationen, die im Verzeichnis *formcentric-blueprint-cae/src/main/resources/META-INF/coremedia* der Webanwendung abgelegt sind. Folgende Einstellungen können darin vorgenommen werden:

component-formcentric.xml

Analog zu den component-*.xml-Dateien im CoreMedia-Workspace, werden in dieser Datei alle benötigten Spring Dateien aggregiert und der *PropertySourcesPlaceholderConfigurer* konfiguriert. Wenn Sie eine eigene Spring-XML-Konfiguration erstellen, fügen Sie diese und benötigte Properties-Dateien hier hinzu.

formcentric-actions.xml

Konfiguriert die benötigten Actions. Die hier aufgeführten Action-Beans müssen zusätzlich in das Action-Mapping eingetragen werden (siehe „formcentric-controllers.xml“).

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

  <!-- Mapping of format identifiers to MailBodyRenderer.
    Note: The available format identifiers are defined in the
    forms.properties configuration of the Form Editor. -->

  <property name="bodyRendererMapping">
    <map>
      <entry key="html" value-ref="htmlBodyRenderer"/>
      <entry key="text" value-ref="textBodyRenderer"/>
    </map>
  </property>
</bean>
...
```

formcentric-captcha.xml

Für die Erzeugung der Captchas wird das Open Source Framework JCaptcha verwendet. Bei dieser Konfiguration handelt es sich um eine JCaptcha Standardkonfiguration, mit der sich das Aussehen und Verhalten der Captchas beeinflussen lässt. Eine ausführliche Beschreibung der Konfigurationsmöglichkeiten findet sich auf der Projekt-Homepage.

https://jcaptcha.atlassian.net/wiki/display/general/Home

formcentric-contentbeans.xml

Konfiguriert das Form-ContentBean. Sollte ihr Dokumentenmodell davon abweichen, so müssen Sie diese Konfiguration löschen oder anpassen.

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
            http://www.springframework.org/schema/beans/spring-beans.xsd">

  <bean name="contentBeanFactory:Form"
    class="com.formcentric.coremedia.blueprint.contentbeans.FormImpl"
       scope="prototype">

      <property name="digester" ref="formDigester" />
  </bean>
</beans>
```

Damit das Formularframework mit Ihrem konkreten Formular-ContentBean umgehen kann, muss dieses das Interface *com.formcentric.contentbeans.WebForm* implementieren.

Eine Implementierung findet sich im Package *com.formcentric.coremedia.blueprint.contentbeans*.

formcentric-services.xml

Konfiguriert die REST-Services. Die hier aufgeführten *RestService*-Beans müssen zusätzlich in das Service-Mapping des REST-Cotrollers eingetragen werden (siehe „formcentric-controllers.xml“).

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

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

formcentric-controllers.xml

Konfiguriert den Formular-Controller, den REST-Controller sowie die *FormCommandBeanFactory*, mit der das *FormCommandBean* erzeugt wird. Das *FormCommandBean* ruft die konfigurierten Initializer, Validatoren und Actions auf und erzeugt das Form-Model.

Neue Actions, Validatoren und Formularelemente konfigurieren Sie auf der Command-Bean-Factory:

```
<bean id="fcFormCommandBeanFactory"
    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="fcNotemptyValidator"/>
      <entry key="jcaptcha" value-ref="fcCaptchaValidator"/>
      <entry key="email" value-ref="fcEmailValidator"/>
      <entry key="date" value-ref="fcDateValidator"/>
      <entry key="number" value-ref="fcNumberValidator"/>
      <entry key="javascript" value-ref="fcJavascriptValidator"/>
      <entry key="regex" value-ref="fcRegexValidator"/>
      <entry key="length" value-ref="fcLengthValidator"/>
      <entry key="zipcode" value-ref="fcZipCodeValidator"/>
      <entry key="phone" value-ref="fcPhoneValidator"/>
      <entry key="password" value-ref="fcPasswordValidator"/>
      <entry key="bic" value-ref="fcBicValidator"/>
      <entry key="iban" value-ref="fcIbanValidator"/>
      <entry key="file" value-ref="fcFileValidator"/>
      <entry key="equal" value-ref="fcEqualValidator"/>
    </map>
  </property>

  <property name="actionMapping">
    <map>
      <entry key="mailAction" value-ref="fcMailAction"/>
      <entry key="datastoreAction" value-ref="fcDatastoreAction"/>
      <entry key="sequenceAction" value-ref="fcSequenceAction"/>
    </map>
  </property>
</bean>

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

  <!-- Part specific to form controller -->
  <property name="formCommandBeanFactory" ref="fcFormCommandBeanFactory"/>
  <property name="bindOnNewForm" value="true" />
  <property name="contentRepository" ref="contentRepository"/>
  <property name="contentBeanFactory" ref="contentBeanFactory"/>
  <property name="dataViewFactory" ref="dataViewFactory" />
  <property name="commandNamePrefix" value="command"/>
</bean>

<bean id="fcRestController"
```

```
    class="com.formcentric.controllers.RestController">

  <property name="contentRepository" ref="contentRepository" />
  <property name="contentBeanFactory" ref="contentBeanFactory" />
  <property name="dataViewFactory" ref="dataViewFactory" />
  <property name="commandNamePrefix" value="command" />

  <!-- Service mapping -->
  <property name="restServiceMapping">
    <map>
      <entry key="Länder" value-ref="fcDeCountriesRestService"/>
      <entry key="Countries" value-ref="fcEnCountriesRestService"/>
    </map>
  </property>
</bean>
```

formcentric-digester.xml

Konfiguriert das *formDigester*-Bean, mit dessen Hilfe das XML-Unparsing der Formularbeschreibung durchgeführt wird.

```
<bean name="formDigester"
    class="com.formcentric.model.FormDigester" scope=”prototype”>

  <property name="dataViewFactory" ref="dataViewFactory" />
  <property name="contentRepository" ref="contentRepository" />
  <property name="contentBeanFactory" ref="contentBeanFactory" />
</bean>
```

Das FormDigester-Bean muss unbedingt mit dem Scope *prototype* definiert werden, da es anderenfalls zu Wechselwirkungen zwischen unterschiedlichen Formulardokumenten kommen kann.

formcentric-resourcebundle.xml

Fügt dem bestehenden *messageSource*-Bean das Formcentric Resource-Bundle hinzu.

```
<customize:prepend id="fcMessageSourceCustomizer"
    bean="messageSource" property="basenames">
  <list>
    <bean class="java.lang.String">
      <constructor-arg
          value="com.formcentric.resourcebundle.messages"/>
    </bean>
  </list>
</customize:prepend>
```

Im CoreMedia Blueprint werden die Resource-Bundles im Content-Repository verwaltet (siehe *formcentric_blueprint_install_de.pdf*), so dass in dieser Konfiguration kein Resource-Bundle angegeben werden muss.

formcentric-views.xml

Fügt dem bestehenden *programmedViews*-Bean weitere Views hinzu.

```
  <bean id="fcJsonView" class="com.formcentric.view.JsonView"/>
  <bean id="fcFileInfoView" class="com.formcentric.view.FileInfoView"/>
  <bean id="fcEmptyView" class="com.formcentric.view.EmptyView"/>
  <bean id="fcCaptchaView" class="com.formcentric.view.ImageCaptchaView"/>
  <bean id="fcThumbnailView" class="com.formcentric.view.ThumbnailView">
    <property name="thumbnailWidth" value="90"/>
    <property name="thumbnailHeight" value="90"/>
    <property name="crop" value="true"/>
  </bean>

  <customize:append id="fcViewsCustomizer" bean="programmedViews"
      order="200">
    <map>
      <entry key="java.util.Map#json" value-ref="fcJsonView"/>
      <entry key="java.util.List#fileInfo" value-ref="fcFileInfoView"/>
      <entry key="com.formcentric.model.FileHolder#thumbnail"
          value-ref="fcThumbnailView"/>
      <entry key="java.awt.image.BufferedImage#captcha"
          value-ref="fcCaptchaView"/>
      <entry key="com.formcentric.contentbeans.WebForm#EMPTY"
          value-ref="fcEmptyView"/>
    </map>
  </customize:append>
  ...
```

formcentric-mail.xml

In dieser Konfiguration legen Sie die Verbindungseinstellungen zum Mail-Server fest. Das konfigurierte *mailer*-Bean wird von der Mail-Action verwendet. Wenn Sie die grundlegenden Properties der *mailer*-Bean anpassen möchten, finden Sie alle benötigten Werte in der *formcentric-mail.properties*-Datei.

```
<bean id="fcMailProperties"
    class="org.springframework.beans.factory.config.PropertiesFactoryBean">
    <property name="locations">
        <list>
            <value>classpath*:/META-INF/resources/WEB-INF/
formcentric-mail.properties</value>
        </list>
    </property>
    <property name="propertiesArray">
        <list>
            <bean class="java.lang.System" factory-method="getenv" />
        </list>
    </property>
    <property name="localOverride" value="true"/>
</bean>

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

```
</bean>
```

formcentric-security.xml

In dieser Datei können Sie den *formcentricSecurityServletFilter* zur Abwehr von Cross Site Scripting (XSS) und Cross Site Request Forgery Angriffen (XSRF) konfigurieren. Weitere Informationen erhalten Sie im Abschnitt Web-Security (siehe Abschnitt 5.2.5, „Web-Security“).

```
<bean class="com.formcentric.web.SecurityServletFilterInitializer">
  <property name="urlPatterns">
    <array>
      <value>/servlet/mwf-form/*</value>
      <value>/servlet/mwf-rest</value>
      <value>/servlet/mwf-upload</value>
    </array>
  </property>
  <property name="xsrfPrevention" value="true"/>
  <property name="xsrfSessionBased" value="true"/>
  <property name="xssPrevention" value="true"/>
  <property name="xsrfTokenName" value="com.formcentric.XSRFToken"/>
  <property name="xsrfMethods" value="POST"/>
</bean>
```

formcentric-analytics.xml

In dieser Konfiguration legen Sie die Verbindungseinstellungen zum Formcentric Backend fest. Das konfigurierte *BackendApiClient*-Bean wird von der Analytics-Action, dem *BackendFormStateStore* und der *TrackingCommandBean* verwendet.

Für die Authentifizierung gegenüber dem Formcentric Backend wird ein *Access-Token* benötigt. Wenn dieses Token automatisch zur Laufzeit der CAE generiert werden soll, können Sie den *ClientSecretCredentialsAuthProvider* nutzen. Dieser benötigt das Client-Secret, das in der Konfiguration des Backends vergeben wurde. Alternativ können Sie das Token manuell erzeugen und den *ApplicationAuthProvider* verwenden. Nähere Informationen zur Generierung eines Access-Tokens finden Sie im Installationshandbuch für das Formcentric Backend.

Sie können die zugehörigen Properties in der *formcentric-analytics.properties*-Datei individuell anpassen. Unter anderem lässt sich dort die Übermittlung der potentiell personenbezogenen Meta-Daten an das Formcentric Backend konfigurieren.

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

<!--
  METHOD 2: Pass the full client credentials and request a token at runtime.
-->
```

```
<bean id="fcAnalyticsAuthenticator" class="com.formcentric.backend
    .api.auth.ClientSecretCredentialsAuthProvider">
  <constructor-arg index="0" value="${analytics.backendUrl}" />
  <constructor-arg index="1" value="${analytics.formcentricClientSecret}" />
</bean>

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

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

5.2.2. Verwendung ohne Formcentric Analytics

Die Spring-Konfigurationen *formcentric-actions.xml*, *formcentric-controllers.xml* und *formcentric-analytics.xml* beinhalten Komponenten, die nur bei der Verwendung von Formcentric Analytics genutzt werden können. Wenn Sie Formcentric ohne Analytics verwenden möchten, müssen folgende Beans und Bean-Referenzen entfernt werden:

- **formcentric-actions.xml:** *datastoreAction*
- **formcentric-controllers.xml:** *formStateStore* (*BackendFormStateStore*), *defaultTrackingCommandBean*, *formcentricTrackingController*
- **formcentric-analytics.xml:** Sämtliche Beans in dieser Datei werden nicht benötigt.

5.2.3. Formcentric Lizenzdatei

Die Datei *formcentric-license.xml* konfiguriert den LicenseLoader von Formcentric. Über die zugehörige *formcentric-license.properties* kann der Pfad zur Lizenzdatei angegeben werden.

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

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

Pfade, die nicht mit einem / beginnen, werden relativ zur Webapp aufgelöst.

5.2.4. Dataviews

Um eine schnelle Anzeige der Formulare zu ermöglichen, sollte die Dataview-Konfiguration wie folgt erweitert werden:

```
<!-- Form bean -->
<dataview appliesTo="com.formcentric.coremedia.blueprint.
        contentbeans.FormImpl”>
```

```
  <property name="delegate" associationType="static"/>
</dataview>

<!-- Form elements -->
<dataview appliesTo="com.formcentric.model.xml.ActionNode">
  <property name="children" associationType="composition"/>
  <property name="properties" associationType="dynamic"/>
</dataview>

<dataview appliesTo="com.formcentric.model.xml.FormNode">
  <property name="action" associationType="composition"/>
  <property name="children" associationType="composition"/>
  <property name="childrenFlat" associationType="composition"/>
  <property name="elementNamesFlat" associationType="composition"/>
  <property name="elementsFlat" associationType="composition"/>
  <property name="parent" associationType="static"/>
  <property name="validator" associationType="composition"/>
</dataview>

<dataview appliesTo="com.formcentric.model.xml.InputNode">
  <property name="children" associationType="composition"/>
  <property name="elements" associationType="dynamic"/>
  <property name="parent" associationType="static"/>
  <property name="properties" associationType="dynamic"/>
  <property name="validators" associationType="composition"/>
</dataview>

<dataview appliesTo="com.formcentric.model.xml.OptionNode">
  <property name="children" associationType="composition"/>
</dataview>

<dataview appliesTo="com.formcentric.model.xml.SelectionNode">
  <property name="children" associationType="composition"/>
  <property name="validators" associationType="composition"/>
</dataview>

<dataview appliesTo="com.formcentric.model.xml.ValidatorNode">
  <property name="children" associationType="composition"/>
</dataview>
```

Ersetzen Sie den Klassennamen des ContentBeans für den Formular-Dokumenttyp durch den Klassennamen Ihrer Implementierung. Die Angaben für die verschiedenen Formularelementtypen bleiben in der Regel unverändert.

5.2.5. Web-Security

Zur Abwehr von Cross Site Scripting (XSS) Angriffen und Cross Site Request Forgery Angriffen (XSRF) beinhaltet Formcentric einen Security-Servlet-Filter. Dieser entfernt unzulässige HTML-Tags aus den übertragenen Formulardaten. Zusätzlich prüft der Filter, ob die Formulardaten einen gültigen XSRF-Token enthalten.

Für die Abwehr von XSRF-Angriffen kann jedem Formular ein zusätzlicher XSRF-Token als Hidden-Parameter hinzugefügt und zusammen mit den übrigen Formulardaten an die

Web-Anwendung übertragen werden. Der Security-Filter prüft, ob der übertragene Token mit dem in der User-Session hinterlegten Token übereinstimmt. In diesem Fall wird der Request an die Webanwendung weitergeleitet. Andernfalls wird eine 401-Fehlermeldung an den aufrufenden Client zurückgegeben und der Aufruf im Log der Web-Anwendung im Log-Level *warn* mit den folgenden Informationen protokolliert:

- aufgerufene URL
- übertragene Formulardaten (POST-Parameter)
- IP-Adresse des aufrufenden Clients
- vollqualifizierter Name des aufrufenden Clients oder des letzten Proxies

Das nachfolgende Beispiel zeigt Ihnen, wie Sie den XSRF-Token in das Ausgabe-Template des Formulardokuments einfügen können:

```
...
<#assign targetUrl=cm.getLink(self, "ajax")!""/>
<form method="post" class="mwf-form ${self.properties['style_class']!""}"
        ...
        data-mwf-form="${self.shortId}"
        data-mwf-settings='{
            "url":"${targetUrl}",
            ...
        }'>

    <#-- Include XSRF token&#8211; -->
    <@fc.xsrfToken />

    ...

</form>
```

Zusätzlich zum Formular-Template müssen Sie den XSRF-Token auch in den Ausgabe-Templates der Elemente *InputField, ComboBox, RadioGroup, CheckBoxGroup* und *FileUpload* berücksichtigen.

Das folgende Beispiel zeigt Ihnen, wie Sie den XRSF-Token im Template *InputNode.fileUpload.ftl* in die Formulardaten einfügen können.

```
...
<#-- Construct upload URL, add XSRF token parameters and
store in variable -->
<#assign uploadUrl=cm.getLink(self, "upload", {"form": form,
    "tokenName": fc.xsrfTokenName(),
    "tokenValue": fc.xsrfTokenValue()})!"" />

<div class="mwf-upload"
     data-mwf-fileupload='{
        "url":"${uploadUrl}",
        "id":"${self.id}",
```

```
        "name":"${self.name!""}",
        "autoUpload": ${self.properties['auto_upload']},
        "labels": ${rowLabels},
        "previewMaxWidth": "120",
        "previewMaxHeight": "120"
     }'>

...
```

Den Security-Filter konfigurieren Sie mithilfe der *SecurityServletFilterInitializer*-Bean (siehe „formcentric-security.xml“). Folgende Konfigurationsparameter stehen Ihnen dabei zur Verfügung:

Parameter	**Beschreibung**
xsrfPrevention	Mit diesem Parameter legen Sie fest, dass die User-Session durch einen zusätzlichen XSRF-Token abgesichert werden soll. (true | false)
xsrfMethods	Geben Sie hier an, bei welchen HTTP-Methoden (GET,POST) ein XSRF-Token notwendig sein soll.
xsrfSessionBased	Mit diesem Parameter können Sie festlegen, ob der Token für die gesamte Session gültig ist (true) oder bei jedem Seiten-Reload erneuert wird (false).
xsrfTokenName	Geben Sie hier den Namen des Request-Parameters an, in dem der XSRF-Token übertragen werden soll. Wenn Sie hier nichts angeben, wird der Wert *com.formcentric.XSRFToken* verwendet. Der Name des Token wird automatisch um die aktuelle Formular-ID erweitert. Beispiel: _mwfToken:1234=5E29A7D49404A97E91CB49D799203AB9
xssPrevention	Mit diesem Parameter legen Sie fest, dass unzulässige HTML-Tags aus den übertragenen Formulardaten entfernt werden. Standardmäßig werden alle HTML-Tags entfernt.

Hinweis: Früher wurde der Security-Filter in der *web-fragment.xml* konfiguriert, welche mittlerweile nicht mehr von CoreMedia unterstützt wird. Die Konfigurationswerte können Sie jedoch problemlos 1:1 in die *formcentric-security.xml* übernehmen.

5.2.6. Speichern des Formularstatus

Standardmäßig werden alle von einem Benutzer eingegebenen Daten in der User-Session auf dem Server gespeichert. Bei umfangreichen Formularen kann es jedoch dazu kommen, dass die Session abläuft, bevor der Anwender das Formular zu Ende ausgefüllt und abgesendet hat. In diesem Fall gehen die in der Session gespeicherten Daten verloren.

Formcentric bietet daher zusätzlich die Option, die eingegebenen Daten längerfristig zu speichern. Dies ermöglicht es Benutzern, die Formulareingabe zu unterbrechen und zu einem späteren Zeitpunkt fortzusetzen.

Um diese Funktion zu aktivieren, konfigurieren Sie in der Spring-Konfiguration *formcentric-controllers.xml* einen *FormStateStore*.

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

<bean id="fcFormController"
  class="com.formcentric.controllers.FormController">
  ...
  <property name="autoSaveFormState" value="true" />
</bean>
```

Formcentric stellt zwei verschiedene Store-Implementierungen zur Verfügung:

FileFormStateStore

Diese Implementierung speichert die Formulardaten in einer verschlüsselten Datei auf dem Server. Der zugehörige Dateiname wird in einem Cookie gespeichert. Das Verzeichnis sowie das Verschlüsselungspasswort, die Lebensdauer, die Domain und der Pfad des Cookies können in der Spring-Konfiguration angegeben werden.

```
<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/formcentric" />
</bean>
```

BackendFormStateStore

Diese Implementierung speichert die Formulardaten in der Datenbank des Formcentric-Backends.

```
<bean id="fcFormStateStore"
  class="com.formcentric.store.BackendFormStateStore">
  <property name="client" ref="fcBackendApiClient" />
  <property name="cookiePath" value="/" />
  <property name="cookieDomain" value="my-domain.com" />
  <property name="cookieMaxAge" value="604800" />
</bean>
```

5.2.7. Verschlüsselung von Passwörtern

In der Grundkonfiguration werden Zugangsdaten zu Datenbanken Mail-Servern etc. im Klartext in verschiedenen Konfigurationsdateien gespeichert. Bei einem eventuellen Einbruch in den Server können so gültige Zugangsdaten entwendet werden. Aus diesem Grund haben Sie die Möglichkeit Passwörter auch verschlüsselt abzulegen. Die Passwörter werden in diesem Fall erst beim Start der Anwendung mit dem hinterlegten Verschlüsselungspasswort entschlüsselt. Das zur Verschlüsselung verwendete Passwort müssen Sie vor dem Start der Formcentric Web-Anwendungen in einer Umgebungsvariable speichern. Standardmäßig verwendet Formcentric hierfür die Umgebungsvariable *fc_ENCRYPTION_PASSWORD*.

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

Zur Verschlüsselung der Passwörter steht Ihnen ein Kommandozeilenprogramm zur Verfügung. Die damit verschlüsselten Passwörter müssen Sie anschließend manuell in die entsprechende Konfiguration eintragen.

Laden Sie das Programm aus dem Monday Maven-Repository herunter, indem Sie folgende Befehlszeile in der Konsole ausführen. Die hierfür erforderlichen Zugangsdaten erhalten Sie über unseren Helpdesk (helpdesk@formcentric.com).

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

Um ein Passwort zu verschlüsseln, geben Sie in der Kommandozeile folgenden Befehl ein:

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

Bitte beachten Sie, dass die Parameter in einfachen Anführungszeichen angegeben werden müssen. Folgende Kommandozeilenparameter können Sie beim Start angeben:

Parameter	**Beschreibung**
`-p encryption-password`	Passwort, das zur Ver- oder Entschlüsselung verwendet wird.
`-d`	Passwort entschlüsseln
`-e`	Passwort verschlüsseln
`-?`	Hilfe anzeigen

5.3. Formcentric Headless-Server

Der Formcentric Headless Server integriert sich als zusätzlicher App-Workspace in den CMCC Workspace. Nach dem Entpacken der Formcentric Extension (siehe Abschnitt 4.2, „Formcentric Extensions Archiv herunterladen“) findet sich der Formcentric Headless-Server als neues Maven Modul im Verzeichnis *apps/formcentric-server*. Die Struktur und der Aufbau orientiert sich in weiten Teilen an den App-Workspaces des CMCC, daher sollten Sie bereits mit den meisten der verwendeten Konzepte vertraut sein.

Um den Formcentric Headless-Server-Workspace vollständig in den CMCC Build zu integrieren, nehmen Sie das Maven Modul in die Liste der App Workspace Module in der Root-Pom des CMCC-Workspaces auf:

```
<module>apps/formcentric-server</module>
```

Die vollständige Integration des Formcentric Headless-Server in den CMCC Workspace ist ein optionaler Schritt. Der Workspace ist auch einzeln verwendbar und ermöglicht z. B. ein Deployment oder Betriebsszenario abseits des CoreMedia Stacks zu realisieren.

Bei dem Formcentric Headless Server handelt es sich um eine Spring Boot Anwendung. Damit stehen alle gängigen Arten der Konfiguration zur Verfügung, sprich Properties Dateien, Umgebungsvariablen, Laufzeitparameter etc. Die zur Verfügung stehenden Parameter werden im folgenden Absatz beschrieben.

Der Ort, an dem die Formcentric Lizenz hinterlegt ist, wird für den Formcentric Headless-Server in der Datei *license.properties* konfiguriert.

Eine beispielhafte Integration in das Docker Compose Setup kann wie folgt aussehen (davon ausgehend, dass die Formcentric Lizenz in den Pfad `/formcentric/license/` im Container gemounted oder kopiert ist):

```
<![CDATA[formcentric-server:
  image: ${REPOSITORY_PREFIX:-coremedia}/formcentric-server:latest
  container_name: formcentric-server
  restart: always
  logging: *default-logging
  networks:
    - backend
  environment:
    << : *default-boot-envs
    SPRING_APPLICATION_NAME: 'formcentric-server'
    ANALYTICS_BACKENDURL:
    ANALYTICS_CLIENTTOKEN:
    MAIL_SMTP_HOST:
    MAIL_SMTP_PORT:
    MAIL_USER:
    MAIL_PASSWORD:
    UPLOAD_LOCATION: /tmp
    FORMCENTRIC_LICENSE: /formcentric/license/formcentric-license.txt
    (... and further properties)
  healthcheck: *default-boot-healthcheck
]]>
```

6. Programmierung und Anpassung

6.1. Erweiterung des Formcentric Formular-Editors

Bei der Formcentric Studio-Integration handelt es sich um eine Single-Page-Anwendung, die auf dem JavaScript Framework *React* basiert. Die Benutzeroberfläche des Formulareditors wird dabei browser-seitig aus den vom Server übertragenen JSON-Daten erzeugt. Der Aufbau der Oberfläche wird deklarativ über verschiedene JavaScript-Konfigurationsdateien festgelegt. Dieser Ansatz ermöglicht es Ihnen, auf einfache Weise Änderungen und Erweiterungen an der Redaktionsoberfläche vorzunehmen.

Zentraler Ansatzpunkt für die Anpassung der Redaktionsoberfläche sind die JavaScript-Konfigurationsdateien, die im Entwicklungs-Workspace im Modul *apps/studio-client* im Verzeichnis *apps/formcentric/app/config* abgelegt sind. Alle nachfolgend beschriebenen Anpassungen erfolgen in den darin enthaltenen Dateien.

Die verfügbaren Formularelemente mit ihren Eigenschaften werden in Form von JSON-Objekten beschrieben. Die React-Anwendung erzeugt daraus die Redaktionsoberfläche. Das nachfolgende Beispiel zeigt einen Auszug der Konfiguration für das Formularelement *textArea*.

```
{
    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'
            },

            ...
        ]
    }
}
```

6.1.1. Neues Formularelement hinzufügen

Erweitern Sie den Formulareditor um ein neues Formularelement, in dem Sie die Konfiguration *fields_custom.js* erweitern. Möchten Sie den Editor beispielsweise um das Formularelement *termsCheckbox* mit den Eigenschaften *name, text* und *link* erweitern, so fügen Sie dem JavaScript-Array in der Konfiguration *fields_custom.js* folgende Objektdefinition hinzu.

```
[
    {
        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: {}
            }
        }
    }
]
```

Hinweis: Das äußere JavaScript-Array ist bereits vorhanden und muss lediglich um das Konfigurationsobjekt erweitert werden.

Die nachfolgende Tabelle beschreibt die möglichen Attribute, die eine Felddefinition auf erster Ebene besitzen kann.

Attribut	Beschreibung
icon	Typ: *String* Name des zu ladenden Icons. Der angegeben Name muss mit dem Dateinamen des Icons ohne Dateiendung übereinstimmen.
type	Typ: *String* Name des Formularelements
properties	Typ: *Object* Definiert die Eigenschaften eines Feldes, die auf der rechten Seite unter *Feldeigenschaften* im Formular-Editor bearbeitet werden können. Die Objekteigenschaften von *properties* stellen dabei einzelne Editor-Tabs dar. Der folgende JSON-Ausschnitt konfiguriert zwei Tabs mit den Namen *general* und *special*, mit insgesamt drei Eigenschaften: *name, label, hint*. `properties: { general: [ { title: 'name', type: 'text' }, { title: 'label', type: 'text' } ], special: [ { title: 'hint', type: 'text' } ] }` Damit das Feld bei der weiteren Verarbeitung eindeutig identifiziert werden kann, wird die Eigenschaft *titel* mit dem Wert *name* im Array *general* benötigt. Eine Liste aller möglichen Eigenschaftstypen finden sie unter Abschnitt 6.1.5, „Eingabeelemente für Elementeigenschaften“
specialProperties	Typ: *Object* Im Attribut *specialProperties* konfigurieren Sie Eigenschaften, die vom Editor für interne Funktionen ausgewertet werden. Der folgende JSON-Ausschnitt definiert die Verwendungsweise des Feldes innerhalb einer Condition. `specialProperties: { condition: {`

Attribut	Beschreibung

	<pre> conditionable: true,
	        operators: {
	            startswith: {
	                values: [],
	                freeField: true,
	                useChildren: false
	            },
	            endswith: {
	                values: [],
	                freeField: true,
	                useChildren: false
	            },
	            contains: {
	                values: [],
	                freeField: true,
	                useChildren: false
	            }
	        }
	    }
	}</pre>
	Mit der Angabe von `conditionable: true` legen Sie fest, dass das Feld in einer Bedingung angewählt werden kann.
	
	Die in der Bedingung für diesen Formularelementtyp auswählbaren Operatoren legen Sie im Object *operators* fest. Eine Operator-Definition folgt dabei immer dem Schema
	
	`<operator-name>: {values: [], freeField: true, useChildren: false}`.
	
	Der Name des Operators wird zudem auch als Übersetzungs-ID für die Internationalisierung der Benutzeroberfläche verwendet (siehe Abschnitt 6.1.7, „Internationalisierung der Benutzeroberfläche“).
	
	Im Attribut *values* haben Sie die Möglichkeit, ein String-Array mit Werten anzugeben, die vom Redakteur bei der Definition einer Condition augewählt werden können.
	
	Durch Angabe des Attributs `freeField: true` ermöglichen Sie es Redakteuren, benutzerdefinierte Werte einzugeben. Diese Option wird zum Beispiel für Vergleichsoperatoren benötigt, bei denen Redakteure eigene Vergleichswerte angeben müssen.
	
	Wenn es sich bei dem neuen Feldtyp um einen Listentyp mit vorgegebenen Optionen handelt, haben Sie im Attribut *useChildren* die Möglichkeit festzulegen, dass die Optionen als Wert der Bedingung ausgewählt werden können.

6.1.2. Neuen Validator hinzufügen

Um einem Eingabefeld einen neuen Validator hinzuzufügen, erweitern Sie die *format*-Eigenschaft des zugehörigen Eingabeelements.

Das nachfolgende Beispiel zeigt die Konfiguration des E-Mail-Validators für das einzeilige Textfeld (*inputField*).

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

Der angegebene Attributname (im Beispiel *email*) muss dem externen Namen des Validators entsprechen. Der Name wird auch für die Internationalisierung der Oberfläche verwendet. In der Übersetzungsdatei wird mit der Übersetzungs-ID `<validator-name>Validator` nach einer Beschriftung für den Validator gesucht.

Sie können unter *fields* die gewünschten Felder des Validators definieren. Die verfügbaren Feldtypen sind in der Tabelle unter Abschnitt 6.1.5, „Eingabeelemente für Elementeigenschaften“ aufgeführt.

6.1.3. Neue Aktion hinzufügen

Erweitern Sie den Formulareditor um eine neue Aktion, in dem Sie die Konfiguration *actions_custom.js* erweitern.

Möchten Sie den Editor beispielsweise um die Aktion *simpleMailAction* mit den Eigenschaften *to, subject, body* und *note* erweitern, so fügen Sie dem JSON-Array in der Konfiguration *actions_custom.js* folgende Objektdefinition hinzu.

```
[
    {
        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: {}
            }
        }
    }
]
```

Die nachfolgende Tabelle beschreibt die möglichen Attribute, die eine Aktionsdefinition auf erster Ebene besitzen kann.

Attribut	**Beschreibung**
icon	Typ: *String* Name des zu ladenden Icons. Der angegeben Name muss mit dem Dateinamen des Icons ohne Dateiendung übereinstimmen.
type	Typ: *String* Name des Feldtyps
properties	Typ: Object Beschreibt die Eigenschaften einer Aktion die auf der rechten Seite unter *Eigenschaften* im Formular-Editor bearbeitet werden können. Die Objekt-Eigenschaften von *properties* stellen dabei einzelne Tabs da. Der folgende JSON-Ausschnitt erzeugt zwei Tabs mit den Namen *general* und *special*, mit insgesamt drei Eigenschaften *to*, *subject* und *hint*. `properties: { general: [ {`

Attribut	Beschreibung

	<pre> title: 'to',
	            type: 'text',
	            properties: {
	                required: true
	            }
	        },
	
	        {
	            title: 'subject',
	            type: 'text',
	            properties: {
	                required: true
	            }
	        }
	    ],
	    special: [
	        {
	            title: 'hint',
	            type: 'text'
	        }
	    ]
	}</pre>
	Eine Liste aller möglichen Eigenschaftstypen finden sie unter Abschnitt 6.1.5, „Eingabeelemente für Elementeigenschaften“

specialProperties	Typ: *Object* Im Attribut *specialProperties* konfigurieren Sie Eigenschaften, die vom Editor für interne Funktionen ausgewertet werden. `specialProperties: { maxCount: 1 }` Mit `maxCount: <anzahl>` legen Sie fest, wie oft die Aktion innerhalb eines Formulars verwendet werden kann.

6.1.4. Neue Elementeigenschaften hinzufügen

Die Definition von Elementeigenschaften erfolgt unterhalb des Attributs *properties* der übergeordneten Formularelementdefinition (siehe Abschnitt 6.1.1, „Neues Formularelement hinzufügen“). Durch Angabe eines JSON-Objekts mit folgender Struktur fügen Sie dem Formularelement (Formularfeld, Aktion oder Validator) eine neue Eigenschaft hinzu.

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

Die nachfolgende Tabelle beschreibt die Attribute des Konfigurationsobjekts.

Attribut	Beschreibung
title	Name, unter dem die Feldeigenschaft in der Formulardefinition gespeichert wird.
type	Typ der Eigenschaft. Die möglichen Typen werden in der folgenden Liste erklärt.
value	Optionale Angabe eines Vorbelegungswertes
properties	Weitere typspezifische Konfigurationsmöglichkeiten
properties.required	Legt fest, ob es sich bei der Eigenschaft um ein Pflichtfeld handelt.

6.1.5. Eingabeelemente für Elementeigenschaften

In der folgenden Tabelle finden Sie die Konfigurationsobjekte für die Eingabeelemente der verfügbaren Elementeigenschaften beschrieben. Diese können Sie bei der Definition der verschiedenen Formularelementeigenschaften verwenden. Bitte beachten Sie, dass einige Typen nicht für alle Formularelemente verwendet werden können.

Typ	Beschreibung
text	Freitextfeld Verwendung: alle Formularelemente `{ title: 'label', type: 'text' }`
number	Zahlenfeld das ausschließlich numerische Eingaben zulässt. Verwendung: alle Formularelemente `{ title: 'maxlength', type: 'number', properties: { min: 0, max: null } }` Unterstützt auch die wissenschaftliche Schreibweise (z. B. *10e6*). Durch die Angaben `properties.min` und `properties.max` können Sie Grenzen definieren. Mit `properties.min: null` heben Sie eine vorhandene Begrenzung auf.
date	Element für die Auswahl eines Datums. Verwendung: alle Formularelemente `{ title: 'from',`

Typ	Beschreibung

	`type: 'date'`
	`}`
	
	Mit `value` können sie eine Vorbelegung definieren. Folgt dem Standard JavaScript-Datumsformat.

checkbox	Checkbox
	
	Verwendung: alle Formularelemente
	
	`{`
	`title: 'requiredField',`
	`type: 'checkbox'`
	`}`

dropdown	Liste mit festen Einträgen, aus der der Redakteur einen Eintrag auswählen kann.
	
	Verwendung: alle Formularelemente
	
	`{`
	`title: 'pattern',`
	`type: 'dropdown',`
	`properties: {`
	`options: ['dd.MM.yyyy', 'yyyy-MM-dd']`
	`}`
	`}`
	
	Unter `properties.options` können Sie die Auswahlmöglichkeiten als String-Array angeben.
	
	`properties: {`
	`options: [`
	`{text: 'Value 1', value: 'one'},`
	`{text: 'Example Two', value: two}`
	`]`
	`}`
	
	Optionen können auch als Objekte mit `value` (Wert) und `text` (Anzeigename) definiert werden.

dropdown_format	Auswahlliste für Feldvalidatoren
	
	Verwendung: Eingabeelemente (*inputField*, *passwordField*, etc.)
	
	Bei der Auswahl werden die Eigenschaften des ausgewählten Validators unterhalb der Auswahlliste eingeblendet.
	
	Das folgende Beispiel zeigt die Definition des E-Mail-Validators.
	
	`{`
	`title: 'format',`
	`type: 'dropdown_format',`
	`properties: {`

Typ	Beschreibung

	<pre> options: {
	            email: {
	                enabled: true,
	                fields: {
	                    errormessage: {
	                        title: 'errormessage',
	                        type: 'text'
	                    }
	                }
	            }
	        }
	    }
	}</pre>
	Unter `properties.options` können Sie die auswählbaren Validatoren festlegen.
	
	Unter `properties.options["<validator-name>"].fields` können Sie die Validator-Eigenschaften pflegen.
	
	Durch die `properties.options["<validator-name>"].enabled=true` oder `properties.options["<validator-name>"].enabled=false` aktivieren bzw. deaktivieren Sie den Validator, so dass er nicht mehr auswählbar ist.

syntax	Mehrzeiliges JavaScript-Eingabefeld mit Syntax-Highlighting
	
	Verwendung: alle Formularelemente
	
	<pre>{
	    title: 'script',
	    type: 'syntax',
	    value: 'function calculate() {};'
	}</pre>

condition	Eingabeelement für die Bearbeitung von Bedingungen.
	
	Verwendung: *condition*
	
	<pre>{
	    title: 'conditionContent',
	    type: 'condition',
	    properties: {
	        conditional_fields: [],
	        condition_conjunction: 'true',
	        condition: []
	    }
	}</pre>

wysiwyg	Mehrzeiliges Texteingabefeld, das die Angabe von Formatierungen ermöglicht. Verwendung: alle Formularelemente

Typ	Beschreibung

	<pre>{
	    title: 'value',
	    type: 'wysiwyg'
	}</pre>

element	Ermöglicht die Auswahl von anderen Elementen desselben Formulars.
	Verwendung: alle Formularelemente
	<pre>{
	    title: 'elements',
	    type: 'element'
	}</pre>

dataSource	Eingabeelement für die variable Parameterliste einer Datenquelle
	Verwendung: inputField, comboBox, radioGroup, checkboxGroup, hiddenField
	<pre>{
	    title: 'datasource',
	    type: 'dataSource',
	    properties: {
	        datasource_params: []
	    }
	}</pre>

reference	Element für die Auswahl einer FirstSpirit-Referenz (öffnet den Auswahldialog von FirstSpirit).
	Verwendung: alle Formularelemente
	<pre>{
	    title: 'pictureUrl',
	    type: 'reference',
	    properties: {
	        FS_refType: 'picture'
	    }
	}</pre>
	Im Property *properties.FS_refType* haben Sie die Möglichkeit, die Auswahl auf einen bestimmten Inhaltstyp (Page, Picture, Folder etc.) einzuschränken. Folgende Angaben sind an dieser Stelle möglich: *pageref, picture, file*

multi_dropdown	Liste mit festen Einträgen, aus der der Redakteur mehrere Einträge auswählen kann.
	Verwendung: alle Formularelemente
	<pre>{
	    title: 'numberType',</pre>

Typ	Beschreibung

	<pre> type: 'multi_dropdown',
	    properties: {
	        configuration_name: 'phoneNumberTypes'
	    }
	}</pre>
	Im Attribut *properties.options* können Sie die Auswahlmöglichkeiten (als *Strings*) definieren
	
	Alternativ können die Werte auch aus der Absatzvorlage bezogen werden. Geben Sie in diesem Fall im Attribut *configuration_name* den Namen des entsprechenden GOM-Elements an.
	
	Der Parameter *configuration_name* legt fest, unter welchem Namen der Wert in der Formulardefinition gespeichert wird.

regEx_dropdown	Auswahlliste für reguläre Ausdrücke
	
	Verwendung: regex
	
	<pre>{
	    title: 'mailPattern',
	    type: 'regEx_dropdown',
	    properties: {
	        options: [
	            {
	                text: '^[+]{0,1}[0-9\\s-/]*$',
	                label: 'phone'
	            },
	            {
	                text: '^[a-zA-ZÁ-ÿöäüÖÄÜß\\s-]*$',
	                label: 'characters'
	            }
	        ]
	    }
	}</pre>

field_mapping	Element für die Auswahl einer PDF-Vorlage (öffnet einen Auswahl-dialog von FirstSpirit).
	
	Verwendung: *pdfAction*
	
	Anschließend kann den Formularfeldern ein PDF-Feld aus der Vorlage zugeordnet werden. Wenn es sich bei den Feldern um Auswahllisten handelt, können deren Optionen ausgewählt und einander zugeordnet werden.
	
	Verwendung: pdfAction
	
	<pre>{
	    title: 'field_mapping',
	    type: 'field_mapping',
	    value: '[]'</pre>

Typ	Beschreibung
	`}`

custom_mapping	Mit diesem Eingabelement können eigene Zuordnungs Tabellen gestalltet werden.
	
	`{`
	`  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'`
	`  },`
	`  {`
	`  name: 'connectedMapField_type',`
	`  key: 'type',`
	`  value: 'field'`
	`  },`
	`  {`
	`  name: 'connectedMapField',`
	`  key: 'value',`
	`  value: 'field'`
	`  },`
	`  ],`
	`  placeholder: 'custom_table.attribut',`
	`  },`
	`  {`
	`  type: 'dropdown',`

Typ	Beschreibung

	<pre> name: 'attributoption',
	            placeholder: 'custom_table.attributoption',
	            loadRemoteData: 'FS_ServiceField',
	            connectedField: 'attribut',
	         },
	      ],
	   },
	},</pre>
	Im Attribut *properties.mapping* können Sie die Spalten definieren. Mit dem key *type*, entscheiden Sie ob es sich um ein Auswahlliste (dropdown) oder um ein Eingabefeld (text).
	
	Der key *name*, setzt den key für den Export, der jeweiligen Felder in einer Zeile.
	
	Mit dem key*placeholder*, können Sie den Platzhalter, für das Feld definieren.
	
	Mit dem key*loadRemoteData*, können Sie wie bei einem dropdown, entscheiden ob die Optionen von einem FirstSpirit Serice kommen sollen.
	
	Falls die Optionen aus einem FirstSpirit Service kommen, können Sie mit dem key *loadRemoteDataOptions* zusätzliche Attribute mitgeben, wie zum Beispiel Werte von anderen Feldern auf dem Element oder aus dem Mapping selbst. Das folgende Beispiel, erstellt dieses Object, um es beim loadRemoteData Aufruf mitzugeben. `{task: <ValueOfFieldsomeDropdown>, type: <TypeVonFormElementAusgewähltInField>, value: '<ValueVon-MappingDropdownField>'}`
	
	Falls Sie in einem Dropdown vorhandene Formularfelder auswählen möchten, können Sie diese mit dem key *selectableField-Types* mitgeben.
	
	Möchten Sie auf verschachtelte Optionen, von einem Formularfeld oder von den Optionen aus loadRemoteData zugreifen, können Sie mit *connectedField* und Angabe des Namens, eines Mapping Feldes auf diese zugrifen und die zur auswahl bereit stellen.

6.1.6. Bestehende Formularelemente anpassen

Um ein bestehendes Formularelement anzupassen, kopieren Sie dessen vollständige Elementdefinition aus der zugehörigen Standardkonfiguration (*fields_default.js* bzw. *actions_default.js*) in die entsprechende Konfigurationsdatei (*fields_custom.js* bzw. *actions_custom.js*).

Ändern oder ergänzen Sie anschließend wie oben beschrieben die Elementeigenschaften entsprechend Ihrer Anforderungen.

Hinweis: Änderungen an den Standardkonfigurationen im Entwicklungs-Workspace haben keine Auswirkungen auf den Formulareditor.

6.1.7. Internationalisierung der Benutzeroberfläche

Zur Internationalisierung der Benutzeroberfläche werden die sprachabhängigen Beschriftungen aus zentralen Sprachdateien gelesen. Formcentric unterstützt standardmäßig die Sprachen Deutsch und Englisch.

Um die Beschriftungen bestehender oder neuer Formularelemente anzupassen bzw. hinzuzufügen, erweitern bzw. ändern Sie die Sprachdateien *formeditor_de_custom.json* und *formeditor_en_custom.json*, die Sie im Entwicklungs-Workspace finden.

Jede Beschriftung ist mit einer eindeutigen Übersetzungs-ID in den Sprachdateien gespeichert. Die Übersetzungs-IDs der Elementeigenschaften setzen sich dabei typischerweise aus dem internen Elementnamen und dem Namen der jeweiligen Eigenschaft zusammen. Für die Eigenschaft *placeholder* des Passwortfelds sieht der Eintrag wie folgt aus:

```
"passwordField.placeholder": "Platzhalter"
```

Die Beschriftung einer neuen Elementeigenschaft können Sie hinzufügen, in dem Sie in jeder Sprachdatei einen entsprechenden Eintrag hinzufügen.

6.2. Erweiterung der CAE-Integration

6.2.1. Freemarker-Templates

Die Ausgabe der Formulare und Formularelemente erfolgt wie bei allen anderen Dokumenttypen durch Freemarker-Templates innerhalb der CAE. Analog zu den CoreMedia Dokumenttypen, ist dabei jedem Formularelementtyp ein eigenes Template zugeordnet.

Das nachfolgende Beispiel zeigt das Template *InputNode.inputField.ftl* des einzeiligen Texteingabefelds.

```
<@spring.bind fc.bind(self) />
<#assign hasErrors=spring.status.error />
    <#assign restUrl=cm.getLink(self, "rest", {"form": form,
        "tokenName": fc.xsrfTokenName(),
        "tokenValue": fc.xsrfTokenValue()})!"" />

    <#assign params=self.properties['datasource_params']!"{}"/>

    <input id="${self.id}"
           type="text"
           name="${spring.status.expression!""}"
           value="${spring.status.value!""}"
           class="mwf-text ${self.properties['style_class']!""}"
           ${self.properties['readonly']?boolean?then("readonly", "")}
           maxlength="${self.properties['maxlength']!""}"
           data-mwf-id="${self.id}"
           placeholder="${self.properties['placeholder']!""}"
           data-mwf-datasource='{
```

```
                "type" : "suggestion",
                "url" : "${restUrl}",
                "data" : {},
                "params" : ${params}
            }'/>
<@spring.showErrors separator="<p>" classOrStyle="mwf-error"/>
```

Alle Formularelemente werden auf der Datenebene (Model) durch ein Objekt vom Typ *com.formcentric.model.xml.InputNode* repräsentiert. Auf die Eigenschaften *name, label, value* und *children* können Sie über entsprechende Getter-Methoden auf dem *InputNode*-Bean zugreifen. Der Zugriff auf alle weiteren Eigenschaften erfolgt über die Properties-Map des InputNode-Beans.

Die nachfolgende Tabelle zeigt Ihnen alle Formularelementtypen und deren Eigenschaften. Die in eckigen Klammern dargestellten Properties müssen aus der Properties-Map gelesen werden.

Element	**Eigenschaften**
form	name, [style_class, next_label, submit_label, cancel_label, script, save_state]
inputField	name, label, value, [hint, placeholder, style_class, readonly, maxlength, datasource, datasource_params]
textArea	name, label, value, [hint, placeholder, style_class, readonly, maxlength, rows, cols]
passwordField	name, label, [hint, placeholder, style_class]
button	name, label, [hint, style_class, onclick]
checkBoxGroup	name, label, children, [hint, style_class, datasource, dynamic, datasource_params]
comboBox	name, label, value, children, [hint, style_class, datasource, dynamic, datasource_params]
pageBreak	name, label, [style_class, condition, style_class, next_label, back_label, script]
paragraph	name, value, [bold, italic, style_class]
captcha	name, label, [hint]
radioGroup	name, label, children, [hint, style_class, datasource, dynamic, datasource_params]
summary	label, [style_class, elements]
hiddenField	name, value
fileUpload	name, [multiple, hint, style_class, auto_upload]
condition	[condition, condition_conjunction, conditional_fields]
pageCondition	[condition, condition_conjunction, next_page, script]
layout	label, [layout]

Element	Eigenschaften
calculatedValue	name, label, [script, visible, clientside, style_class]
mailAction	[subject, to, cc, bcc, from, body, format, note, replyto, send_hidden_fields, condition, condition_conjunction]
datastoreAction	[note, condition, 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	-

Neben den oben beschriebenen InputNode-Beans übergibt das System im Request weitere Objekte an die Formular-Templates. Die nachfolgende Tabelle zeigt Ihnen eine Übersicht aller übergebenen Objekte.

Parametername	Typ	Beschreibung
self	com.formcentric.contentbeans.WebForm com.formcentric.model.xml.InputNode	Bean des aktuellen Formulardokuments oder Formularelements
pageElements	java.util.List	Liste mit den Elementen der aktuellen Formularseite
pageCount	java.lang.Integer	Anzahl der Formularseiten
form	com.formcentric.contentbeans.WebForm	Formulardefinition
currentPage	java.lang.Integer	Seitennummer der aktuellen Formularseite
currentPageNode	com.formcentric.model.xml.InputNode	Bean der aktuellen Formularseite
formdata	java.util.Map	Map mit den vom Anwender eingegebenen Formulardaten

Freemarker Funktionen und Makros

Formcentric stellt Ihnen eine Freemarker Library zur Verfügung, die spezialisierte Funktionen für die Darstellung der Formulare beinhaltet.

Um die Funktionen verwenden zu können, fügen Sie folgende Anweisung in die Freemarker-Templates ein:

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

Nachfolgend finden Sie die in der Library enthaltenen Funktionen beschrieben.

fc.forEachPageElement

Listen-Funktion, die die Elemente der aktuellen Seite beinhaltet.

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

Parameter	Beschreibung
layoutFacets	Falls dieser Wert "true" ist werden die Elemente in layoutFacets aufgeteilt. (optional)
removeEmptyFacets	Legt fest, ob leere Layouts bei Erstellung der Liste ignoriert werden sollen. (optional) **Standardwert:** *false*
exclude	Kommaseparierte Liste der Elementtypen, die bei der Erstellung der Liste ignoriert werden. Erfolgt hier keine Angabe, so werden alle Elementtypen – mit Ausnahme der exkludierten Typen – berücksichtigt. (optional)
include	Kommaseparierte Liste der Elementtypen, die bei der Erstellung der Liste berücksichtigt werden. (optional)

```
<#list fc.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>
            <@cm.include self=input view=input.type />
        </#list>
    </ul>
</#list>
```

fc.forEachPage

Listen-Funktion, die eine Liste der gesammelten Seiten zurückliefert.

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

Parameter	Beschreibung
compact	Legt fest, dass Formularseiten mit gleichem Titel zusammengefasst werden. (optional)

Parameter	Beschreibung
	Standardwert: *false*

fc.summary

Funktion, die eine Liste aller Elemente als *com.formcentric.model.InputBean* des Formulars zurückliefert. Dabei können auch die vom Benutzer eingegebenen Daten abgefragt werden.

Folgende Properties können darauf abgefragt werden:

name	Name des Formularelements
label	Beschriftung des Formularelements.
type	Typ des Formularelements.
object	Value-Bean des Formularelements.
value	String-Repräsentation des Value-Beans.
valueLabels	String-Array mit den Labels der Optionen, die in der Auswahl (*comboBox, radioGroup, checkboxGroup*) selektiert wurden. Wenn es sich bei dem zugehörigen Eingabeelement nicht um eine Auswahl handelt, so wird der Wert des Elements in dem Array zurückgegeben.
page	Nummer der Seite, auf der sich das Element befindet.
pageLabel	Beschriftung der Seite, auf der sich das Element befindet.
layout	Name des Layouts, in dem sich das Element befindet.
input	*InputNode* des Elements.

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

Parameter	Beschreibung
self	*InputNode* eines Formularelements. Wenn dieser Wert gesetzt ist, so wird die Iteration bei dem angegebenen Element abgebrochen.
elements	Kommaseparierte Liste mit den Namen der Formularelemente, die in der Zusammenfassung angezeigt werden sollen. Wenn dieses Attribut gefüllt ist, wird das Attribut *self* ignoriert. (optional)
include	Kommaseparierte Liste der Elementtypen, die bei der Iteration berücksichtigt werden. Erfolgt hier keine Angabe, so werden alle Elementtypen – mit Ausnahme der exkludierten Typen – berücksichtigt. (optional)
exclude	Kommaseparierte Liste der Elementtypen, die bei der Iteration ignoriert werden. (optional)

Parameter	Beschreibung
	Standardwert: *button, hiddenField, condition, pageCondition, page-break, captcha, passwordField*
hideEmptyFields	Legt fest, dass alle leeren Felder ignoriert werden. **Standardwert:** *false*
excludeEmptyFields	Legt fest, dass leere Felder ignoriert werden. (optional) **Standardwert:** *false*

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

fc.captcha

Template, mit dem Sie ein Captcha-Bild erzeugen können.

Attribut	Beschreibung
url	Url des Captcha-Servlets.
id	Die Id des Captcha-InputNodes.
linkClass	CSS-Klasse(n) die auf den Link um das Captcha-Bild angewandt wird. (optional) **Standardwert:** *""*
imgClass	CSS-Klasse(n) die auf das Captcha-Bild angewandt wird. (optional) **Standardwert:** *""*
title	Das title-Attribut für das Captcha-Bild. (optional) **Standardwert:** *""*
alt	Das alt-Attribut für das Captcha-Bild. (optional) **Standardwert:** *Captcha*

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

fc.ifCaptcha

Boolean Funktion die auswertet, ob das Captcha *name* **nicht** korrekt eingegeben wurde.

Parameter	Beschreibung
name	Name des Captcha-Elements.

```
<#if fc.ifCaptcha(self.name!"")>
    <#assign captchaUrl=cm.getLink(self, "captcha")!"" />
    <@fc.captcha url=captchaUrl id=self.id />
</#if>
```

fc.getStandardButton

Funktion, die den über das Attribut ″buttonType″ bestimmten StandardButton des Formulars liefert.

Parameter	Beschreibung
buttonType	Typ des StandardButtons. Er kann die folgenden Werte annehmen: _next Button der auf die nächste Seite des Formulars leitet. _back Button der auf die vorherige Seite des Formulars leitet. _cancel Button der die Bearbeitung des Formulars abbricht. _finish Button der das Formular abschickt. _exit Button mit dem das Formular verlassen werden kann.

```
<#assign finishButton=fc.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>
```

fc.valueOut

Funktion, mit der der aktuelle Wert eines Formularfeldes ausgegeben werden kann.

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

Parameter	Beschreibung
name	Name des Formularfeldes.
preferLabel	Legt fest, dass anstelle des Werts das Label des Werts ausgegeben werden soll. (optional)

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

fc.conditions

Funktion, mit der die JavaScript-Definitionen der Bedingungselemente erzeugt werden können. Platzieren Sie die Funktion am Ende des Formular-Templates.

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

fc.calculatedValues

Funktion, die die JSON-Definitionen der *Berechneten Werte* erzeugt.

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

fc.markdown

Makro, mit dem ein Wert mit Markdown interpretiert ausgegeben werden kann. Dieses Makro kann sowohl mit einem übergebenen Wert (s. Parameter ″value″) oder mit einem body umgehen.

Parameter	**Beschreibung**
value	Wert der mit Markdown interpretiert werden soll. (optional)
inline	Legt fest, ob das Ausgabe-HTML auf Inline Elemente beschränkt werden soll. (optional) **Standardwert:** *false*

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

fc.vars

Makro, mit dem Variablen aus dem Formularkontext in der Ausgabe ersetzt werden.

Parameter	**Beschreibung**
map	Map, mit den Variablenwerten (Key, Value). **Hinweis:** Im Page-Scope wird standardmäßig eine Map mit den Formuarvariablen (*formVariables*) und eine Map mit den Formularwerten (*formdata*) übergeben.

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

fc.bind

Funktion, die den Pfad zurückgibt, an den der Node gebunden ist.

Parameter	**Beschreibung**
node	InputNode, dessen Pfad gesucht wird.

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

fc.encodeUrl

Funktion die die übergebene Url in UTF-8 encodiert.

Parameter	Beschreibung
url	Url, die codiert werden soll.

fc.hasValidator

Funktion, die überprüft, ob der im Parameter *name* festgelegte Validator in dem InputNode *node* vorhanden ist.

Parameter	Beschreibung
node	InputNode, der untersucht werden soll.
name	Name des Validators.

fc.validatorByName

Funktion die den im Parameter *name* festgelegten Validator des InputNode *node* zurückgibt.

Parameter	Beschreibung
node	InputNode, der untersucht werden soll.
name	Name des Validators.

fc.elementByName

Funktion die den im Parameter *name* festgelegten InputNode des Formular *form* zurückgibt.

Parameter	Beschreibung
form	Formular, das den InputNode enthält.
name	Name des Elements.

Security Library

Für die Erzeugung und Ausgabe von XSRF-Token (siehe Abschnitt 5.2.5, „Web-Security“) beinhaltet Formcentric eine Security Library. Die Security Library wird bei der Integration der Freemarker Funktionen mit eingebunden.

Nachfolgend beschriebene Freemarker Funktionen sind darin enthalten.

fc.xsrfToken

Makro, das ein verstecktes Formularfeld mit einem XSRF-Token erzeugt.

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

fc.xsrfTokenName

Funktion, die aus der FormularId einen xsrfTokenName erzeugt.

Parameter	Beschreibung
formId	ID des Formulars, für das der Token erzeugt werden soll. Falls dieser Parameter leer ist wird die im Request mitgeschickte FormularId verwendet. (optional)

```
<#assign restUrl=cm.getLink(self, "rest", {"form": form,
  "tokenName": fc.xsrfTokenName(), "tokenValue": fc.xsrfTokenValue()})!""/>
```

fc.xsrfTokenName

Funktion, die aus der FormularId einen xsrfTokenValue erzeugt.

Parameter	Beschreibung
formId	ID des Formulars, für das der Token erzeugt werden soll. Falls dieser Parameter leer ist wird die im Request mitgeschickte FormularId verwendet. (optional)

```
<#assign restUrl=cm.getLink(self, "rest", {"form": form,
  "tokenName": fc.xsrfTokenName(), "tokenValue": fc.xsrfTokenValue()})!""/>
```

6.2.2. Implementierung einer Action

Wie bereits beschrieben, wird die Business-Logik der Formulardatenverarbeitung in der Webanwendung in Actions gekapselt. Konkret handelt es sich dabei um Klassen, die das Interface *com.formcentric.actions.Action* implementieren. Zusätzliche Business-Beans können einer Action über Spring injiziert werden. Auf diese Weise können beispielsweise Data-Access-Objekte (DAO) für den Zugriff auf externe Datenbanken zur Verfügung gestellt werden. Die Actions werden dem Formular-Controller beim Start der Anwendung über Spring injiziert.

Variable Action-Parameter, die bei der Anlage des Formulars vom Redakteur eingegeben werden müssen (beispielsweise die Zieladresse der Mail-Action), werden in der Properties-Map des ActionNode-Beans an die Action-Implementierung übergeben.

Das nachfolgende Beispiel zeigt Ihnen, wie Sie die im Abschnitt 6.1.3, „Neue Aktion hinzufügen“ beschriebene *CustomAction* implementieren und konfigurieren können.

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

  public static final String PROP_CUSTOM = "anyCustomActionPropertyName";

  @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
      ...

    return HandlerHelper.createModelWithView(formDefinition, "success");
  }

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

Die Action erhält beim Aufruf der *execute*-Methode alle zur Verfügung stehenden Daten. Neben den eigentlichen Formulardaten werden im *ExecutionContext* die Formulardefinition, die Action-Definition, die Formularvariablen (siehe Abschnitt 6.2.3, „Variable zur Vorbelegung von Formularfeldern hinzufügen“) sowie das Request-Objekt übergeben. Die *parameters*-Map enthält nur die Werte der sichtbaren Formularelemente. Durch Aufruf der Methode *getRawFormData*() auf dem *ExecutionContext*-Bean kann auf alle Formulardaten zugegriffen werden.

Die *execute*-Methode muss ein Objekt vom Typ *ModelAndView* zurückgeben. Dieses wird für die Darstellung der Ergebnisseite verwendet, die dem Benutzer nach dem Absenden der Daten angezeigt wird.

In der Regel wird das *ModelAndView*-Objekt mit dem Formular-Bean und einem speziellen View (beispielsweise *success*) erzeugt.

Darüber hinaus besteht aber auch die Möglichkeit, den Request auf eine andere Seite weiterzuleiten. In diesem Fall kann das *ModelAndView*-Objekt wie folgt erzeugt werden:

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

Das Objekt *renderBean* und der View-Name können dabei von der speziellen Business-Logik der Action-Implementierung erzeugt werden.

Bei manchen Anwendungsfällen werden Fehler in den Eingabedaten erst bei der Verarbeitung durch das angebundene Backend entdeckt. In diesem Fall soll dem Anwender nicht die Ergebnisseite, sondern erneut das Formular mit einer Fehlermeldung angezeigt werden. Um dies zu erreichen, muss die Action – analog zu den Validatoren – einen Fehler auf dem im *ExecutionContext* übergebenen *Errors*-Bean erstellen.

```
public ModelAndView execute(ExecutionContext<WebForm> context, Map<String,
    Object> formData) {
```

```
    ...
    context.getErrors().rejectValue("username", DUPLICATE_USER_ERROR,
        "That user name is already being used.");

    return null;
}
```

Tragen Sie die Action in die Spring-Konfiguration *formcentric-actions.xml* ein:

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

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

</bean>
```

Tragen Sie sie zusätzlich in der Spring-Konfiguration *formcentric-controllers.xml* in das Action-Mapping ein:

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

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

</bean>
```

6.2.3. Variable zur Vorbelegung von Formularfeldern hinzufügen

Für die Vorbelegung von Formularfeldern stehen dem Redakteur eine Reihe vordefinierter Variablen zur Verfügung. Standardmäßig kann er die Variablen *date, time, serverDate, serverTime, clientDate, clientTime, timezone, url, language, ip, remoteUser, principal, userAgent* und *referer* verwenden.

Für die Bereitstellung eigener Variablen müssen Sie die Methode *getVariables*() auf dem FormCommandBean überschreiben.

```
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;
  }
}
```

Zur Instanziierung des neuen *CustomFormCommandBeans* ist es notwendig, dass Sie auch die *FormCommandBean*-Factory überschreiben und diese in die Konfiguration *form-centric-controllers.xml* eintragen.

```
public class CustomCommandBeanFactory extends DefaultFormCommandBeanFactory
{
  @Override
  public CustomFormCommandBean createBeanFor(WebForm formDefinition) {

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

     return commandBean;
  }
}
```

Tauschen Sie die *DefaultFormCommandBeanFactory* in der Spring-Konfiguration *form-centric-controllers.xml* gegen die *CustomCommandBeanFactory* aus:

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

Die Formularfelder werden einmalig beim ersten Aufruf des Formulars mit den definierten Vorbelegungswerten initialisiert. Dabei werden auch die Variablen durch ihre Werte ersetzt. Anschließende Änderungen der Variablenwerte werden daher nicht in ein bereits initialisiertes Formular übernommen.

6.2.4. Implementierung eines REST-Services

Formcentric beinhaltet eine REST-Schnittstelle, die Sie dazu verwenden können, Auswahllisten oder Eingabefelder zur Laufzeit mit Daten aus externen Systemen zu befüllen. Dabei kann es sich um statische, dynamische oder benutzerindividuelle Daten handeln. Alle spezifischen Funktionen der Schnittstelle sind in Klassen vom Typ *com.formcentric.rest.RestService* gekapselt. Durch die Implementierung eines eigenen REST-Services können Sie die Schnittstelle um zusätzliche Funktionen erweitern. Das nachfolgende Beispiel zeigt Ihnen einen REST-Service, der eine Map mit statischen Key/Value-Paaren erzeugt.

```
public class ExampleRestService extends BaseRestService {
```

```
  @Override
  public Object invoke(ServiceContext<WebForm> context, Map<String,
    Object> formData, Map<String, Object> data) {

    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;
  }
}
```

Bei Aufruf der *invoke*-Methode werden dem REST-Service neben dem ServiceContext auch die bereits abgesendeten Benutzereingaben (Parameter *formData*) und die noch nicht abgesendeten Benutzereingaben (Parameter *data*) übergeben. Damit sind Sie in der Lage, direkt auf die Eingaben des Benutzers zu reagieren, unabhängig davon ob, diese bereits abgesendet wurden oder nicht.

Über den *ServiceContext* haben Sie zudem Zugriff auf die Formulardefinition, das Eingabeelement, die Konfigurationsparameter des REST-Services und das Request-Object.

Tragen Sie den REST-Service in die Spring-Konfiguration *formcentric-services.xml* ein:

```
<bean name="exampleRestService"
   class="com.custom.forms.web.ExampleRestService">

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

</bean>
```

Tragen Sie ihn zusätzlich in der Spring-Konfiguration *formcentric-controllers.xml* in das Service-Mapping des REST-Controllers ein:

```
<bean id="fcRestController"
    class="com.formcentric.controllers.RestController">
        <property name="commandNamePrefix" value="command" />

        <property name="restServiceMapping">
            <map>
                <entry key="Example" value-ref="exampleRestService"/>
                ...
            </map>
        </property>
```

```
</bean>
```

Der Zugriff auf den Service erfolgt über die URL:

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

Als Antwort des Aufrufs wird folgender JSON-String zurückgegeben:

```
[
  {
    "k":"key1",
    "v":"value1",
    "i":"mwf6aab0bb24033",
    "h":"8d0c3e13950d86c1a7383f066105f78c"
  },
  {
    "k":"key2",
    "v":"value2",
    "i":"mwf06a7a0930d37",
    "h":"d22d445101243a5f616cfd64c765e399"
  },
  {
    "k":"key3",
    "v":"value3",
    "i":"mwf1674ffb0a121",
    "h":"c0ad1fa77bb1b79ca757ee1ffce9f416"
  }
]
```

Um Manipulationen an den übertragenen JSON-Daten zu verhindern, werden die einzelnen Key/Value-Paare durch einen zusätzlichen HASH-Wert abgesichert, der beim Absenden des Formulars serverseitig validiert wird.

Aufgrund dieser Absicherung können keine externen REST-Services aufgerufen werden, da deren Daten nicht die erforderlichen HASH-Werte enthalten. Für den Zugriff auf externe Services können Sie jedoch einen eigenen Proxy-REST-Service implementieren, der seinerseits auf den externen REST-Service zugreift.

Ab Version 2.3 von Formcentric erfolgt der Aufruf der REST-Services innerhalb der Freemarker-Templates über das HTML-Attribut *data-mwf-datasource*. Im Attributwert müssen Sie ein JSON-Objekt angeben, das die URL des REST-Services, die Verwendungsart (*checkbox*, *radio*, *selection* oder *suggestion*) sowie ggf. weitere Parameter enthält.

Standardmäßig können Sie bei folgenden Eingabeelementen einen REST-Service angeben:

inputField:

```
<#assign restUrl=cm.getLink(self, "rest", {"form": form,
        "tokenName": fc.xsrfTokenName(),
        "tokenValue": fc.xsrfTokenValue()})!"" />

<#assign params=self.properties['datasource_params']!"{}"/>
```

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

hiddenField:

```
<#assign restUrl=cm.getLink(self, "rest", {"form": form,
        "tokenName": fc.xsrfTokenName(),
        "tokenValue": fc.xsrfTokenValue()})!"" />

<#assign params=self.properties['datasource_params']!"{}"/>

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

comboBox:

```
<#assign restUrl=cm.getLink(self, "rest", {"form": form,
        "tokenName": fc.xsrfTokenName(),
        "tokenValue": fc.xsrfTokenValue()})!"" />
<#assign userValue=fc.valueOut(self.name!"")!""/>
<#assign params=self.properties['datasource_params']!"{}"/>

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

checkBoxGroup:

```
<#assign restUrl=cm.getLink(self, "rest", {"form": form,
```

```
        "tokenName": fc.xsrfTokenName(),
        "tokenValue": fc.xsrfTokenValue()})!"" />
<#assign userValue=fc.valueOut(self.name!"")!""/>
<#assign params=self.properties['datasource_params']!"{}"/>

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

radioGroup:

```
<#assign restUrl=cm.getLink(self, "rest", {"form": form,
        "tokenName": fc.xsrfTokenName(),
        "tokenValue": fc.xsrfTokenValue()})!"" />
<#assign userValue=fc.valueOut(self.name)!""/>
<#assign params=self.properties['datasource_params']!"{}"/>

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

Da innerhalb des JSON-Strings das doppelte Anführungszeichen verwendet wird, müssen Sie für das HTML-Attribut das einfache Anführungszeichen verwenden.

Wie zuvor beschrieben, stehen Ihnen innerhalb des REST-Services die abgesendeten und die noch nicht abgesendeten Formulareingaben zur Verfügung. Dies können Sie beispielsweise dafür verwenden, einen REST-Service zu implementieren, der zu einer vom Benutzer eingegebenen Postleitzahl passende Standorte in einer Auswahlliste zurückgibt.

In diesem Beispiel ist es sinnvoll, die Auswahlliste automatisch zu aktualisieren, wenn der Anwender die Postleitzahl ändert, da zu der geänderten Postleitzahl evtl. andere Standorte gehören. Hierfür steht ab Version 1.4 des Formcentric jQuery-Plugins der Parameter *dependsOn* zur Verfügung. Dieser kann in der Redaktionsoberfläche in die Parameterliste eines REST-Services eingetragen werden (siehe dazu auch Kap. 3.5 im Studio Benutzerhandbuch). Als Wert müssen dabei die Namen der Eingabeelemente angegeben werden, von

denen das Ergebnis des ausgewählten REST-Services abhängt. Jede Änderung in einem der angebenen Eingabeelemente führt dazu, dass der REST-Service erneut aufgerufen wird.

6.2.5. JavaScript

Formcentric beinhaltet und benötigt standardmäßig die nachfolgend beschriebenen JavaScript-Abhängigkeiten.

blueimp-file-upload (npm-package)

Für den Upload von Dateien verwendet Formcentric das Blueimp jQuery-File-Upload Plugin. Je nach verwendetem Browser werden die Dateien per AJAX oder in einem versteckten Iframe übertragen.

jquery-autocomplete.js (bundled)

Dieses JavaScript enthält ein JQuery-Plugin, mit dem Eingabefelder mit einer Vorschlagsfunktion ausgestattet werden können. Die Vorschlagswerte werden asynchron von dem angegebenen REST-Service geladen.

jquery-format.js (bundled)

Dieses JavaScript enthält ein jQuery-Plugin, das die Formatierung und Analyse von Daten und Zahlen ermöglicht. Dabei handelt es sich um eine JavaScript-Alternative der Java-Klassen *SimpleDateFormat* und *NumberFormat*.

JSON (npm-package)

Formcentric verwendet das native *JSON*-Objekt moderner Browser, um JSON-Daten zu parsen und zu schreiben. Bei älteren Browsern, die das *JSON*-Objekt nicht unterstützen, wird das Objekt durch dieses JavaScript bereitgestellt.

jquery-webforms.js

Dieses JavaScript enthält ein jQuery-Plugin, das die von Formcentric benötigten JavaScript-Funktionen bereitstellt. Die variablen Konfigurationsparameter des Plugins können Sie wie nachfolgend dargestellt auf dem *form*-Tag im Freemarker-Template *WebForm.ajax.ftl* angeben.

```
<#assign conditions=fc.conditions() />
<#assign calculatedValues=fc.calculatedValues() />

<form method="post" ...
    data-mwf-form="${self.shortId}"
    data-mwf-settings='{
        "url":"${targetUrl}",
        "statisticsUrl":"${statisticsUrl!""}",
        "query":"navigationId=${cmpage.navigation.contentId}",
        "calculatedValues" : ${calculatedValues},
        "conditions" : ${conditions}
    }'>
```

Die Konfiguration erfolgt durch Angabe eines JSON-Strings im Attribut *data-mwf-settings*, der die nachfolgenden Parameter enthalten kann.

Parameter	**Beschreibung**
conditions	Typ: *JSON* JSON-Definition der clientseitig auszuwertenden Bedingungen.
calculatedValues	Typ: *JSON* JSON-Definition der *Berechneten Werte* die clientseitig berechnet werden.
appendUrlVars	Typ: *Boolean* Mit diesem Parameter legen Sie fest, dass die URL-Parameter der umgebenden Seite in den AJAX-Request an den Formular-Controller übernommen werden.
createOption	Typ: *Function($form, entry, selected)* Funktion, die ein Option-Element einer dynamischen Auswahlliste (*comboBox*) erzeugt.
createRadio	Typ: *Function($form, name, entry, checked)* Funktion, die einen Radio-Button einer dynamischen Einfachauswahl (*radioGroup*) erzeugt.
createCheckBox	Typ: *Function($form, name, entry, checked)* Funktion, die einen Checkbox-Button einer dynamischen Mehrfachauswahl (*checkBoxGroup*) erzeugt.
createUploadFileRow	Typ: *Function(Object $form, Object attr, file)* Diese Funktion erzeugt einen neuen Eintrag in der Dateiliste des FileUpload-Elements, bevor die Datei hochgeladen wurde.
createDownloadFileRow	Typ: *Function(Object $form, Object attr, file)* Diese Funktion erzeugt einen neuen Eintrag in der Dateiliste des FileUpload-Elements, nachdem die Datei hochgeladen wurde.
updateCalculatedValue	Typ: *Function($form, id, value)* Funktion, die die Darstellung eines *calculatedValues* aktualisiert, wenn dieser neu berechnet wurde.
updateFormValue	Typ: *Function($form, $elem, name, l)* Diese Funktion aktualisiert den Wert eines Formularfelds in der Zusammenfassung, wenn dieser vom Anwender eingegeben oder verändert wurde. Wenn es sich bei dem Formularfeld um eine Auswahl handelt, werden die Labels der

Parameter	Beschreibung
	ausgewählten Optionen im Parameter *l* übergeben. Anderenfalls wird der eingegebene Text übergeben.
onFillDropdown	Typ: *Function(Object $form, Object $elem))* Callback-Funktion, die aufgerufen wird, nachdem eine dynamsiche Dropdown-Liste gefüllt wurde.
onFillSelection	Typ: *Function(Object $orm, Object $elem))* Callback-Funktion, die aufgerufen wird, nachdem eine dynamsiche Auswahlliste gefüllt wurde.
onInit	Typ: *Function(Object $form)* Callback-Funktion die aufgerufen wird, nachdem das jQuery-Plugin initialisiert wurde. Nutzen Sie diese Funktion, um eigene Initalisierungen durchzuführen.
onSubmit	Typ: *Function(Object $form, String url, String query)* Callback-Funktion die aufgerufen wird, wenn das Formular abgesendet wird. Nur wenn die Funktion den boolschen Wert *true* zurückgibt, wird das Formular abgesendet.
onSuccess	Typ: *Function(Object $form, Object data, String status, Object jqXHR)* Callback-Funktion die aufgerufen wird, nachdem das Formular erfolgreich abgesendet wurde.
onAjaxError	Typ: *Function(jqXHR jqXHR, String status, String error)* Funktion die aufgerufen wird, wenn bei einem AJAX-Request ein Fehler aufgetreten ist. Standardmäßig erzeugt diese Funktion einen Eintrag im Fehlerprotokoll des Browsers.
operations.visible	Type: *Function(Object $form, Object field)* Funktion, mit der Eingabefelder eingeblendet werden.
operations.hidden	Type: *Function(Object $form, Object field)* Funktion, mit der Eingabefelder ausgeblendet werden.
operations.alterable	Type: *Function(Object $form, Object field)* Funktion, mit der Eingabefelder von schreibgeschützt auf änderbar gesetzt werden.
operations.readonly	Type: *Function(Object $form, Object field)* Funktion, mit der Eingabefelder von änderbar auf schreibgeschützt gesetzt werden.
operations.enabled	Type: *Function(Object $form, Object field)*

Parameter	Beschreibung
	Funktion, mit der Eingabefelder von deaktiviert auf aktiviert gesetzt werden.
operations.disabled	Type: *Function(Object $form, Object field)* Funktion, mit der Eingabefelder von aktiviert auf deaktiviert gesetzt werden.
operations.optional	Type: *Function(Object $form, Object field)* Funktion, mit der Eingabefelder als optional gekennzeichnet werden.
operations.mandatory	Type: *Function(Object $form, Object field)* Funktion, mit der Eingabefelder als Pflichtfelder gekennzeichnet werden.

Die Standardimplementierungen der aufgeführten JavaScript-Funktionen finden Sie im JavaScript *jquery-webforms.js*.

Im folgenden Beispiel werden die Funktion *operations.mandatory* durch eine angepasste Version ersetzt.

```
<#assign conditions=fc.conditions() />
<#assign calculatedValues=fc.calculatedValues() />

<form method="post" ...
  data-mwf-form="${self.shortId}"
  data-mwf-settings='{
    "url":"${targetUrl}",
    "statisticsUrl":"${statisticsUrl!""}",
    "calculatedValues" : ${calculatedValues},
    "conditions" : ${conditions},
    "operations": {
      "mandatory": "function ($form, field) {\r\n
      var $label = $form.find('label[for=\"' + field.input + '\"]'),\r\n
      $span = $('<span>').attr('class', 'mwf-required').text('*');\r\n
      $label.children('span.mwf-required').remove();\r\n
      $label.append($span);};
      }"
    }
  }'>
```

Ab Version 5.6.2.CM9 haben Sie auch die Möglichkeit die oben aufgeführten Konfigurationsparameter in einer separaten JavaScript-Datei anzugeben. Dies ist insbesondere bei der Angabe von JavaScript-Funktionen der einfachere Weg, da das fehleranfällige Maskieren der reservierten Zeichen entfällt.

```
(function($) {
  $.fn.webforms.defaults().operations.mandatory =
```

```
    function($form, field) {
      var $label = $form.find('label[for="' + field.input + '"]'),
          $span = $('<span>').attr('class', 'mwf-required').text('*');
          $label.children('span.mwf-required').remove();
          $label.append($span);
      };
})(jQuery);
```

Event-Referenz

Das Formcentric jQuery-Plugin stellt eine Reihe von Events zur Verfügung, die es Ihnen ermöglichen, auf bestimmte Ereignisse zu reagieren. Die zugehörigen Event-Handler müssen auf dem *document* Object registriert werden.

Event-abhängige Detailinformationen wie beispielsweise das zugehörige Formularelement werden im Event-Object *event.details* an den Event-Handler übergeben.

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

Die nachfolgende Tabelle beschreibt die Ereignisse, auf die Sie hören und programmgesteuert reagieren können.

Event-Name	**Detailinformationen**	**Wird abgesendet, wenn**
mwf-ajax-finished	*$dest*	die Funktion *mwfAjaxReplace* erfolgreich ausgeführt wurde.
mwf-ajax-error	*$dest, jqXHR, textStatus, errorThrown*	der asynchrone Aufruf (AJAX-Call) fehlschlägt.
mwf-fill-dropdown	$form, $elem	eine Auswahlliste durch eine Datenquelle befüllt wurde.
mwf-fill-selection	$form, $elem	eine Einfach- oder Mehrfachauswahl durch eine Datenquelle befüllt wurde.
mwf-fill-hidden	$form, $elem	ein verstecktes Feld durch eine Datenquelle befüllt wurde.
mwf-suggestion-selected	$form, $elem, id, selection, params	bei einem Eingabefeld ein Vorschlag ausgewählt wurde.
mwf-value-changed	$form, $elem, name, value	sich der Wert eines Formularfelds ändert.

Event-Name	Detailinformationen	Wird abgesendet, wenn
mwf-form-replaced	*$form, id*	das Formular abgesendet wurde.

6.3. Erweiterung der Server-Anwendung

Bei dem Formcentric Headless-Server handelt es sich um eine Spring-Boot Anwendung, die eine REST-Schnittstelle mit verschiedenen Endpunkten für die Formularverarbeitung bereitstellt. Für die browser-seitige Anbindung des Formcentric Headless-Servers steht Ihnen ein vorgefertigter React-Client zur Verfügung, den Sie ebenfalls an Ihre Anforderungen anpassen können (siehe auch Abschnitt 6.4, „Formcentric Client“).

Der folgende Abschnitt beschreibt, wie der Formcentric Headless-Server funktional erweitert werden kann. Bitte beachten Sie, dass die Namen der Framework-Klassen teilweise mit denen der CAE-Integration übereinstimmen, aber unterhalb des Package `com.formcentric.headless.rest` liegen.

6.3.1. Implementierung einer Action

Analog zur CAE-Integration verwendet auch der Formcentric-Server Actions, in denen die Business-Logik der Formulardatenverarbeitung gekapselt ist. Diese Klassen implementieren das Interface `com.formcentric.headless.actions.Action`. Durch die Entwicklung einer Custom-Action sind Sie in der Lage beliebige Backend-Systeme anzubinden. Da es sich bei den Actions um Spring-Beans handelt, können Konfigurationsparameter über standard Spring-Mechanismen an die Action übergeben werden. Das nachfolgende Beispiel zeigt Ihnen, wie Sie eine *CustomAction* implementieren und konfigurieren können.

```
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";
  }
}
```

6.3.2. Variable zur Vorbelegung von Formularfeldern hinzufügen

Neben den vordefinierten Variablen haben Sie auch die Möglichkeit eigene Variablen zur Vorbelegung von Formulafeldern hinzuzufügen. Registrieren Sie hierzu ein Spring-Bean vom Typ *VariablesService* im Application-Context des Headless-Servers.

```
import com.formcentric.headless.services.VariablesService;
import com.formcentric.headless.model.WebForm;
import org.springframework.stereotype.Service;
import javax.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;
  }
}
```

Da es sich bei dem *VariablesService* um ein Spring-Beans handelt, können Sie bei der Erzeugung der Variablen auch auf externe System oder Dienste zugreifen.

In manchen Anwendungsszenarien ist es erforderlich, Formularfelder mit Werten aus der Webseite oder Client-Anwendung vorzubelegen, in die das Formular eingebettet ist. Für diesen Anwendungsfall ist es ausreichend, die Variablen im Data-Attribut *data-fc-vars* des *div*-Tags anzugeben, an den das Formular gebunden ist.

```
<div
    data-fc-id="1249010"
    ...
    data-fc-vars='{"custom_var":"custom_value"}'
></div>
```

6.3.3. Implementierung eines REST-Services

Die in Abschnitt Abschnitt 6.2.4, „Implementierung eines REST-Services“ beschriebenen REST-Services stehen Ihnen grundsätzlich auch bei Einsatz des Formcentric Head-

less-Servers zur Verfügung. Das nachfolgende Beispiel zeigt Ihnen einen REST-Service, der eine Map mit statischen Key/Value-Paaren erzeugt.

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

@Bean
public class ExampleRestService extends BaseRestService {

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

    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;
  }

  @Override
  public String name() {
    return "exampleRestService";
  }
}
```

Bei Aufruf der *invoke*-Methode werden dem RestService neben dem ServiceContext auch die aktuellen Benutzereingaben (Parameter *formData*) übergeben. Damit sind Sie in der Lage, direkt auf die Eingaben des Benutzers zu reagieren.

Über den *ServiceContext* haben Sie zudem Zugriff auf die Formulardefinition, das Eingabeelement, die Konfigurationsparameter des RestService und das Request-Object.

Alle Formularelemente, zu denen auch die REST-Services gehören, müssen über einen eindeutigen Namen verfügen, mit dem Sie innerhalb der Formulardefinition referenziert werden können. Der Name wird beim Start der Anwendung durch Aufruf der Methode `name()` ermittelt.

Durch die *@Bean*-Annotation wird Ihr REST-Service automatisch durch Spring instanziiert und unter dem festgelegten Namen registriert.

6.4. Formcentric Client

Zur Darstellung von Formcentric-Formularen im Browser ist das NPM-Modul `@formcentric/client` (https://www.npmjs.com/package/@formcentric/client) erforderlich. Dies

gilt sowohl für Projekte, die auf reinem HTML und JavaScript basieren als auch für Projekte, die Frontend-Frameworks oder Frontend-Libraries verwenden.

Das installierte Package enthält verschiedene Varianten von Modulen für unterschiedliche Anwendungen. Die Installation der benötigten Dateien erfolgt über NPM, wobei dieses keine eigenen Abhängigkeiten aufweist und auch ohne Bundler verwendet werden kann.

Für die Installation führen Sie folgendes Kommando aus:

```
npm install @formcentric/client
```

oder alternativ

```
pnpm install @formcentric/client
```

Damit ein Formular korrekt angezeigt werden kann, müssen folgende Dinge vorhanden sein:

1. Ein *div*-Tag mit einem *fc-id* Data-Attribut, in das das Formular gerendert werden soll.
2. Ein geladenes Theme, bestehend aus CSS, Templates und CSS-Custom-Properties, sofern diese in der CSS-Datei benutzt werden.
3. Die *formapp.js* muss zugänglich sein, um als Script-Tag eingebunden zu werden.

```
<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 ... ffUuAm4OyEQsC9"
    data-fc-refs="ffUuAm4OyEQsC9 ... 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>
```

6.4.1. Theme

Um sicherzustellen, dass das Formular korrekt angezeigt wird, muss das Theme-CSS geladen werden. Dies kann durch die Verwendung eines Link-Tags im HTML-Head und die Verwendung von Custom-Properties erreicht werden. Wenn vorhanden, müssen diese im HTML-Code gesetzt sein.

Jedes Eingabefeld hat sein eigenes Template, das angepasst werden kann. Diese Templates werden in einer JS-Datei namens *formcentric_templates.js* auf dem *Window*-Objekt defi-

niert. Dadurch werden sie später beim Rendern des Formulars gefunden. Die Templates sind zwingend erforderlich, um das Formular korrekt darzustellen (siehe Abschnitt 6.4.3, „Templates“).

6.4.2. Initialisierung

Um den Client zu starten, kann entweder dass Skript *formcentric.js* geladen werden, oder nachdem dieses geladen wurde zu einem späteren Zeitpunkt `window.formcentric.initFormcentric()` aufgerufen werden. Über das Data-Attribut `fc-data-url` können Sie die URL für den Zugriff auf den Formcentric Headless Server konfigurieren.

6.4.3. Templates

Template bestehen immer aus einer Funktion, deren Rückgabewert vom Formcentric Client für das Rendern von HTML-Code verwendet wird. Hierbei übergibt der Formcentric Client zwei Parameter (html und props) an eine Template Funktion. Die genaue Struktur und die verwendeten Parameter hängen von der spezifischen Verwendung des Templates ab.

html: Ein Template-Literal-Tag, welcher verwendet wird, um HTML-Code zu rendern. Dieser Parameter ermöglicht das Einbetten von HTML im JavaScript-Code des Templates.

props: Ein Objekt, das die Eigenschaften des Formularfelds enthält. Diese bestehen aus vom Formcentric Client berechneten Werten und aus über den Editor eingestellten Felddaten. Die spezifischen Eigenschaften variieren je nach Formularfeld.

Das fertige HTML wird durch die Kombination von statischem HTML-Code und den Werten der *props*-Eigenschaften erstellt. Dies kann durch Interpolation, Zusammenführung von Strings oder Verwendung von Funktionen zum Rendern von HTML erfolgen. Das hieraus entstehende HTML wird dann als Rückgabewert der Template-Funktion an den Formcentric Client zurückgegeben, welcher es dann in den DOM rendert.

Alle Template Funktionen können auch asynchron als Promise vom Formcentric Client verarbeitet werden.

Template Properties

Template Properties werden vom Formcentric Client als Parameter (*props*) an die entsprechende Template Funktion übergeben. Diese enthalten alle nötigen Informationen für die Anzeige und das Verhalten der jeweiligen Formularfelder.

Folgende Properties werden an die Templates übergeben:

Property	**Beschreibung**
key	Die eindeutige ID des Elements
oninput	(event: InputEvent) => void Aktualisiert den Wert des Felds

Property	Beschreibung
onfocus	(event: FocusEvent) => void Gibt gegebenenfalls vorhandene Validierungsfehler des Felds zurück
onclick	Eine Funktion, die über den Editor definierte *onClick* Funktionen evaluiert
fieldSuccess	Ein boolescher Wert, der angibt, ob das Element erfolgreich validiert wurde und keine Fehler enthält.
fieldError	Ein Objekt, das Informationen über einen Fehler im Element enthält.
properties	Ein Objekt, das die Eigenschaften des Feld-Elements enthält
components	Ein Objekt, das Komponenten für bestimmte Feldtypen enthält. Diese Komponenten werden verwendet, um das Element im Template zu rendern. Es enthält Komponenten für *captcha, fileUploader, comboBox, suggestions, hint, datePicker und markdown*
fieldSetFields	Ein Array, welcher die Felder eines *FieldSets* enthält
layoutFields	Ein Array, welcher die Felder eines *FieldSets* enthält
summaryFields	Ein Array welches die Informationen von Feldern, die in einem *SummaryField* festgelegt wurden, enthält
fieldEmptyText	Ein Standardtext, der angezeigt wird, wenn ein *SummaryField* keine Werte enthält
contentMarkup	Eine Content-Komponente, welche von einer über das *form*-Div festgelegten Funktion zurückgegeben wird
hasService	Ein boolescher Wert, der angibt, ob das Element einen REST-Service hat
setRESTParams	(params: Record<string,any>) => void: Eine Funktion mit der die Parameter für den REST-Service des Elements festgelegt werden können.

Element: In den Properties sind außerdem alle Informationen über das darzustellende Element enhalten.

```
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)

}
```

Feldtypen:

```
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'
```

Validatoren:

```
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: In *props.properties* sind alle HTML Attribute und Feld-Properties aus der Formulardefinition enthalten. Diese werden vom Formcentric Client zum Beispiel durch Bedingungen berechnet oder vom Redakteur an dem entsprechenden Formularfeld eingestellt. Die nachfolgende Tabelle zeigt eine Übersicht der möglichen Properties:

Property	**Beschreibung**
hint	Ein optionaler Hinweistext, der dem Benutzer weitere Informationen oder Anweisungen gibt.
placeholder	Ein optionaler Platzhaltertext, der in einem Eingabefeld angezeigt wird, wenn kein Wert eingegeben wurde.
selected	Ein boolescher Wert, der angibt, ob das Element standardmäßig ausgewählt ist.
errormessage	Eine optionale Fehlermeldung, die angezeigt wird, wenn das Element ungültig ist.
multiple	Ein boolescher Wert, der angibt, ob mehrere Werte für dieses Element ausgewählt werden können.
auto_upload	Ein boolescher Wert, der angibt, ob eine automatische Hochladefunktion aktiviert ist.
datasource	Eine Zeichenkette, die eine Datenquelle angibt.
datasource_params	Eine Zeichenkette, die Parameter für die Datenquelle angibt.
dynamic	Ein boolescher Wert, der angibt, ob das Element dynamisch ist und seine Eigenschaften zur Laufzeit geändert werden können.
visible	Ein boolescher Wert, der angibt, ob das Element sichtbar sein soll.
hidden	Ein boolescher Wert, der angibt, ob das Element ausgeblendet werden soll.
writable	Ein boolescher Wert, der angibt, ob das Element beschreibbar sein soll.
readonly	Ein boolescher Wert, der angibt, ob das Element schreibgeschützt sein soll.
optional	Ein boolescher Wert, der angibt, ob das Element optional ist.
mandatory	Ein boolescher Wert, der angibt, ob das Element obligatorisch ist.

Property	Beschreibung
disabled	Ein boolescher Wert, der angibt, ob das Element deaktiviert sein soll.
enabled	Ein boolescher Wert, der angibt, ob das Element aktiviert sein soll.
type	Eine Zeichenkette, die den Typ des Elements angibt.

Components

In *props.components* stehen interne Komponenten zur Auswahl, welche die Arbeit mit einzelnen Formularfeldern vereinfachen sollen, wenn kein Bedarf nach Anpassung der Funktionalität dieser Felder besteht. Folgende Komponenten sind vorhanden:

Property	Beschreibung
captcha	Lädt Captcha-Bilder vom Headless-Server
combobox	Stellt Auswahllisten dar
datePicker	Stellt einen Date Picker dar
fileUploader	Stellt einen Upload-Dialog dar
hint	Stellt Hinweise dar
markdown	Stellt Markdown- als HTML-Elemente dar
suggestions	Stellt Vorschläge von REST-Services als Auswahlliste unter Eingabefeldern dar

Folgende Properties können an die oben genannten Komponenten weitergegeben werden:

captcha:

Property	Beschreibung
buttonText	Eine optionale Zeichenfolge für den Refresh-Button der Captcha Komponente. Wird diese nicht angegeben, dann enthält der Button ein Icon.

combobox:

Property	Beschreibung
alle	Alle Properties aus dem Parameter des Templates müssen übergeben werden.

datepicker:

Property	Beschreibung
alle	Alle Properties aus dem Parameter des Templates müssen übergeben werden.

fileUploader:

Property	Beschreibung
trigger	Die Klasse oder ID des Trigger-Elements
inline	Ein optionaler boolescher Wert, der angibt, ob das Element in einer Inline-Darstellung angezeigt werden soll.

hint:

Property	Beschreibung
alle	Alle Properties aus dem Parameter des Templates müssen übergeben werden.

markdown:

Property	Beschreibung
markdown	Das *markdown* Property nimmt stringifiziertes Markdown als Wert an.

suggestions:

Property	Beschreibung
Alle	Alle Properties aus dem Parameter des Templates müssen übergeben werden.

Die Komponenten werden innerhalb des HTML-Template-Literal-Tags in den Templates ausgeführt:

```
${ props.components.captcha( {…} ) }
```

Anpassung und Erweiterung von Templates

Um die Templates zu erweitern, können Sie die HTML-Elemente und Klassen innerhalb der Vorlagen anpassen. Dies ermöglicht es Ihnen, das Erscheinungsbild und das Verhalten der Komponenten anzupassen.

Sie können Klassen hinzufügen oder entfernen, um das Styling anzupassen, oder zusätzliche HTML-Elemente einfügen, um zusätzliche Funktionalitäten bereitzustellen.

Darüber hinaus können die Templates auch asynchron ausgeführt werden, was bedeutet, dass Sie Daten abrufen und anzeigen können, die nicht direkt vom Formcentric-Client an die Templates übergeben werden. Dies eröffnet Möglichkeiten zur Integration mit APIs oder anderen externen Datenquellen.

Um asynchrone Daten in den Templates zu verwenden, können Sie JavaScript-Funktionen wie *fetch* verwenden, um Daten von einem Server abzurufen. Sie können dann die erhaltenen

Daten in den Vorlagen anzeigen, indem Sie die entsprechenden Variablen oder Platzhalter verwenden.

Erweiterungsmöglichkeiten:

1. Unterstützung für eine oder mehrere benutzerdefinierte CSS-Klassen: Es können optionale CSS-Klassen hinzugefügt werden, um das Styling des Eingabeelements anzupassen. Dazu kann eine Custom-Klasse in der *className*-Definition verwendet werden:

```
input className="customClass" />
```

2. Anpassung des Markups: Es können ohne weiteres neue Markup-Elemente hinzugefügt oder bereits vorhandene Elemente angepasst werden:

```
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(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>
```

6.4.4. Troubleshooting

Kontrollieren Sie das Browser-Log. Wenn dort keine Ausgaben des Clients zu finden sind, wurde das Skript *formcentric.js* nicht geladen bzw. ausgeführt.

Der Hinweis, dass kein Form-*div* gefunden wurde kann zwei Ursachen haben:

1. Es wurde kein *div*-Tag mit dem Data-Attribut *fc-id* gefunden
2. Das Skript *formcentric.js* wurde ohne Angabe des *defer*-Attributs geladen.

Das kein Formular angezeigt wird, obwohl ein Form-*div* gefunden wurde, kann folgende Ursachen haben:

1. Es wurde kein *div*-Tag mit dem Data-Attribut *fc-id* gefunden
2. Das Skript *formapp.js* wurde nicht geladen.
3. Ein fehlendes Template hat den Start des Clients verhindert.

6.5. Spezielle Integrationsszenarien

In den meisten Anwendungsfällen wird das *@formcentric/client*-Skript einfach geladen und ausgeführt. Es gibt jedoch besondere Szenarien, wie etwa bei Single Page Applications (SPA), in denen das Skript erst ausgeführt werden muss, wenn der DOM-Baum vollständig aufgebaut ist. In solchen Fällen bietet sich ein dynamischer Import des Skripts an, der genau dann ausgeführt wird, wenn der virtuelle DOM vollständig aufgebaut wurde. Dieser Zeitpunkt kann je nach SPA-Framework variieren.

```
function App() {
    const ref = useRef(null);

    const formDef = "TGU5Kmxe4svPaahc2aSm-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>
    );
}
```

Falls kein dynamischer Import möglich ist, kann nach dem Laden des Skripts die Funktion *initFormcentric* des *window.formcentric*-Objekts aufgerufen werden.

```
window.formcentric.initFormcentric()
```