FirstSpirit™
Your Content Integration Platform

Entwicklerhandbuch für Komponenten

FirstSpirit Version 4.0, 4.1 und 4.2

Version	**1.2**
Status	**RELEASED**
Datum	**2010-09-21**
Abteilung	FS-Core
Autor/ Autoren	A. Pfeiler
Copyright	2010 e-Spirit AG

MODDEV40DE_FirstSpirit_ModulDeveloperDoc

e-Spirit AG

Barcelonaweg 14
44269 Dortmund | Germany

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

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

e-Spirit AG

Inhaltsverzeichnis

1 Thema dieser Dokumentation

Diese Dokumentation soll die Modul-/Komponenten-Entwicklung für FirstSpirit™ unterstützen und einen Einblick in die Konzeption geben. Es beschreibt die grundlegenden Designprinzipien, die Modul-Descriptor-Syntax und die Integration in FirstSpirit™. Es enthält Beispiel-Implementierungen mit den wichtigsten Codeblöcken der in FirstSpirit™ möglichen Komponenten-Typen (siehe Kapitel 2.9.1 Seite 16).

Kapitel 2 erläutert die Verwendung von Modulen und Komponenten in FirstSpirit inklusive Installation und Deinstallation, Konfiguration, Aktivierung und Aktualisierung, Import und Export von Modulen. Darüber hinaus werden die Bestandteile von Modulen und verschiedene Komponenten-Typen vorgestellt (siehe Kapitel 2 ab Seite 7).

In **Kapitel 3** werden Komponenten-Typen näher beschrieben (siehe Kapitel 3 ab Seite 26). Die **Kapitel 3.5**, **3.6**, **3.7** und **3.16** beschäftigen sich mit FSM-Archiven, dem Modul- und dem Komponenten-Descriptor. **Kapitel 3.9** führt in das FirstSpirit GUI Object Model (GOM) ein, **Kapitel 3.10** geht auf Sicherheitsaspekte ein, **Kapitel 3.11** gibt ein Beispiel für eine einfache Editor-Komponente. Die **Kapitel 3.17** und **3.18** beschäftigen sich mit dem Signieren und Lokalisieren von Modulen.

2 FirstSpirit Modul-Grundkonzeption

In FirstSpirit wird das Handling von Modulen/Komponenten neu organisiert. Zentrale Einheit ist das Modul, welches wiederum aus einer oder mehreren Komponenten besteht. Jede Komponente nutzt Ressourcen (shared, web-local – z.B. Klassen, Jar-Archive, Properties-Dateien), hat eine bestimmte Sichtbarkeit/Scope (siehe Kapitel 2.9.2 Seite 20), besitzt einen Typ (siehe Kapitel 2.9.1 Seite 16) und Eigenschaften (siehe Kapitel 3.7.1 Seite 36). Es gibt drei Sichtbarkeitsstufen und sieben verschiedene Komponenten-Typen, die wiederum miteinander in einem Modul

kombiniert werden können.

Jedes Modul enthält eine Descriptor-Datei (module.xml, siehe Kapitel 3.6 Seite 33), die alle Komponenten-Ressourcen, Sichtbarkeits-Stufen, Versionen einzelner Komponenten und Ressourcen sowie deren Abhängigkeiten, Abhängigkeiten zu anderen Modulen oder Modul-Herstellern, Modul- sowie Komponenten-Namen beschreibt. Jedes FirstSpirit-Modul-Archiv hat die Dateinamen-Erweiterung `*.fsm` (siehe Kapitel 3.5 Seite 32).

2.1 Installation von Modulen

Ausgangspunkt für die FirstSpirit Modul-Installation ist die Server- und Projektkonfiguration. Über diese wird das Modul auf dem FirstSpirit-Server installiert. Die Ressourcen der system-weiten Komponenten stehen über den Server-Classloader (siehe Kapitel 2.10 Seite 21) ab diesem Zeitpunkt zur Verfügung. Es gibt *system-weite*, *web-lokale* sowie *projekt-lokale* Modul-Komponenten. Auf diese drei Sichtbarkeits-Typen wird im Verlauf dieses Dokumentes näher eingegangen (siehe Kapitel 2.9.2 Seite 20).

2.1.1 Konfiguration von Modul-Komponenten

Nach erfolgreicher Modul-Installation besteht die Möglichkeit, den ausgewählten Projekten projekt- oder web-lokale Komponenten (siehe Kapitel 2.9.1 Seite 16) über die Projekteigenschaften hinzuzufügen. Optional können diese Komponenten konfiguriert werden (falls vorgesehen und im Modul-Descriptor definiert, `<configurable>`, siehe Kapitel 2.9.1.5 Seite 18), entweder mit einer von der Komponente erzeugten oder einer generischen GUI. Letztere zeigt die Parameter des Moduls und unterstützt dabei Text und numerische Werte.

2.1.2 Aktivierung von Modul-Komponenten

Web-lokale Komponenten müssen nach ihrer Konfiguration aktiviert werden. In diesem Schritt werden alle Web-Anwendungen eines Projektes zusammengefasst und abhängig vom verwendeten Web-Server deployed. Wird der interne Web-Server eingesetzt, ist nichts weiter zu tun. Generische Web-Server können das automatische Deployment anbieten, wenn ein entsprechendes Deployment-Script hierfür existiert. Andernfalls wird eine War-Datei erzeugt, die heruntergeladen und manuell installiert werden muss.

2.2 Aktualisierung eines Moduls

Identisch zur „Installation eines Moduls“ wird bei der Aktualisierung das Modul über die Server- und Projektkonfiguration auf dem FirstSpirit-Server installiert. Die Laufzeitdateien projekt-lokaler Komponenten werden nur dann automatisch aktualisiert, wenn die Komponente(n) beim Hinzufügen in die Projekte entsprechend markiert wurden. Die Aktivierung von Web-Anwendungen kann ebenfalls über die Server- und Projektkonfiguration angestoßen werden.

Da nach der Aktualisierung eines Moduls die Konfigurationen und Laufzeitdateien projekt-lokaler Komponenten potentiell in einer älteren Version vorliegen können, müssen Module immer mit den Dateien ihrer Vorgängerversionen umgehen können (lesend und schreibend). Das gilt vor allem für die Konfiguration und die dazugehörige GUI – das Modul muss diesen Anforderungen genügen.

2.3 Deinstallation eines Moduls

Die Modul-Deinstallation wird ebenfalls über die FirstSpirit Server- und Projektkonfiguration durchgeführt. Hierbei werden alle im Modul-Descriptor definierten Komponenten und Ressourcen-Einträge entfernt. Alle dem Modul nicht bekannten bzw. im Modul-Descriptor nicht definierten Ressourcen können beispielsweise über die Definition einer eigens für diese Aufgabe vorgesehenen Modul-Klasse `<class>` (siehe Kapitel 2.6 Seite 11) deinstalliert werden. Im Umkehrschluss gilt dieses auch für die Installation sowie Aktualisierung.

2.4 Import und Export von Modulen

Bei Projektexport-Operationen werden Konfigurations- und Laufzeitdateien *projekt-lokaler* Komponenten ebenfalls exportiert. Während eines Imports besteht die Möglichkeit, ein projekt-lokales Modul dem importierten Projekt hinzuzufügen und die Konfiguration zu übernehmen. Voraussetzung hierfür ist jedoch, dass das Modul auf dem Server installiert ist und in einer gleichen oder einer neueren Version vorliegt.

2.5 Modul-Bestandteile

Im Kern besteht ein Modul aus einem Modul-Descriptor (siehe Kapitel 3.6 Seite 33), Ressourcen und einer bis mehreren Komponenten, wobei jede Komponente wiederum Ressourcen enthalten kann.

2.5.1 Ressourcen

Klassen und andere Ressourcen werden im Modul und in den Komponenten in `<resource>`-Einträgen definiert. Diese verweisen auf eine Jar-Datei oder ein Verzeichnis. Um diese Einträge im Classloader verwenden zu können, werden sie von FirstSpirit ausgepackt (Modul-Installation, Serverstart) und in ein temporäres Verzeichnis gelegt. Daher sollten die Ressourcen möglichst minimal gewählt werden.

```
1.    <resources>
2.        <resource>/libs/exmod.jar</resource>
3.        <resource>files/</resource>  <!-- Wichtig: "/" am Ende des
4.                                          Verzeichnisnamens -->
5.        <resource>libs/simple.jar</resource>
6.    </resources>
```

Listing 1: Ressourcen im Module-, Komponenten-Descriptor

Zur Nutzung namensgleicher Modul-Resourcen siehe Kapitel 3.15 ‚Namensgleichheit bei Modul-Ressourcen (z.B. Zip-Exportdateien)' Seite 123.

Alle Ressourcen, einschließlich Zip-Dateien etc., sollten immer mit `MyClass.class.getRessourceAsStream(...)` geladen werden.

2.5.1.1 Versionierung

Ressourcen-Einträge können mit bestimmten Attributen versehen werden, welche einen eindeutigen Bezeichner sowie die mitgelieferte Version (**`version`**) und die zur aktuellen Version minimal (**`minVersion`**) und maximal (**`maxVersion`**) kompatible Versionsnummer definieren.

```
1.    <!-- Datei mit Versionsangabe -->
2.    <resource name="myLib" version="1.2.7" minVersion="1.2"
maxVersion="1.2.999" >libs/myLib.jar</resource>
```

Listing 2: Versionierung von Ressourcen im Modul-, Komponenten-Descriptor Komponenten

Diese Informationen helfen dabei, doppelte Ressourcen bei der Kombination mehrerer Komponenten zu vermeiden. Dies kann beispielsweise bei system-weiten Komponenten oder bei Web-Anwendungen auftreten.

2.5.1.2 Komponenten

Ein Beispiel für mehrere in einem FirstSpirit-Modul gekapselte Komponenten ist eine Editor-Komponente (erweitert den FirstSpirit-Client um Eingabemöglichkeiten, ist immer global d.h. serverweit) und eine Service-Komponente (eine Server-Komponente, die eine öffentliche Schnittstelle besitzt und somit über Eingabekomponenten oder Scripte angesprochen werden kann; ist immer global, d.h. serverweit). Ein einfacher Anwendungsfall für ein derartiges Modul bzw. zwei solcher Komponenten wäre z.B. das Einlesen, Bearbeiten, Speichern einer serverseitigen Datei: Einlesen der Datei durch den Service, Bearbeiten durch den Editor in z.B. einer (J)Textarea und anschließendes Speichern derselbigen Datei durch den Editor bzw. durch Auslösen der „Speichern"-Aktion des Editors und anschließender Weitergabe an den Service.

2.6 Modul-Ereignisbehandlung

Auf bestimmte Ereignisse wie Installation, Deinstallation und Aktualisierung durch Benutzer in der Administrationsoberfläche kann durch eine spezialisierte Modul-Klasse `<class>` reagiert werden.

```
<class>de.espirit.firstspirit.module.AnohterClass</class>
```

Des Weiteren implementiert jede Komponente die Methoden `installed()`, `updated()` und `uninstalling()`, siehe auch Kapitel 3.4 Seite 29.

Bestehen Abhängigkeiten zu anderen Modulen, müssen die Modulnamen als `<depends>`-Einträge definiert werden.

```
1.    <dependencies>
2.        <depends>AnotherModule</depends>
3.    </dependencies>
```

2.7 Modul-, Komponenten-Verzeichnisstruktur

Die Verzeichnisstruktur der Module und Komponenten ist an FirstSpirit angepasst, alle Daten-, Konfigurations- und Logverzeichnisse sind voneinander getrennt. Nachfolgend ist die Struktur ausgehend vom FirstSpirit Root-Verzeichnis *($cmscroot)* dargestellt:

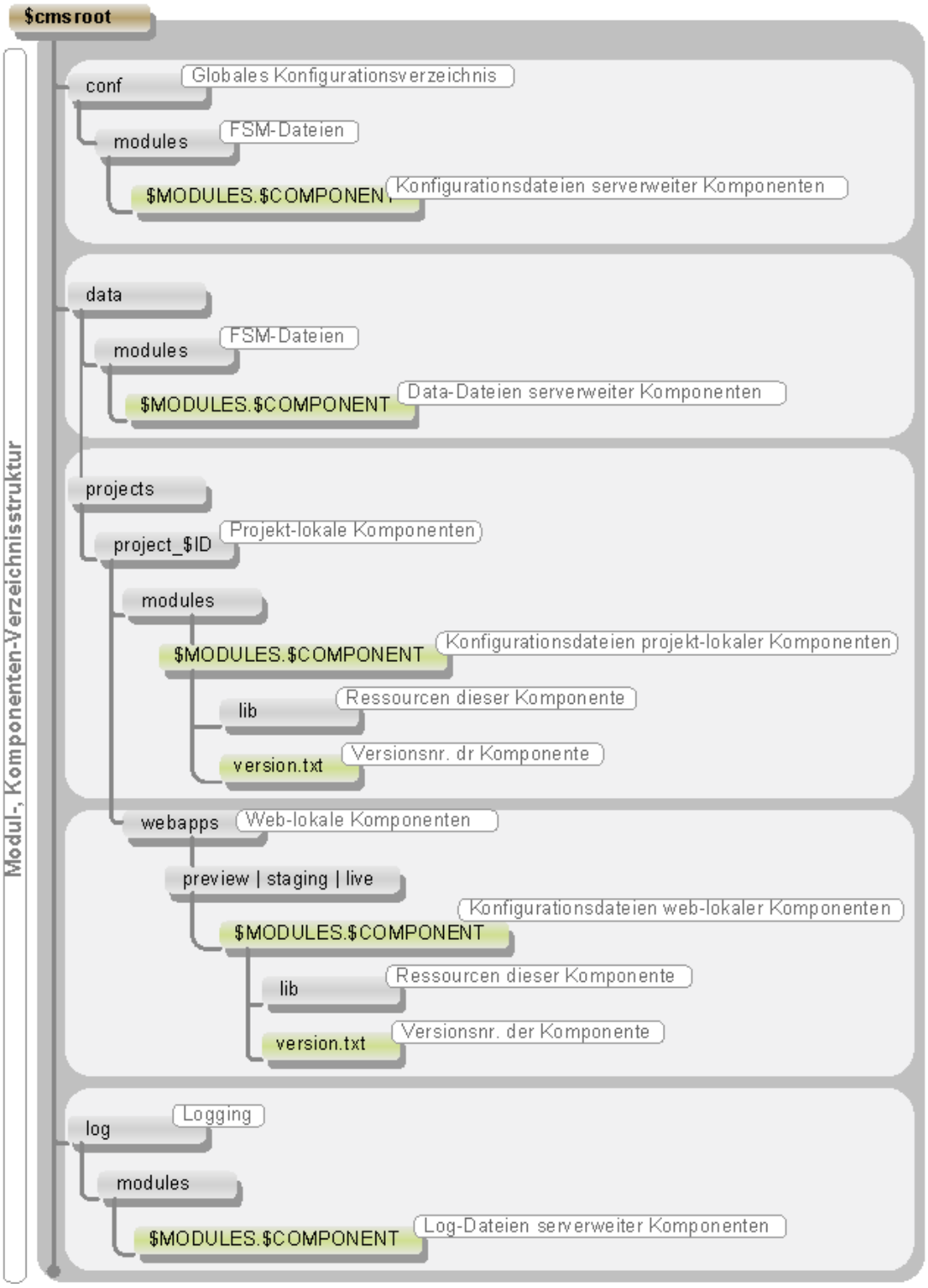


Abbildung 2-1: Module/Komponenten Verzeichnisstruktur

$MODULES.$COMPONENT steht für Modulname.Komponentenname

```
1.
2.      <module>
3.              <name>MyModule</name>
4.              <components>
5.                  <service>
6.                      <name>MyService</name>
7.                  </service>
8.          </component>
9.      </module>
10.
```

Aus Zeile 3 und Zeile 6 ergibt sich dann der Verzeichnisname: ***MyModule.MyService***

2.8 FSM-Archiv-Struktur

Abbildung 2-2: FSM-Archiv-Verzeichnisstruktur

2.9 Komponenten

Komponenten erweitern den `<components>`-Teil des Modul-Descriptors (siehe Kapitel 3.6 und 3.6.1 ab Seite 33).

```
1.    <components>
2.        <-- (siehe 3.6) -->
3.    </components>
```

2.9.1 Komponenten-Typen

2.9.1.1 Bibliothek

```
<library></library>
```

- **Bibliothek:** Eine Bibliothek ist die einfachste Form einer Komponente. Sie ist eine konfigurationslose Sammlung von Klassen, verpackt in einem oder mehreren Jar-Archiven. Bibliotheken sind immer server-weit, d.h. sie stehen nach der Installation auf dem Server, im Client, in Scripten und anderen Modulen ohne weitere Aktivierung zur Verfügung. Die Definition einer Library-Komponente im Modul-Descriptor kann über Ressourcen innerhalb des Library-Elements oder als Modul-Resource umgesetzt werden. Vorteil der zweiten Variante, es können Archive mit gleichnamigen Klassen in unterschiedlichen Versionen vorliegen und parallel zur Verwendung kommen siehe Kapitel 3.12 ‚Komponentenlose Modul-Implementierung (JDBC-Connector-Modul)' Seite 87.

- `de.espirit.firstspirit.module.Library`

2.9.1.2 Editoren

```
<editor></editor>
```

Editor: Eine Editor-Komponente stellt eine Eingabekomponente inklusive Datenmodell- und Render-Klasse zur Verfügung, über die es möglich ist, den Client um eigene Eingabemöglichkeiten zu erweitern. *CMS_INPUT_PERMISSION* ist ein Beispiel für einen eigenen Editor, der die Definition von Rechten erlaubt. Er arbeitet mit einem passenden Service zusammen, der die Gruppendefinitionen vom Server lädt und bereitstellt. Editoren implementieren das typisierte `Editor<ServerEnvironment>`-Interface. In der Regel werden FirstSpirit Eingabekomponenten entwickelt, wenn dem Redakteur entweder eine neue Funktionalität zur Verfügung gestellt oder eine projektspezifische Aufgabe vereinfacht werden soll.

- `de.espirit.firstspirit.module.Editor`
- `de.espirit.firstspirit.module.ServerEnvironment`
- `de.espirit.firstspirit.client.access.editor.swing.AbstractValueGuiEditor`

.

2.9.1.3 Projektanwendung

```
<project-app></project-app>
```

- **ProjektAnwendung:** Diese Komponenten sind projekt-lokal, d.h. sie müssen nach der Installation des Moduls den gewünschten Projekten hinzugefügt werden. Die selektierten Projekte werden mit bestimmten Eigenschaften/Fähigkeiten ausgestattet. Über eine Konfigurationsoberfläche kann dieser Komponenten-Typ für das jeweilige Projekt konfiguriert werden. Eine Projekt-Anwendung implementiert das Interface `ProjectApp`, eine Konfigurationsmaske für die Projekt-Anwendung implementiert `Configuration<ProjectEnvironment>`.

- `de.espirit.firstspirit.module.ProjectApp`
- `de.espirit.firstspirit.module.Configuration`
- `de.espirit.firstspirit.module.ProjectEnvironment`

2.9.1.4 Service

```
<service></service>
```

- **Service:** Ein Service ist eine Server-Komponente, ausgestattet mit einem öffentlichen Interface. Über den `ServiceLocator` der FirstSpirit Access-API [2] ist der Service serverweit verfügbar. Über die öffentliche Schnittstelle können Eingabekomponenten (Editoren) oder Scripte den Dienst ansprechen. Als Beispiel ist hier eine Virenscan-Modul-Implementierung zu nennen, die im Verlauf dieses Dokumentes näher erläutert wird (siehe Kapitel 3.14 Seite 93). Ein Service kann eine serverweite Konfigurationsmaske zur Verfügung stellen, die innerhalb der FirstSpirit Server- und Projektkonfiguration aufgerufen werden kann, implementiert werden muss `Configuration<ServerEnvironment>`.

- `de.espirit.firstspirit.access.ServiceLocator`
- `de.espirit.firstspirit.module.Service`
- `de.espirit.firstspirit.module.Configuration`
- `de.espirit.firstspirit.module.ServerEnvironment`
- `de.espirit.firstspirit.module.ServiceProxy`

2.9.1.5 Web-Anwendung

```
<web-app></web-app>
```

- **Web-Anwendung:** Definition von JSP-Tags und Servlets, die in den Projekten verwendet und aufgerufen werden können. Aus allen für ein Projekt definierten Web-Komponenten muss bei der Aktivierung (Deployment) eine einzige Web-Anwendung mit einer gemeinsamen web.xml entstehen, deren XML-Einträge einer fest vorgegebenen Reihenfolge genügen müssen. Hierzu werden die folgenden Elemente aus den einzelnen web.xml-Templates herausgelesen und in die gemeinsame web.xml kopiert:

  `<context-param>`,
  `<filter>`,
  `<filter-mapping>`,
  `<listener>`,
  `<servlet>`,
  `<servlet-mapping>`,
  `<mime-mapping>`,
  `<error-page>`,
  `<taglib>`.

  Alle übrigen Einträge werden ignoriert. Context-, Filter- und Servletnamen müssen eindeutig sein. Bei der Entwicklung muss dies berücksichtigt und mit passenden Prefixes gewährleistet werden. Gleiches gilt für Taglib-URIs. Konflikte bei Mime-Mappings und Fehlerseiten dagegen werden ignoriert und nach dem Prinzip first-come-first-serve aufgelöst. Konfigurationsdateien, die von der Komponente selbst geschrieben wurden, landen im WEB-INF-Verzeichnis der Web-Anwendung und stehen den Servlets so über `ServletContext.getResourceAsStream()` zur Verfügung. Stehen im web.xml-Template Platzhalter in der Form `${propertyKey}`, so werden diese mit den Werten aus der Konfiguration ersetzt.

- `de.espirit.firstspirit.module.WebApp`

2.9.1.6 Web-Server

```
<web-server></web-server>
```

- **Web-Server:** Eine Web-Server-Komponente steuert einen laufenden Web-Server. Beispiele hierfür sind die interne Web-Server-Steuerung oder eine

Tomcat-Unterstützung. Eine Web-Server-Komponente implementiert die Steuerung eines bestimmten Web-Servers, beispielsweise Tomcat oder Jetty), zunächst das Deploy/Undeploy von Web-Anwendungen. Die Konfiguration geschieht analog zu den Services. Eine Web-Server-Komponente kann einen eigenen Konfigurationsdialog haben, um Pfade, URLs, Passwörter u.ä. einzustellen. In den Server-Eigenschaften (Server- und Projektkonfiguration) lassen sich Web-Server hinzufügen und so dem FirstSpirit-Server bekannt machen. In den Projekt-Eigenschaften wird den Web-Bereichen (Preview, Staging, Live) einer der zuvor hinzugefügten Web-Server zugeordnet.

Im System vorhandene Web-Server:

- **Intern:** Steuerung des internen Jetty.
- **Generic:** Deploy/Undeploy über Beanshell-Scripte, die mit einer einfachen Key-Value-Map konfiguriert werden können. Die Scripte liegen im Konfigurationsverzeichnis (Kapitel 2.7 Seite 12) und lassen sich über den Konfigurations-Dialog editieren.
- **Extern:** Leere Dummy-Implementierung; unterstützt kein Deploy oder Undeploy.

Anwendungsbeispiel:

- Die Unterstützung für WebSphere wird als Modul (fs-ibmws.fsm) umgesetzt.
- Installation des Moduls.
- Server-Eigenschaften: WebSphere als "WS-1" hinzufügen, URLs und Passwörter konfigurieren.
- Projekt-Eigenschaften: Staging, Umschalten von "Intern" auf "WS-1".

- `de.espirit.firstspirit.module.WebServer`
- `de.espirit.firstspirit.access.schedule.WebServerConfiguration`
- `de.espirit.firstspirit.access.schedule.DeployTarget`

2.9.1.7 Public

```
<public></public>
```

Neben komplexeren Komponenten wie Editoren oder Web-Anwendungen existieren im FirstSpirit-System auch Schnittstellen/HotSpots, für die kein spezialisierter Komponenten-Typ definiert wurde. Über die Public-Deklaration des Moduls werden Klassen dem FirstSpirit-Server bekannt gemacht, die solche Schnittstellen implementieren. Diese Klassen müssen über Modul-Bibliotheken (Library-

Komponente, siehe Kapitel 2.9.1.1 und 2.5.1 Seiten 16 bzw. 10) oder Modul-Ressourcen analog zu einer komponentenlosen Library (siehe Kapitel 3.12 ‚Komponentenlose Modul-Implementierung (JDBC-Connector-Modul)' Seite 87 und Absatz 2.9.1.1 ‚Bibliothek', Seite 16) gefunden werden.

- `Bei diesem Typ ist keine Implementierung (implements) eines speziellen Interfaces notwendig.`

2.9.2 Sichtbarkeit (Scope) von Komponenten

```
<scope>server | project | web</scope>
```

Jede Komponente ist innerhalb eines definierten Scopes im FirstSpirit-Environment sichtbar. Die Zuweisung folgender Scopes ist hierbei möglich. Dabei sind einige Komponenten generell serverweit sichtbar, hierzu gehören folgende Komponenten: Editor, Web-Server, Library (Bibliothek) und Service (siehe auch Kapitel 3.7.1 Seite 36).

Server: Serverweite Komponenten stehen nach ihrer Installation auf dem kompletten Server zur Verfügung – damit auch im JavaClient, in allen Scripten sowie in allen anderen Komponenten.

Project: Projekt-lokale Komponenten werden manuell an einzelne Projekte gebunden. Konfigurations- und Laufzeitdateien werden bei diesem Komponenten-Typ im Projektverzeichnis abgelegt.

Web: Web-lokale Komponenten (zu denen per Definition die Web-Anwendungen gehören) werden manuell den einzelnen Web-Bereichen Preview, Staging und Live bestimmter Projekte zugefügt. Das ermöglicht nicht nur verschiedene Konfigurationen, sondern es können auch Komponenten nur für bestimmte Bereiche aktiviert werden.

2.10 Classloading

2.10.1 Hierarchie

Um Konflikte zwischen verschiedenen Modulen und dessen Komponenten zu vermeiden, werden die Klassen – soweit wie möglich – über einzelne, voneinander getrennte Classloader geladen. Hierbei gilt die folgende Classloader-Hierarchie:

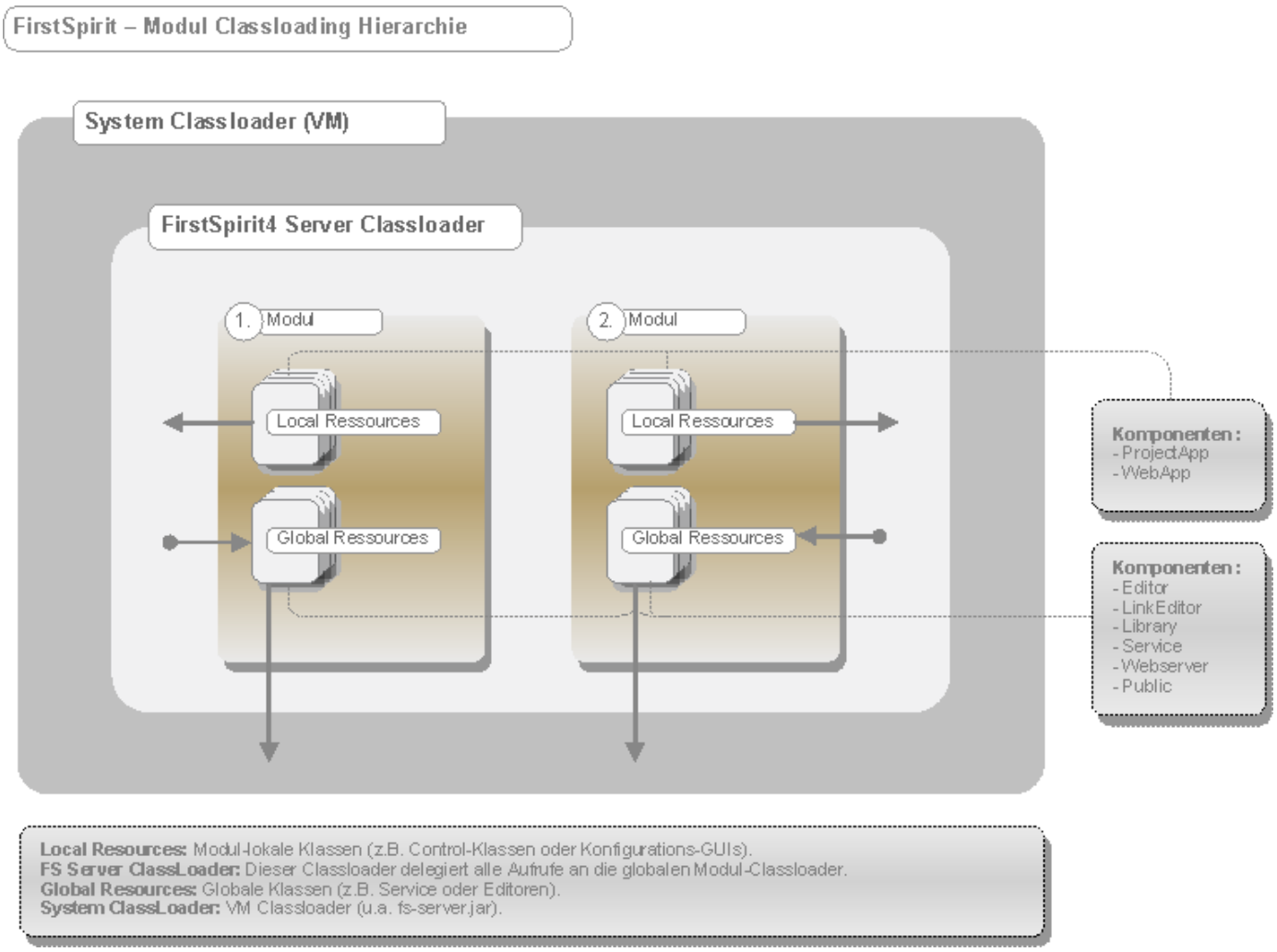


Abbildung 2-3: Modul-Classloading-Hierarchie

Die nachfolgende Grafik zeigt die Modul-Ressourcen-Aufteilung in die einzelnen Classloader. Das Beispiel-Modul in der folgenden Grafik enthält eine Projekt- und eine Web-Komponente (lokal), einen Editor (global) sowie eine Library und einen Service (global). Außerdem sind noch Modul-Ressourcen für die Modul-Steuerungs-Klasse definiert.

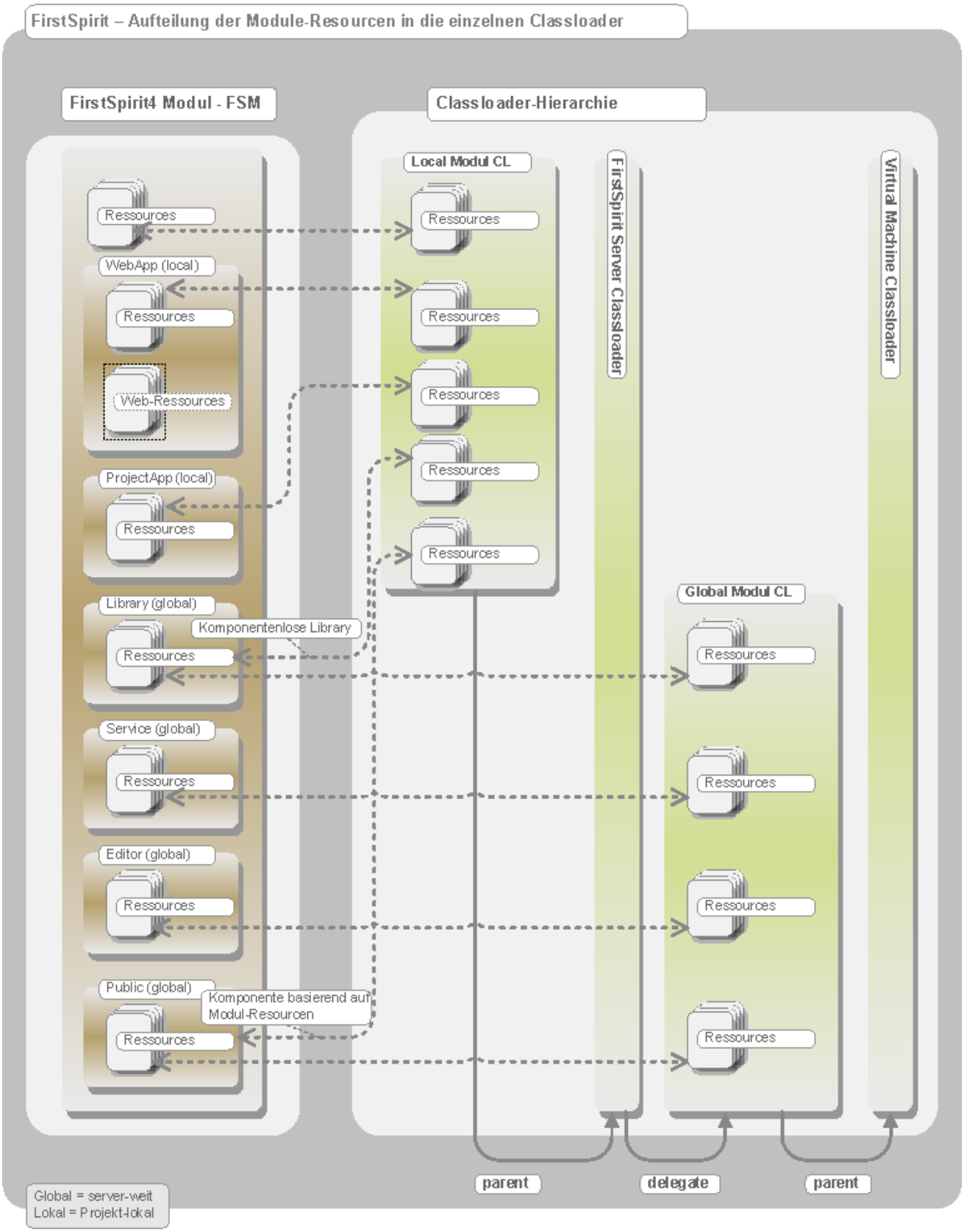


Abbildung 2-4: Modul-Classloading-Hierarchie – Classloader-Aufteilung

Alle lokalen Komponenten werden zusammen mit den Modul-Ressourcen im Local Module-Classloader zusammengefasst.

Die serverweiten Ressourcen der Library und des Services gelangen in den Global Module-Classloader. Dieser ist nicht direkt, sondern nur über den FirstSpirit-Server-Classloader erreichbar. Dieser delegiert alle `findClass()`- und `findResource()`-Aufrufe an die Module weiter. Die Classloading-Hierarchie ermöglicht, dass die lokalen Klassen (wie die Konfigurations-GUI) globale Klassen (beispielsweise einen Service) verwenden können.

2.10.2 Classloading in WebAnwendungen am Beispiel FirstSpirit u. Websphere

Anwendungsfall:

Eine Webanwendung, die in Websphere 6.1 eine FirstSpirit-Connection hält, und über diese Verbindung einen Modul-Service mittels `getService(„ServiceName“)` remote vom FirstSpirit-Server lädt, wird unter Umständen eine `ClassCastException` werfen, da die Klasse des Services, die vom FirstSpirit-Server zurückgeliefert wird, von zwei verschiedenen ClassLoadern geladen wird. Die Klasse des Service, die zurückgeliefert wird, ist vom FirstSpirit-RemoteClassLoader geladen worden und kann nicht auf die Klasse gekastet werden, die der Anwendung durch den Websphere-ClassLoader (CompoundClassloader) bekannt ist.

Die ClassLoader-Hierarchie stellt sich in etwa wie folgt dar:

```
WebApp(ComboundClassloader) ⟶  VM-Classloader
```

Im VM-Classloader liegt das `fs-server.jar`, unter `WEB-INF` liegt der Service und somit im WebApp-Classloader, dieser delegiert an VM-Classloader. Da die FirstSpirit-Connection (VM-Classloader) versucht, die Service-Klasse zu laden und nichts findet, lädt sie diese über den FirstSpirit-RemoteClassloader. Dieser wird in der Hierarchie zwischen WebApp und VM nicht erreicht und daher wird die Service-Klasse von zwei Classloadern geladen.

Lösung:

Das JAR der Modul-Service-Komponente und das FirstSpirit fs-server.jar müssen in diesem Fall an gleicher Stelle liegen, also entweder im VM-Classloader oder im WebApp-Classloader (WEB-INF/lib).

2.11 Komponenten-Konfigurationsdateien

Hier muss zwischen globalen (serverweiten) Komponenten und lokalen (projekt-lokalen) Komponenten unterschieden werden. Konfigurationsdateien globaler Modul-Komponenten befinden sich im Verzeichnis `conf/module/$MODULE.$COMPONENT`, Konfigurationsdateien lokaler Komponenten sind im Verzeichnis `data/projects/project_$ID/conf` zu finden bzw. sollten hier abgelegt werden.

```
FileHandle handle = _environment.getConfDir().obtain(„my-
module.conf“);
```

Das ServerEnvironment **_environment** wird beim Aufruf bzw. bei der Initialisierung der Komponente durch die FirstSpirit Modul-Umgebung über die `init()`-Methode des jeweiligen Komponenten-Interfaces übergeben. Das ServerEnvironment (**de.espirit.firstspirit.module.ServerEnvironment**) stellt nützliche Informationen bereit, wie z.B. das Modul-Konfigurations-Verzeichnis oder das Modul-Verzeichnis für Logdateien. Die Methode `getConfDir()` liefert das Konfigurationsdateien-Verzeichnis für das jeweilige Environment. Die Komponente sollte alle Konfigurationsdateien in diesem Verzeichnis ablegen.

Beispiel `init()`-Methode einer Service-Komponente:

```
1.    public void init(final ServiceDescriptor descriptor, final
 ServerEnvironment env) {
2.        _environment = env;
3.    }
```

Für eine Editoren-Komponente stellt sich diese wie folgt dar:

```
1.    public void init(final EditorDescriptor descriptor, final
 ServerEnvironment env) {
2.        _environment = env;
3.    }
```

2.12 Logging in FirstSpirit

2.12.1 Globales Logging – fs-server.log / fs-client.log

```
4.    import de.espirit.common.base.Logging;
5.
6.    private static final Class<?> LOGGER = MyClassName.class;
7.
1.    Logging.logInfo("…the string to log", LOGGER);
2.    Logging.logWarn("…the string to log", LOGGER);
3.    Logging.logDebug("…the string to log", LOGGER);
4.    Logging.logTrace("…the string to log", LOGGER);
5.    Logging.logError("…the string to log", LOGGER);
```

Listing 3: Globales Logging

2.12.2 Lokales Logging auf Modulebene

Um auf Modulebene eine Logdatei in `log/module/$MODULE.$COMPONENT` zu schreiben (ein einfaches Anwendungsbeispiel für eine Logdatei wäre die Installation oder Aktualisierung), siehe auch Kapitel 2.7 Seite 12, kann über das ServerEnvironment (siehe Kapitel 2.11 Seite 24) das hierfür in der globalen Verzeichnisstruktur vorgesehene Logverzeichnis bezogen werden. Die Methode `getLogDir()` liefert das Logdateien-Verzeichnis für das jeweilige Environment. Die Komponente sollte alle Logdateien in diesem Verzeichnis speichern. Für diese Art des Logging's wird eine modulspezifische Implementierung benötig z.B. Log4J oder SLF4J.

```
FileHandle handle = _environment.getLogDir().obtain(„my-
module.log“);
```

Listing 4: Logging auf Modulebene

3 Das FirstSpirit Modul- / Komponenten-Modell

Alle Komponenten-Typen in FirstSpirit lassen sich untereinander kombinieren, so dass ein komponenten-basierter Informationsfluss möglich ist. Unter bestimmten Anforderungen ist diese Kombination auch zwingend erforderlich.

Soll beispielsweise eine Editor-Komponente auf einen entfernten Web-Service zugreifen, wird dieses durch die clientseitige FirstSpirit-Security-Policy verhindert. Die Implementierung eines solchen Zugriffs muss über einen serverseitigen Service umgesetzt werden, d.h. die Eingabekomponente (Editor-Komponente) kommuniziert mit dem Service über das FirstSpirit-Protokoll bzw. über das ServerEnvironment besteht die Möglichkeit, Verbindungsinformationen zu beziehen und so über die aktuelle Verbindung (`getConnection()`) mit dem spezialisierten Service zu kommunizieren. Der Service stellt die benötigte Verbindung mit dem entfernten Web-Service her und delegiert alle Daten weiter an den Web-Service bzw. zurück an die Editoren-Komponente.

```
// a) Get the service implementation by class
Service =
_environment.getConnection().getService(MyService.class);


// b) Get the service implementation by descriptor name
MyService service = (MyService)
_environment.getConnection().getService("MyService");
```

Listing 5: ServerEnvironment – Service

a) Die erste Variante nutzt hier die konkrete Klasse und sollte aus Gründen der Typensicherheit bevorzugt genutzt werden.
b) Der String „MyService“ entspricht hier dem definierten Namen im Modul-Descriptor.
   `<name>MyService</name>`

`_environment.getConnection().` – bietet außerdem grundlegende Methoden zur Beschaffung von serverseitigen Daten, beispielsweise

```
_environment.getConnection().getProjects()
_environment.getConnection().getProjectById(long id)
_environment.getConnection().getProjectByName(String name)
```

Listing 6, zu finden in der FirstSpirit-Access-API

3.1 Komponenten-Container (Typen)

Wie schon zuvor in Kapitel 2.9.1 (ab Seite 16) erwähnt, gibt es sechs Komponenten-Typen oder Komponenten-Container.

- Eine **Bibliothek** implementiert das typisierte `Library<ServerEnvironment>`-Interface;
- eine **Editor**-Komponente implementiert das typisierte `Editor<ServerEnvironment>`-Interface;
- eine **Projektanwendung** implementiert das Interface `ProjectApp`, die optional eine Konfigurationsoberfläche der Projektanwendung implementiert, ggf. das `Configuration<ProjectEnvironment>`-Interface;
- ein **Service** implementiert das `Service<T>`-Interface (implementierende Klassen müssen einen no-arg-Konstruktor besitzen), die eine mögliche Service-Konfigurationsmaske implementiert `Configuration<ServerEnvironment>`;
- eine **Web-Anwendung** erweitert `AbstractWebApp` und implementiert das Interface `WebServer;`
- eine **Web-Server**-Komponente implementiert das Interface `Webserver`.

```
de.espirit.firstspirit.module.Library
de.espirit.firstspirit.module.Editor
de.espirit.firstspirit.module.ServerEnvironment
de.espirit.firstspirit.module.ProjectApp
de.espirit.firstspirit.module.Configuration
de.espirit.firstspirit.module.WebEnvironment
de.espirit.firstspirit.module.ProjectEnvironment
de.espirit.firstspirit.module.ServerEnvironment
de.espirit.firstspirit.module.Service
de.espirit.firstspirit.access.ServiceLocator
de.espirit.firstspirit.module.ServiceProxy
de.espirit.firstspirit.module.WebApp
```

```
de.espirit.firstspirit.module.WebServer
```

3.2 Initialisierung von Komponenten

Die Komponenten werden vom FirstSpirit-System automatisch initialisiert. Der Zeitpunkt der Initialisierung einer Komponente ist abhängig vom Komponenten-Typ.

- **Editoren** werden „on demand“ geladen, d.h. bei der ersten Benutzung der jeweiligen Eingabekomponente im JavaClient
- **Projektanwendungen** werden bei Projektinitialisierung geladen
- **Bibliotheken**, **Services** (wenn Autostart aktiv), **Webanwendungen**, **Web-Server**-Komponenten werden bei Serverstart geladen.

Eine Komponente und ggf. vorhandene Konfigurationsoberflächen werden durch den Aufruf folgender Methoden initialisiert (Die Methoden müssen durch jede Komponenten implementiert werden):

1. Aufruf des parameterlosen Konstruktors der Komponente
2. Die `init`[1]-Methode wird aufgerufen.

Sind mehrere Komponenten in einem Modul-Descriptor (Kapitel 3.6 Seite 33) definiert, werden alle Komponenten durch den Aufruf ihrer jeweiligen `init`-Methoden initialisiert.

3.3 Entladen von Komponenten

Die Cache-Implementierung in diesem Bereich ist noch nicht vollständig abgeschlossen. Die Dokumentation erfolgt sobald wie möglich.

[1] de.espirit.firstspirit.module.Component#init(D extends ComponentDescriptor, E extends ServerEnvironment)

3.4 Event-Methoden von Komponenten

Jede Komponente, die ihr spezifisches Interface implementiert, besitzt die folgenden Event-Methoden:

```
1.     package de.espirit.firstspirit.opt.example.editor.simple;
2.
3.     import de.espirit.firstspirit.access.editor.EditorValue;
4.     import
 de.espirit.firstspirit.access.store.templatestore.gom.GomElement;
5.     import
 de.espirit.firstspirit.access.store.templatestore.gom.GomModule.Kind;
6.     import de.espirit.firstspirit.client.access.editor.swing.GuiEditor;
7.     import de.espirit.firstspirit.module.Editor;
8.     import de.espirit.firstspirit.module.ServerEnvironment;
9.     import de.espirit.firstspirit.module.descriptor.EditorDescriptor;
10.    import de.espirit.common.base.Logging;
11.
12.    import java.util.EnumSet;
13.
14.
15.    /**
16.     * FirstSpirit editor compoment.
17.     */
18.    public class FSEditorContentComponent implements
 Editor<ServerEnvironment> {
19.
20.        private static final Class<FSEditorContentComponent> LOGGER =
 FSEditorContentComponent.class;
21.
22.        //--- Editor ---//
23.    ----------------------------------------------------------------
24.
25.
26.
27.        /**
28.         * Returns editor's {@link
 de.espirit.firstspirit.access.editor.EditorValue} class.
29.         *
30.         * @return editor's {@link
 de.espirit.firstspirit.access.editor.EditorValue} class.
31.         */
```

```
32.         public Class<? extends EditorValue> getEditorValueClass() {
33.             return FSEditorContentValueImpl.class;
34.         }
35.     ================================================================
36.
37.
38.
39.         /**
40.          * Returns the gui editor class of the specified type or
 <code>null</code> if the type is not supported.
41.          *
42.          * @since 4.1
43.          */
44.           public <T> Class<? extends T> getGuiEditorClass(@NotNull final
 Class<T> type) {
45.                 if (type == GuiEditor.class) {
46.                       return (Class<? extends T>)
 FSEditorContentGuiEditor.class;
47.                 }
48.                 return null;
49.           }
50.     ================================================================
51.
52.
53.
54.         /**
55.          * Returns editor's {@link
 de.espirit.firstspirit.access.store.templatestore.gom.GomElement} class.
56.          *
57.          * @return editor's {@link
 de.espirit.firstspirit.access.store.templatestore.gom.GomElement} class.
58.          */
59.         public Class<? extends GomElement> getGomElementClass() {
60.             return GomFSEditorContent.class;
61.         }
62.     ================================================================
63.
64.
65.
66.         /**
67.          * Returns editor's {@link
 de.espirit.firstspirit.access.store.templatestore.gom.GomElement}, {@link
68.          *
```

```
 de.espirit.firstspirit.access.store.templatestore.gom.GomModule.Kind}
 restrictions.
69.          *
70.          * @return editor's {@link
 de.espirit.firstspirit.access.store.templatestore.gom.GomElement}, {@link
71.          *
 de.espirit.firstspirit.access.store.templatestore.gom.GomModule.Kind}
 restrictions.
72.          */
73.         public EnumSet<Kind> getGomElementRestrictions() {
74.             return EnumSet.allOf(Kind.class);
75.         }
76.
77.
78.         //--- Component ---//
79.     ----------------------------------------------------------------------
80.
81.
82.
83.         /**
84.          * Initializes this component with the given {@link
 de.espirit.firstspirit.module.descriptor.ComponentDescriptor
85.          * descriptor} and {@link
 de.espirit.firstspirit.module.ServerEnvironment environment}. No other
 method will be called
86.          * before the component is initialized!
87.          *
88.          * @param descriptor useful descriptor information for this
 component.
89.          * @param env       useful environment information for this
 component.
90.          */
91.         public void init(final EditorDescriptor descriptor, final
 ServerEnvironment env) {
92.             // do something
93.             Logging.logDebug("init()", LOGGER);
94.         }
95.     ----------------------------------------------------------------------
96.
97.
98.
99.         /**
100.          * Event method: called if Component was successfully installed
```

```
 (not updated!)
101.         */
102.        public void installed() {
103.            // do something
104.            Logging.logDebug("installed()", LOGGER);
105.        }
106.    ================================================================
107.
108.
109.
110.        /**
111.         * Event method: called if Component was in uninstalling procedure
112.         */
113.        public void uninstalling() {
114.            // do something
115.            Logging.logDebug("uninstalling()", LOGGER);
116.        }
117.    ================================================================
118.
119.
120.
121.        /**
122.         * Event method: called if Component was completely updated
123.         *
124.         * @param oldVersionString old version, before component was
 updated
125.         */
126.        public void updated(final String oldVersionString) {
127.            // do something
128.            Logging.logDebug("updated(" + oldVersionString + ")", LOGGER);
129.        }
130.    }
```

Listing 7: Komponenten Event-Methoden – installed, updated, uninstalled

3.5 Modul Datei-, Archiv-Format (.fsm)

Alle Bestandteile eines Moduls werden in einem FSM(Zip)-Archiv mit dem Suffix `.fsm` zusammengefasst (siehe Kapitel 3.16 Seite 125). Signierte Module werden bei der Installation überprüft und gegebenenfalls zurückgewiesen (siehe Kapitel 3.17 Seite 125).

3.6 Der Modul-Descriptor

Das FSM-Archiv muss einen Modul-Descriptor enthalten (/META-INF/module.xml), der Auskunft über das Modul und die darin gekapselten Komponenten gibt.

3.6.1 Beispiel Modul-Descriptor

```
1.  <!DOCTYPE module SYSTEM "http://www.FirstSpirit.de/module.dtd">
2.  <module>
3.    <name>ExampleModule</name>
4.    <version>1.0</version>
5.    <description>This module is just an example.</description>
6.    <license>neccessaryLicenceFeature: e.g. a validKey</license>
7.    <vendor>e-Spirit AG</vendor>
8.    <class>de.espirit.exmod.ModuleImpl</class>
9.    <resources>
10.     <resource>libs/exmod.jar</resource>
11.   </resources>
12.   <components>
13.     <!-- (‚Komponenten-Descriptor' siehe unter 0 2.9.1 u. 03.7) -->
14.   </components>
15.   <dependencies>
16.     <depends>AnotherModule</depends>
17.   </dependencies>
18. </module>
```

Listing 8: Modul-Descriptor-Beispiel

3.6.2 Modul-Descriptor Tags und Attribute

3.6.2.1 Pflichtfelder

Folgende Felder sind für jeden Modul-Descriptor zwingend erforderlich:

```
1.  <module>
2.      <name></name>
3.      <version></version>
4.      <components></components>
5.  </module>
```

<module> Mandatory	Der einleitende Modul-Descriptor-Container
<name> Mandatory	Name des Moduls. Erlaubte Zeichen: A-Za-z0-9; ,_\-
<version> Mandatory	Modulversionierung **Punkt-Notation** z.B. 0.1 oder 0.0.1 oder 0.0.0.1 + optionale Revision z.B. _72
<components> Mandatory	Hier werden die im Modul enthaltenen Komponenten definiert (siehe auch Kapitel 3.7 Seite 36)

Listing 9: Pflichtfelder des Modul-Descriptors

3.6.2.2 Optionale Einträge

```
1.    <description></description>
2.    <license></license>
3.    <vendor></vendor>
4.    <class></class>
5.    <resources>
6.        <resource></resource>
7.    </resources>
```

<description>	Aussagekräftige Beschreibung des Moduls und dessen Funktionalität
<license>	Bspw. ein Lizenzschlüssel; Überprüfungslogik muss vom Modul bzw. von einer der Modul-Komponenten implementiert werden.
<vendor>	Eine Herstellerangabe, z.B. e-Spirit AG
<class>	Auf bestimmte Ereignisse wie Installation, Deinstallation und Aktualisierung durch Benutzer in der Administrationsoberfläche kann durch eine spezialisierte Modul-Klasse reagiert werden, um bspw. Aktionen an allen Komponenten eines Moduls auszuführen.
<resources>	Container für 0..* Ressourcen-Einträge
<resource>	Der Ressourcen-Eintrag
<dependencies>	Abhängigkeiten-Container, kann 0..* abhängige Moduleinträge enthalten
<depends>	Besitzt das Modul Abhängigkeiten zu anderen Modulen, müssen die Modulnamen an dieser Stelle als <depends>-Einträge definiert werden.

Listing 10: Optionale Modul-Descriptor-Elemente

3.7 Der Komponenten-<*components*>-Descriptor-Teil

Komponenten erweitern den <*components*>-Teil des Modul-Descriptors (siehe Kapitel 3.6.1 Seite 33). Das <components>-Element ist zwingend erforderlich auch wenn es sich um eine Komponenten loses Modul handelt (siehe Kapitel 3.12 ‚Komponentenlose Modul-Implementierung (JDBC-Connector-Modul)', Seite 87).

```
1.    ...
2.    <components>
3.    <!-- <library> <editor> <project-app> <web-app> <web-server> -->→→→
4.    </components>
5.    ...
```

3.7.1 Komponenten-Descriptoren und spezielle Eigenschaften

Der Komponenten-Descriptor ist ein „Teil"-Descriptor des Modul-Descriptors.

3.7.1.1 Bibliothek

Eine **Bibliothek** ist die einfachste Form einer Komponente. Sie ist eine konfigurationslose Sammlung von Klassen, verpackt in einem oder mehreren Jar-Archiven. Serverweite Bibliotheken stehen nach der Installation auf dem Server, im Client, in Scripten und anderen Modulen ohne weitere Aktivierung zur Verfügung. Projekt- und weblokale Bibliotheken müssen erst Projekten hinzugefügt werden.

```
1.    <library>
2.      <name>MyLibs</name>
3.      <description>Example library.</description>
4.      <hidden>true</hidden>
5.      <scope>server</scope>
6.      <resources>
7.        <!-- shared resources -->
8.        <resource name="log4j" version="1.2.7" minVersion="1.2"
 maxVersion="1.2.999">libs/log4j.jar</resource>
9.        <resource name="JAMon" version="1.0">libs/JAMon.jar</resource>
10.       <resource name="poi" version="2.5">libs/poi.jar</resource>
11.     </resources>
12.   </library>
```

Eigenschaften:	
<hidden>	Optional. Definiert ob eine Komponente „in der Modul-Konfiguration der Server- und Projekteigenschaften“ zu sehen ist. **false | true** [true]
<scope>	Wo stehen die Ressourcen zur Verfügung? (**server**, **project** oder **web**) [server]
<name> Mandatory	Der Name der Komponente ist in einer Bibliothek zwingend erforderlich, dies gilt für alle Komponenten.
<resources> Mandatory	Das <*resources*>-Element muss definiert sein, auch wenn dieses keine weiteren Ressourcen-Einträge enthält.

Listing 11: Library Komponenten-Descriptor und Eigenschaften

3.7.1.2 Editor

Eine **Editor**-Komponente stellt eine Eingabekomponente inklusive Datenmodell- und Render-Klasse (`<class> Attribute`) zur Verfügung, über die es möglich ist, den Client um eigene Eingabemöglichkeiten zu erweitern.

```
1.    <editor>
2.        <name>CMS_INPUT_SIMPLE_EDITOR_CONTENT</name>
3.        <description>FirstSpirit Simple Example Editor</description>
4.        <class>
 de.espirit.firstspirit.opt.example.editor.simple.FSEditorContentComponent</
 class>
5.        <resources>
6.            <resource version="0.0.1" minVersion="0.0"
 maxVersion="0.0.1">lib/fs-seditor-example.jar</resource>
7.        </resources>
8.    </editor>
```

Eigenschaften:	
<scope>	Editoren sind immer systemweit → **server**
<name> Mandatory	Definiert den Namen, über den die Komponente in FirstSpirit als Eingabekomponente eingebunden wird.
<class> Mandatory	Klasse, die das Interface Editor implementiert. *FSEditorContentComponent implements Editor*[2]*<ServerEnvironment>*
<resource> Mandatory	Die Editor-eigenen Ressourcen. **version** Aktuelle Version (optional)

[2] de.espirit.firstspirit.module.Editor<E extends ServerEnvironment> extends Component<EditorDescriptor, E>

	minVersion	Minimal kompatible Version (optional)
	maxVersion	Maximal kompatible Version (optional)

Listing 12: Editor Komponenten-Descriptor und Eigenschaften

3.7.1.3 Service

Ein **Service** ist eine Server-Komponente, ausgestattet mit einem öffentlichen Interface. Über den `de.espirit.firstspirit.access.ServiceLocator` der FirstSpirit Access-API [2] ist der Service serverweit verfügbar. Über die öffentliche Schnittstelle können z.B. Eingabekomponenten (Editoren) oder Skripte den Dienst ansprechen. Als Beispiel ist hier die Virenscan-Modul-Implementierung zu nennen, welchen im Verlauf dieses Dokumentes näher erläutert wird (siehe Kapitel 3.14).

```
1.    <service>
2.        <name>VScanService</name>
3.        <description>FirstSpirit Virus Scan Service</description>
4.        <class>de.espirit.firstspirit.opt.vscan.VScanServiceImpl</class>
5.    <configurable>de.espirit.firstspirit.opt.vscan.admin.gui.VScanServiceC
 onfigPanel</configurable>
6.        <resources>
7.            <resource name="libvscan">lib/fs-vscan.jar</resource>
8.            <resource>fs-vscan.conf</resource>
9.        </resources>
10.   </service>
```

Eigenschaften:	
<scope>	Services sind immer systemweit → **server**
<name> Mandatory	Definiert den Namen, über den die Komponente in FirstSpirit als Service-Komponente erreichbar ist.
<class> Mandatory	Klasse, die das typisierte Interface `de.espirit.firstspirit.module.Service<T>` implementiert. konkret: *VScanServiceImpl implements VScanService, de.espirit.firstspirit.module.Service<VScanService>*
<configurable>	Definition der Konfigurationsoberflächen-Klasse. Ist dieses Element nicht definiert, stellt der Service in der FirstSpirit Server- und Projektkonfiguration kein Konfiguration-GUI zur Verfügung – der „Konfigurieren“-Button in der FirstSpirit Server- und Projektkonfiguration ist deaktiviert. konkret: *VScanServiceConfigPanel implements de.espirit.firstspirit.module.Configuration<ServerEnvironment>*

<resource> Mandatory	Definition der Service-eigenen Ressourcen. Jeder Service muss zwingend seine “eigenen” Klassen als Ressourcen in Form eines Jars enthalten. ▪ lib/fs-vscan.jar ▪ fs-vscan.conf (Service-Properties-Datei wird bei der Installation des Moduls in das vorgesehene Verzeichnis kopiert, siehe Kapitel 2.7 und 2.8 ab Seite 12) **version** Aktuelle Version (optional) **minVersion** Minimal kompatible Version (optional) **maxVersion** Maximal kompatible Version (optional)

Listing 13: Service Komponenten-Descriptor und Eigenschaften

Eine Beispiel-Service-Implementierung befindet sich in dem Moduldeveloper Archiv-Zip.

Jeder Service ist out-of-the-box multithreaded, d.h. ein ExecutorService oder andere nebenläufige Implementierungen innerhalb der Service-Implementierung sind nicht nötig. Jeder Remote-Request an den Service wird indirekt über den FirstSpirit-NIOSocketServer angesprochen, somit ist die Nutzung/Anfrage des Services z.B. aus einer Editoren-Komponente oder wie in dem Beispiel der Virenscanner-Modul-Implementierung automatisch Multi-Threaded. Alle Anfragen/Threads werden vom FirstSpirit-NIOSocketServer verwaltet.

3.7.1.4 Web-Server

Die Web-Server-Komponente dient zurzeit nur zum Deploy/Undeploy von Web-Anwendungen.

3.7.1.5 Public

Eine **Public**-Komponente (Schnittstelle/HotSpot) ist eine spezialisierte Klasse, die eine Schnittstelle der FirstSpirit Acess-API implementiert, für das kein spezialisierter Komponenten-Typ definiert wurde. Über die Public-Anweisungen des Moduls werden Klassen dem FirstSpirit-Server bekannt gemacht, die solche Schnittstellen implementieren. Diese Klassen müssen über Modul-Bibliotheken (siehe Kapitel 2.9.1.1 und 2.5.1 Seite 16 bzw. 10) gefunden werden. Die Definition der Bibliothek kann auch innerhalb des gleichen Moduls erfolgen, welches die Public-Komponente definiert.

```
1.     <public>
2.       <name>Beanshell</name>
3.       <description>Beanshell Scripting engine.</description>
4.
 <class>de.espirit.firstspirit.server.script.BeanshellScriptEngine</class>
5.     </public>
```

Eigenschaften:	
<scope>	Public-Komponenten sind immer systemweit → **server**
<name> Mandatory	Definiert den Namen, über den die Komponente in FirstSpirit als Public-Komponente erreichbar ist.
<class> Mandatory	Klasse, die ein Interface der FirstSpirit Access-API implementiert.

Listing 14: Public Komponenten-Descriptor und Eigenschaften

Wobei hier die Klasse `BeanshellScriptEngine` das Interface `ScriptEngine` implementiert. Eine mögliche Kombination einer Service- und einer Public-Komponente definiert sich im Modul-Descriptor wie in Kapitel 3.14.4 (Seite 96) zu sehen.

Public-Komponenten benötigen zwingend einen parameterlosen Konstruktor.

3.7.1.5.1 Public-Classes (Schnittstellen für eine Implementierung)

- **Interface *UploadFilter.class*** – für eine UploadFilter-Implementierung als PUBLIC-Komponente, (siehe auch Modul-Implementierung mit den Komponenten-Typen – PUBLIC, SERVICE, LIBRARY, Seite 93), steht berreits eine Abstrakte Umsetzung zu Verfügung (***FileBasedUploadFilter.class***), welche auch in dem zuvor referenzierten Beispiel im Detail beschrieben wird.

- **Interface *GomIncludeValueProvider.class*** – HotSpot-Implementierung für das FirstSpirit Gui-Object-Model (GOM-Form), eine konkrete Umsetzung kann beliebige Mengenwertigelisten aufnehmen aus welche der Benutzer dann in z.B. eine *CMS_INPUT_COMBOBOX*-Komponente selektieren kann. Eine solche Mengenwertigeliste kann über die GOM-Form-XML-Defintion in die Komponente eingebunden werden.

```
1.    <CMS_INCLUDE_OPTIONS type="public">
2.        <NAME>VALUE</NAME>
3.        <PARAMS>
4.            <PARAM name="Classname" />
5.        </PARAMS>
6.    </CMS_INCLUDE_OPTIONS>
```

Konkrete Beispiele finden sich innerhalb der FirstSpirit Modul-Developer-API (dev-api) welche sich in dem Archiv der Beispiel-Modul-Implementierungen befindet.

3.7.1.6 Projekt-Anwendung

Eine **Projekt-Anwendung** ist projekt-lokal, d.h. sie muss nach der Installation des Moduls den gewünschten Projekten hinzugefügt werden. Die selektierten Projekte werden mit bestimmten Eigenschaften/Fähigkeiten ausgestattet. Über eine Konfigurationsoberfläche kann dieser Komponenten-Typ für das jeweilige Projekt konfiguriert werden. Eine Projekt-Anwendung implementiert das Interface `ProjectApp`, eine Konfigurationsmaske für die Projekt-Anwendung implementiert `Configuration<ProjectEnvironment>`.

```
1.     <project-app>
2.         <name>MyProjectConfiguration</name>
3.         <description>My Project Configuration to do some.</description>
4.         <class>
 de.espirit.firstspirit.opt.myprojectapp.MyProjectApp</class>
5.        <configurable>
 de.espirit.firstspirit.opt.myprojectapp.MyProjectConfigurationGui</configur
 able>
6.     </project-app>
```

Eigenschaften:	
<scope>	Project-Anwendungen sind immer project-lokal → **project**
<name> Mandatory	Definiert den Namen, über den die Komponente in FirstSpirit als Projekt-Komponente erreichbar ist und in der Liste der zugehörigen Komponenten des Moduls erscheint.
<class> Mandatory	Klasse, die das Interface `de.espirit.firstspirit.module.ProjectApp` implementiert. konkret: *MyProjectApp implements de.espirit.firstspirit.module.ProjectApp*
<configurable>	Definition der Konfigurationsoberflächen-Klasse. Ist dieses Element nicht definiert, stellt die Projekt-Anwendung in der FirstSpirit Server- und Projektkonfiguration kein Konfiguration-GUI zur Verfügung – der „Konfigurieren"-Button in der FirstSpirit Projektkonfiguration ist deaktiviert. Die Konfigurationsklasse muss das typisierte Interface `de.espirit.firstspirit.module.Configuration<ProjectEnvironment>` implementieren. konkret: *MyProjectConfigurationGui implements de.espirit.firstspirit.module.Configuration<ProjectEnvironment>*

3.8 Getting started

3.8.1 FirstSpirit Version 4.0/4.1/4.2

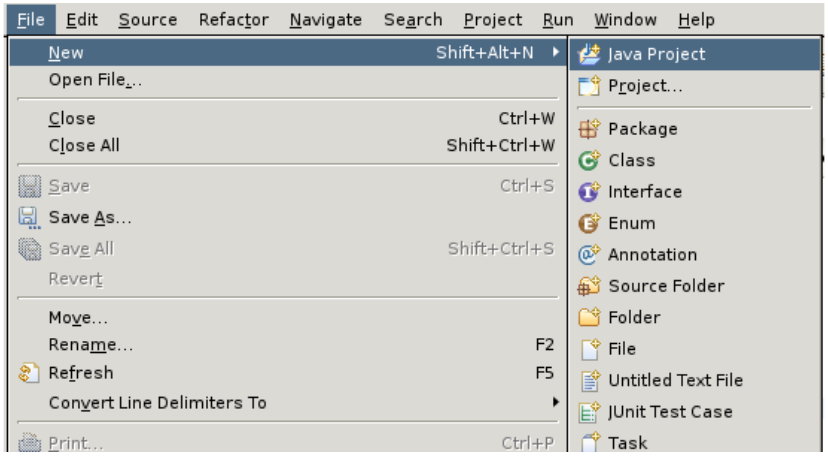
Das fs-client.jar muss im Classpath liegen. Die Access-API Dokumentation sollte in die IDE integriert werden. Das Zip-Archiv (MODDEV4x_modexamples.zip) enthält einige konkrete Implementierungs-Beispiele im Verzeichnis `examples`, sowie die FirstSpirit Developer-API Dokumentation, welche nicht zu verwechseln mit der FirstSpirit Access-API Dokumentation ist. Die Developer-API Dokumentation ist speziell zur Modulentwicklung vorgesehen und befindet sich im Verzeichnis `javadoc` als Zip-Archiv.

Die Beispiele, die eine JDBC-Modul-Library verwenden, werden aus lizenztechnischen Gründen ohne die entsprechende JDBC-Bibliothek ausgeliefert. Diese muss vom jeweiligen Distributor heruntergeladen und im lib-Verzeichnis des Moduls abgelegt werden. Dazu muss die txt-Datei, die als Platzhalter dient, durch die entsprechende Bibliothek ersetzt werden, z.B. „mysql-connector-java-3.1.14-bin.jar.txt" im Verzeichnis Library/MySQL03/lib durch „mysql-connector-java-3.1.14-bin.jar."

3.8.2 Einrichten der IDE

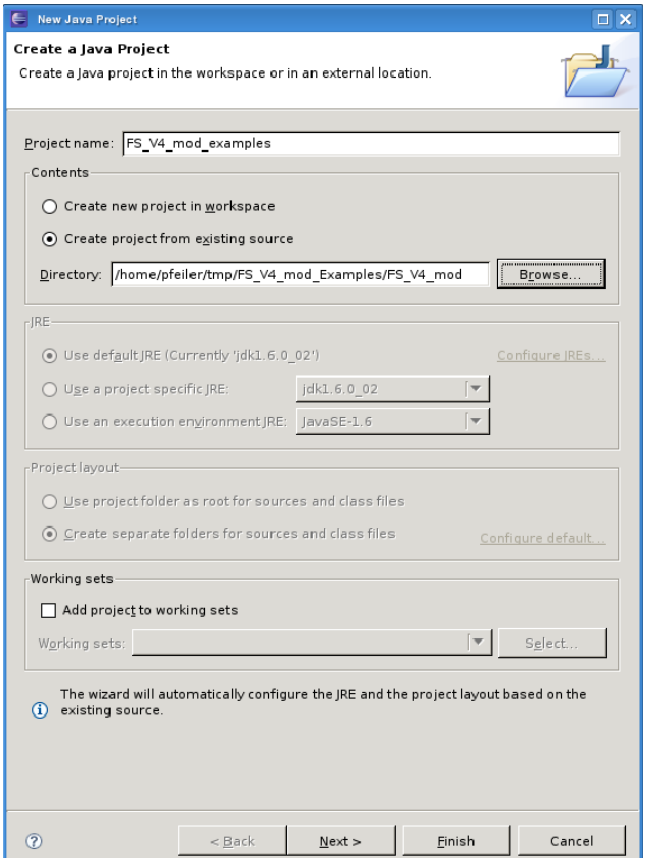
Die FirstSpirit Beispiel-Modul-Suite[3] in die Eclipse IDE importieren.

1. Neues Java-Projekt erstellen

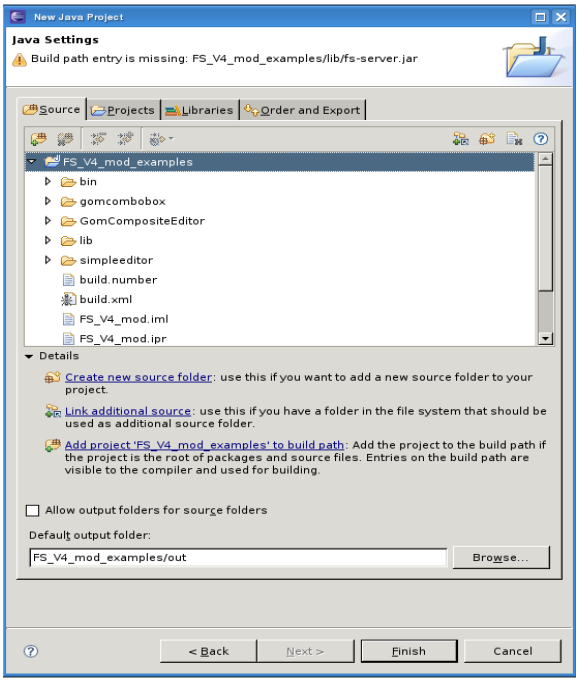


2. Projekt aus den Beispiel-Sourcen erstellen

[3] [10] FirstSpirit Modul-Beispiele auf Basis von Eclipse .classpath, Copyright e-Spirit AG



3. Das fs-server.jar sollte im Classpath liegen kann aber auch später hinzugefügt werden.



Es wird ein komplett compile-fähiges Projekt erstellt mit allen nötigen Archiven im Classpath. Um ein FirstSpirit-Modul-Archiv (*.fsm) zu erstellen, muss vorher `. setenv.sh` unter Unix oder `setenv.bat` unter Windows ausgeführt werden, um verschiedene Umgebungsvariablen zu setzen. In dem Script müssen folgende Umgebungsparameter auf das lokale Environment angepasst werden.

```
JAVA_HOME=/opt/java/jdk1.6.0
CMS_SERVER_HOME=/opt/firstspirit4
PROJECT_HOME=/home/Me/workspace/FS_V4_mod
```

Listing 15: Anpassen der Umgebungsvariablen für das Eclipse Modul Project Beispiel

Jedes Beispiel-Modul hält seine eigene build.xml. Über das Ant-Target ant `assemble-fsm` wird das jeweilige FSM gebaut. Weitere Information zur Integration eines Editors in den FirstSpirit JavaClient finden sich in der README-Datei, welche jedes Modul-Verzeichnis beinhaltet. Bereits erstellte FSM-Archive sind in den Verzeichnissen `target/fsm` eines Moduls zu finden.

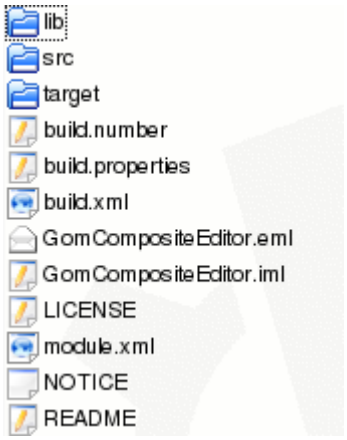


Abbildung 3-1: Modul-Beispiel Build-Verzeichnisstruktur

Die IDE Netbeans kann Eclipse-Projekte importieren, hierzu ist das Plugin Eclipse-Project-Importer erforderlich, diese kann über die Netbeans-Plugin-Verwaltung installiert werden. Um die Beispiel in Intellij zu nutzen, können die Quellen nach dem entpacken des Archivs einfach an eine beliebige Stelle ins Dateisystem kopiert werden und anschließend aus der DIE geladen werden über die IntelliJ (*.iml/*ipr) Projekt-Datei.

3.9 GOM – FirstSpirit GUI Object Model

Das FirstSpirit GUI Object Model definiert die grundlegende Architektur und Funktionalität für die Implementierung neuer sowie die Nutzung bestehender (FirstSpirit-eigener) Eingabekomponenten (Editoren, Kapitel 2.9.1.2 Seite 16). Diese Editoren werden in Java implementiert und verfügen über graphische Oberflächenelemente (Swing) wie z.B. Buttons oder Eingabefelder. In der Regel werden FirstSpirit-Eingabekomponenten entwickelt, um dem Redakteur neue Funktionalitäten zur Verfügung zu stellen, projektspezifische Aufgaben zu vereinfachen oder auf neue Anforderungen im Betrieb zu reagieren.

Editoren nutzen für die Integration der Eingabekomponente das FirstSpirit GUI Object Model, kurz GOM[4]. Hierbei wird über die Definition eines XML-Identifiers, z.B. `CMS_INPUT_SIMPLE_EDITOR_CONTENT` (Kapitel 3.11 Seite 65), in der GOM-Form (`de.espirit.firstspirit.access.store.templatestore.gom.GomFormElement`), die Komponente im FirstSpirit JavaClient eingebunden (siehe auch Kapitel 3.23 Seite 134). Eingabekomponenten in FirstSpirit dienen dazu, die Arbeit (Einpflegen und Bearbeiten von Inhalten) für Redakteure so komfortabel wie möglich zu gestalten. Die gewünschten Eingabekomponenten werden vom Vorlagenentwickler im Registerbereich „Formular" der Vorlage eingebunden, deshalb wird in diesem Dokument der Ausdruck GOM-FORM verwendet. Ein Anwendungsbeispiel für die GOM-Form einer einfachen Editoren-Komponente ist im Kapitel 3.11 ab Seite 65 zu finden.

Prinzipiell erfolgt eine Unterscheidung zwischen GOM-primitiven Elementen und Listen solcher Elemente. Zusätzlich gibt es eine Spezialisierung von Listen, die zur Verarbeitung von Elementgruppen dient. Eine Elementgruppe definiert sich dadurch, dass sie GOM-Elemente (GomLabel, GomRadiobutton) beinhaltet. Da es in diesem Dokument um die Entwicklung eigener, spezialisierter Komponenten geht, wird auf die Vielzahl der in FirstSpirit bereits vorhandenen Eingabekomponenten nicht näher eingegangen. Dieses Kapitel beschränkt sich auf die grundlegende Verwendung von abstrakten GOM-Elementen für die Implementierung eigener Eingabekomponenten und deren GOM. Weiterführende Informationen zum GUI Object Model und den Standard-Eingabekomponenten in FirstSpirit finden sich in der „Online Dokumentation für FirstSpirit – ODFS“[5] im Kapitel *Vorlagenentwicklung / Formulare*.

[4] GOM – FirstSpirit GUI Object Model

[5] [4] Online Dokumentation für FirstSpirit V4.0 - odfs

Ein GOM-Element hat oft ein Swing-Pendant, auf das es durch die GUIEditor-Klasse abgebildet wird.

```
1.     ...
2.     for (final GomSearchField searchField : searchFields) {
3.                                 final int row = tableLayout.getNumRow();
4.                                 tableLayout.insertRow(row,
 TableLayoutConstants.PREFERRED);
5.
6.                                 final JLabel label = new
 JLabel(searchField.getName());
7.                                 topPanel.add(label, "0," + row);
8.
9.                                 final JTextField textField = new
 JTextField();
10.                                _searchFields.put(searchField.getId(),
 textField);
11.                                topPanel.add(textField, "1," + row);
12.                         }
13.    ...
```

3.9.1 Typisierung und Mapping

Grundsätzlich definieren Elemente die für sie zulässigen Attribute und Tags über entsprechende Methoden. Dabei entspricht der Methodenname dem Attribut- bzw. Tagnamen mit „get“-Prefix. Einzige Ausnahme sind Elemente, die `GomList` implementieren. Die für eine Liste zulässigen Elemente werden von der Listenimplementierung selbst als Mapping geliefert (Kapitel 3.9.1.2 Seite 51).

```
1.     public class GomSearchField extends AbstractGomElement {
2.
3.         public static final String TAG = "FIELD";
4.
5.         private String _name;
6.         private Integer _id;
7.     ------------------------------------------------------------------
8.
9.
10.
11.        public GomSearchField() {
12.
```

```
13.         }
14.     ================================================================
15.
16.
17.
18.         public GomSearchField(final String name, final Integer id) {
19.             _name = name;
20.             _id = id;
21.         }
22.     ================================================================
23.
24.
25.
26.         @GomDoc(description = "Search field name", since = "4.0")
27.         public String getName() {
28.             return _name;
29.         }
30.     ================================================================
31.
32.
33.
34.         public void setName(final String name) {
35.             _name = name;
36.         }
37.     ================================================================
38.
39.
40.
41.         @GomDoc(description = "Search field id", since = "4.0")
42.         public Integer getId() {
43.             return _id;
44.         }
45.     ================================================================
46.
47.
48.
49.         public void setId(final Integer id) {
50.             _id = id;
51.         }
52.
53.
54.         //--- GomAbstractFormElement ---//
55.     ================================================================
```

```
56.
57.
58.         @Override
59.         protected String getDefaultTag() {
60.             return TAG;
61.         }
62.     ----------------------------------------------------------------
63.
64.
65.
66.         public void validate() throws IllegalStateException {
67.             super.validate();
68.         }
69.     }
```

Listing 16: GOM-Typisierung und Mapping

Dieses Beispiel definiert eine Klasse GomSearchField. Folgender Tag kann auf diese Klasse abgebildet werden:

```
<FIELD name="street" id="0"/>
```

Wichtig ist hierbei, dass für jedes Attribut eine korrespondierende Methode gleichen Namens mit „get"-Prefix und großgeschriebenem ersten Buchstaben existieren muss. Die Methoden müssen immer parameterlos sein. Der Rückgabetyp entspricht dem erwarteten Attributwert (hier String und Integer nicht int). Jede dieser Methoden muss mit der FirstSpirit-eigenen Gom-Annotation markiert werden. Beispiel: @GomDoc(description = "Search field name"*, since =* "4.0"*).*

3.9.1.1 Direkte Verwendung

Bei direkter Verwendung von Kindelementen muss die beinhaltende Klasse eine entsprechende Methode implementieren, deren Name in Groß-/Kleinschreibung (CamelCase) dem Tagnamen entsprechen muss. Der (konkrete, instanziierbare) Rückgabetyp bestimmt die validen Tag-Inhalte.

```
1.     private static final String TAG = "ADDRESS";
2.     private static final String ENTRY_TAG = GomSearchField.TAG;
3.
4.     @GomDoc( description="Simple form element for address input data
 (street)." since="4.0" )
```

```
5.     public class AddressInputData implements AbstractGomElement {
6.
7.         @GomDoc( description="The street input field" since="4.0" )
8.         public GomSearchField getField () {
9.             return ENTRY_TAG;
10.        }
11.    }
```

Listing 17 : GOM – direkte Verwendung von Kindelementen

XML-Mapping:

```
<ADDRESS>
    <FIELD name="street" id="0"/>
</ADDRESS>
```

Dabei wird der Tagname `FIELD` auf die Methode `getField()` abgebildet und die Attribute des Tags wie oben auf die Methoden von `GomSearchField`.

Der Name des umschließenden Tags `ADDRESS` ist für die implementierende Klasse wie auch zuvor schon ohne Bedeutung. Er wird in dem das Tag umschließenden Element für die Abbildung des Tags auf die Implementierungsklasse benötigt.

3.9.1.2 Mengenwertige Verwendung (Elementlisten)

Bei mengenwertiger Verwendung muss die beinhaltende GUIList-Implementierung ein entsprechendes Mapping bereitstellen.

Beispiel (schematisch): Realisierung einer einfachen Liste von sprachabhängigen Informationsobjekten. Die Implementierung definiert lediglich ein Tag als zulässiges Element. Für eine Erweiterung bzgl. der zu speichernden Informationsobjekte muss eine abgeleitete Klasse lediglich die Methode `getGuiElementMappings()` überschreiben.

Die Methode `get()` wird nicht annotiert. Die Annotation für die Liste erfolgt bei der get-Methode der Listenklasse und die Annotation der Listenelemente erfolgt als Annotation der jeweilig zugewiesenen Klasse auf Typebene.

```
1.
2.     public class GomSearchFields extends AbstractGomList<GomSearchField> {
3.
```

```
4.          private static final String TAG = "SEARCH_FIELDS";
5.          private static final String ENTRY_TAG = GomSearchField.TAG;
6.      ================================================================
7.
8.
9.
10.         public GomSearchFields() {
11.
12.         }
13.     ================================================================
14.
15.
16.         public GomSearchFields(final List<Pair<String, Integer>> fields) {
17.             for (final Pair<String, Integer> field : fields) {
18.                 add(new GomSearchField(field.getKey(), field.getValue()));
19.             }
20.         }
21.     ================================================================
22.
23.
24.
25.         @Override
26.         protected String getDefaultTag() {
27.             return TAG;
28.         }
29.     ================================================================
30.
31.
32.
33.         @Override
34.         public Map<String, Class<? extends GomElement>>
 getGomElementMappings() {
35.             final Map<String, Class<? extends GomElement>> mappings =
 super.getGomElementMappings();
36.             mappings.put(ENTRY_TAG, GomSearchField.class);
37.             return mappings;
38.         }
39.     ================================================================
40.
41.
42.
43.         public Map<String, Integer> getValues() {
```

```
44.             final Map<String, Integer> result = new HashMap<String,
 Integer>();
45.             for (final GomSearchField param : this) {
46.                 result.put(param.getName(), param.getId());
47.             }
48.             return result;
49.         }
50.     }
```

Listing 18: GOM Mengenwertige Verwendung (Elementlisten)

3.9.2 Annotationen

Zur automatischen Verarbeitung der GuiXML (Menge alle Gom-Elemente z.B. eines Absatz-Typs), zur Interpretation als auch zur Dokumentation werden verschiedene Annotationen verwendet.

GomDoc: Die GomDoc-Annotation dient zur Beschreibung anwendungsrelevanter Typen und Methode für die automatische Dokumentation. Sie ermöglicht die Angabe eines Kommentars und einer Bereitstellungs-Version.

```
@GomDoc(description="The name of the street text field",
since="4.0")
```

Mandatory: Indikator, um bestimmte Methoden als zwingend erforderlich zu markieren.

```
@Mandatory
```

InheritAnnotations: Annotation zum Markieren von Stellen, an denen eine Vererbung der Annotation gewünscht ist. Die Vererbung der Annotation wird vom FirstSpirit Annotations-Prozessor geleistet.

```
 @InheritAnnotations
```

3.9.3 Abstrakte GOM-Klassen

Die JavaDoc zu den nachfolgenden abstrakten GOM-Klassen befindet sich in der FirstSpirit Access-Api[6]-Dokumentation im Paket `de.espirit.firstspirit.templatestore.gom`.

[6] [2] Access-API, *http://www.FirstSpirit.de/*, Copyright e-Spirit AG

Alle anwendungsrelevanten Methoden müssen mit der FirstSpirit GomDoc-Notation versehen sein. Diese Objekte bzw. deren XML-Repräsentation wird dann vom GOM-Parser bean-ähnlich über Reflection initialisiert. Für spezialisierte GOM-Elemente wird in den meisten Fällen wohl die Klasse **`AbstractGomElement`** implementiert, welche grundlegende Funktionalitäten für die meisten GOM-Elemente zur Verfügung stellt.

3.9.3.1 AbstractGomComboBox

Die `AbstractGomComboBox`-Klasse stellt die meisten Funktionalitäten zur Verfügung, die eine Combobox nutzt. Ein Anwendungsfall für die Erweiterung dieser abstrakten Klasse ist beispielsweise eine Auto-Completion. Dieses lässt sich ohne die Anpassung des GOMs realisieren, d.h. hier ist reine Swing-Funktionalität erforderlich. Sollen jedoch weitere Attribute oder Elemente für die GOM XML-Definition einer Eingabekomponente zu Verfügung gestellt werden, um diese dann ggf. in der Implementierung auszuwerten, müssen diese in der GOM-Definitions-Klasse der Komponente erstellt werden. Ein Anwendungsfall für das konkrete Beispiel der Auto-Completion-Combobox wäre hier, dem Redakteur die Möglichkeit zu geben, über die Gom-Form-Definition zu entscheiden, ob eine Combobox mit der Completion-Funktionalität ausgestattet sein soll oder nicht.

```
1.    package de.espirit.firstspirit.opt.examples.gom.combobox;
2.
3.    import
 de.espirit.firstspirit.access.store.templatestore.gom.AbstractGomCombobox;
4.    import
 de.espirit.firstspirit.access.store.templatestore.gom.TextGomFormElement;
5.    import de.espirit.firstspirit.access.store.templatestore.gom.YesNo;
6.    import de.espirit.firstspirit.access.editor.ComboboxEditorValue;
7.    import de.espirit.firstspirit.common.text.Designator;
8.    import de.espirit.common.GomDoc;
9.    import de.espirit.common.Default;
10.
11.
12.   /**
13.    * $Date$
14.    *
15.    * @version $Revision$
16.    */
17.   @GomDoc(description = "GomCombobox form element.", since = "4.1")
18.   public class GomCombobox extends AbstractGomCombobox implements
```

```
 TextGomFormElement {
19.
20.         // same TAG defined in the module descript. @see module.xml
 <name>CMS_EXAMPLE_GOMCOMBOBOX</name>
21.         public static final String TAG = "CMS_EXAMPLE_GOMCOMBOBOX";
22.
23.         private String _name;
24.         private Integer _id;
25.         private YesNo _autocompletion;
26.         private YesNo _editable;
27.         private YesNo _extAutoCompletion;
28.     ------------------------------------------------------------------
29.
30.
31.
32.         public GomCombobox() {
33.             super();
34.         }
35.     ------------------------------------------------------------------
36.
37.
38.
39.         public GomCombobox(final String name, final Integer id) {
40.             _name = name;
41.             _id = id;
42.         }
43.     ------------------------------------------------------------------
44.
45.
46.
47.         public void setName(final String name) {
48.             _name = name;
49.         }
50.     ------------------------------------------------------------------
51.
52.
53.
54.         @GomDoc(description = "The Combobox id", since = "4.1")
55.         public Integer getId() {
56.             return _id;
57.         }
58.     ------------------------------------------------------------------
59.
```

```
60.
61.
62.         public void setId(final Integer id) {
63.             _id = id;
64.         }
65.     ------------------------------------------------------------------------
66.
67.
68.
69.         @GomDoc(description="Editable indicator.", since="4.1")
70.         @Default("NO")
71.         public YesNo getEditable() {
72.             return _editable;
73.         }
74.     ------------------------------------------------------------------------
75.
76.
77.
78.         public void setEditable(final YesNo editable) {
79.             _editable = editable;
80.         }
81.     ------------------------------------------------------------------------
82.
83.
84.
85.         @GomDoc(description = "Enable the auto completion feature of the
 combobox.", since = "4.1")
86.         @Default("NO")
87.         public YesNo getAutoCompletion() {
88.             return _autocompletion;
89.         }
90.     ========================================================================
91.
92.
93.
94.         public void setAutoCompletion(final YesNo enable) {
95.             _autocompletion = enable;
96.         }
97.     ------------------------------------------------------------------------
98.
99.
100.
```

```
101.        @GomDoc(description = "Enable the extended combobox auto
 completion mechanism.", since = "4.1")
102.        @Default("NO")
103.        public YesNo getExtAutoCompletion() {
104.            return _extAutoCompletion;
105.        }
106.    ═══════════════════════════════════════════════════════════
107.
108.
109.
110.        public void setExtAutoCompletion(final YesNo enable) {
111.            _extAutoCompletion = enable;
112.        }
113.    ═══════════════════════════════════════════════════════════
114.
115.
116.
117.        /**
118.         * Return the default tag for a gom element.
119.         *
120.         * @return The elements default tag.
121.         */
122.        @Override
123.        protected String getDefaultTag() {
124.            return TAG;
125.        }
126.    ═══════════════════════════════════════════════════════════
127.
128.
129.
130.        @Override
131.        public void validate() throws IllegalStateException {
132.            super.validate();
133.        }
134.    }
```

Listing 19: GOM Beispiel Combobox

```
1.      <CMS_EXAMPLE_GOMCOMBOBOX
2.          name="gomcombobox2"
3.          autoCompletion="no"
4.          editable="yes"
5.          extAutoCompletion="yes"
6.          length="10"
```

```
7.         useLanguages="yes">
8.         <LANGINFOS>
9.           <LANGINFO lang="*" label="GOM ExtAuto-Completion-Combobox
 Example"/>
10.          <LANGINFO lang="DE" label="GOM ExtAuto-Completion-Combobox
 Beispiel"/>
11.          <LANGINFO lang="EN" label="GOM ExtAuto-Completion-Combobox
 Example"/>
12.        </LANGINFOS>
13.      </CMS_EXAMPLE_GOMCOMBOBOX>
```

Listing 20: GOM-Form Representation

Ein vollständiges Beispiel der Combobox-Komponente ist unter Eclipse Modul Suite [7] zu finden.

3.9.3.2 AbstractGomElement

Die abstrakte Klasse `AbstractGomElement` implementiert gemeinsame Funktionalitäten für das GuiXML-Handling.

3.9.3.3 AbstractGomFormElement

Diese Klasse implementiert alle grundlegenden Funktionalitäten für das FirstSpirit GuiXML-Handling und kann von fast jeder zu entwickelnden Editoren-Komponente genutzt werden; um eigene, spezialisierte Editoren zu implementieren (siehe auch Kapitel 3.9.1 Seite 48).

Ein vollständiges Beispiel der Combobox-Komponente ist unter MODDEV40_Sources\EclipseModulSuite zu finden.

3.9.3.4 AbstractGomList<T extends GomElement>

Abstrakte Implementierung einer typisierten Liste, die andere GOM-Elemente aufnehmen kann (Kapitel 3.9.1.2 Seite 51).

[7] [10] FirstSpirit Modul-Beispiele auf Basis von Eclipse .classpath, Copyright e-Spirit AG

Die Implementierung in diesem Bereich ist noch nicht vollständig abgeschlossen. Die Dokumentation erfolgt sobald wie möglich.

3.9.3.5 AbstractGomSelect

Abstrakte Radiobutton-Implementierung.

Die Implementierung in diesem Bereich ist noch nicht vollständig abgeschlossen. Die Dokumentation erfolgt sobald wie möglich.

3.10 FirstSpirit Security Architektur – Java Web Start (javaws)

Der FirstSpirit JavaClient und die FirstSpirit Server- und Projektkonfiguration werden über eine `jnlp`-Datei, d.h. über Java Web Start ausgeführt. Daraus ergeben sich zunächst einmal Einschränkungen in der Nutzung einiger Funktionalitäten für nicht von e-Spirit signierte Module bzw. in den Jar-Archiven enthaltene Klassen. Java-Programme laufen normalerweise in einer „Sandbox" ab. D.h. sie haben keinen vollwertigen Zugriff auf den Rechner und dessen Ressourcen, auf dem sie ausgeführt werden, sondern können nur innerhalb der Java-VM (Java Virtual Machine) arbeiten. Der Zugriff auf lokale Ressourcen wie Dateien, Zwischenablage, Netzwerk etc. geschieht über einen Security-Manager. Dieser ist normalerweise entsprechend den Sicherheitsanforderungen konfiguriert. Praktisch heißt das: Programme, die direkt (z.B. von der Konsole) gestartet werden, haben alle Privilegien; Programme, die über einen Netzwerkverbindung gestartet werden (Applets oder Java Web Start-Applikationen), haben erstmal keine Zugriffsprivilegien auf lokale Ressourcen. Die FirstSpirit-eigenen Module sind mit dem „e-Spirit AG“-Schlüssel signiert. Dieser ist wiederum Bestandteil der FirstSpirit-eigenen Security-Policy. Des Weiteren ist der Schlüssel von einer Root-Authority bestätigt, die wiederum dem Java Zertifikat Manager bekannt ist. Im Allgemeinen erhält man bei einer Applikation, die gegen die Sicherheitsrichtlinien verstößt, eine recht informative Meldung (Popup-Dialog). Alle auftretenden Security-Exceptions werden ggf. in der Java Console ausgeben.

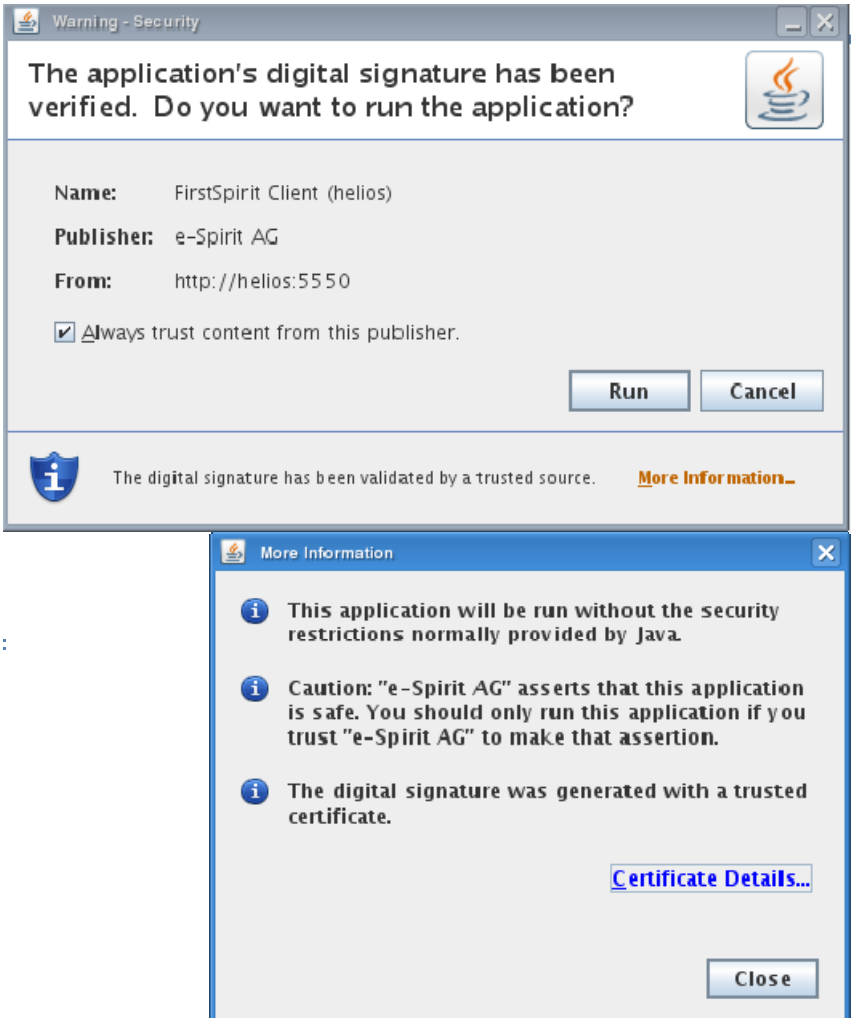


Abbildung 3-2: Java Web Start Security Warning-Dialog

Sollen die in einem Modul enthaltenen Archive und somit deren Klassen mit einem eigenen vertrauenswürdigen Schlüssel signiert werden, um Zugriff auf lokale Ressourcen bzw. sicherheitsrelevante Ressourcen zu erlangen, sind somit zur Ausführung der Web Start-Anwendung zwei vertrauenswürdige Schlüssel innerhalb des FirstSpirit JavaClients vorhanden. Nun sollte man annehmen, dass zwei „recht informative Meldungen“ (Popup-Dialog) beim ersten Starten des JavaClients angezeigt werden, wie es von Applets bekannt ist. Damit sollte die Möglichkeit bestehen, beide Zertifikate zu akzeptieren, d.h. als vertrauenswürdig einzustufen und in den Java Web Start-internen Zertifikat-Cache zu übernehmen. Laut JSR-56[8] ist dieses nicht der Fall. Es wird immer nur ein Security-Warning-Dialog angezeigt, dieses ist immer das vertrauenswürdige e-Spirit-Zertifikat.

[8] JSR-56 [9] – JavaTM Network Launching Protocol (JNLP) Specification ("Specification")

– Was bedeutet dieses nun für externe Komponenten/Module, die auf sicherheitsrelevante Funktionalitäten zugreifen müssen?

Jar-Archive bzw. die darin enthaltenen Klassen müssen signiert sein, der Signaturschlüssel muss im Java Web Start Zertifikat-Cache auf dem Client importiert werden (importierte Zertifikate überprüfen mittels: `jcontrol` unter dem Reiter Security.)

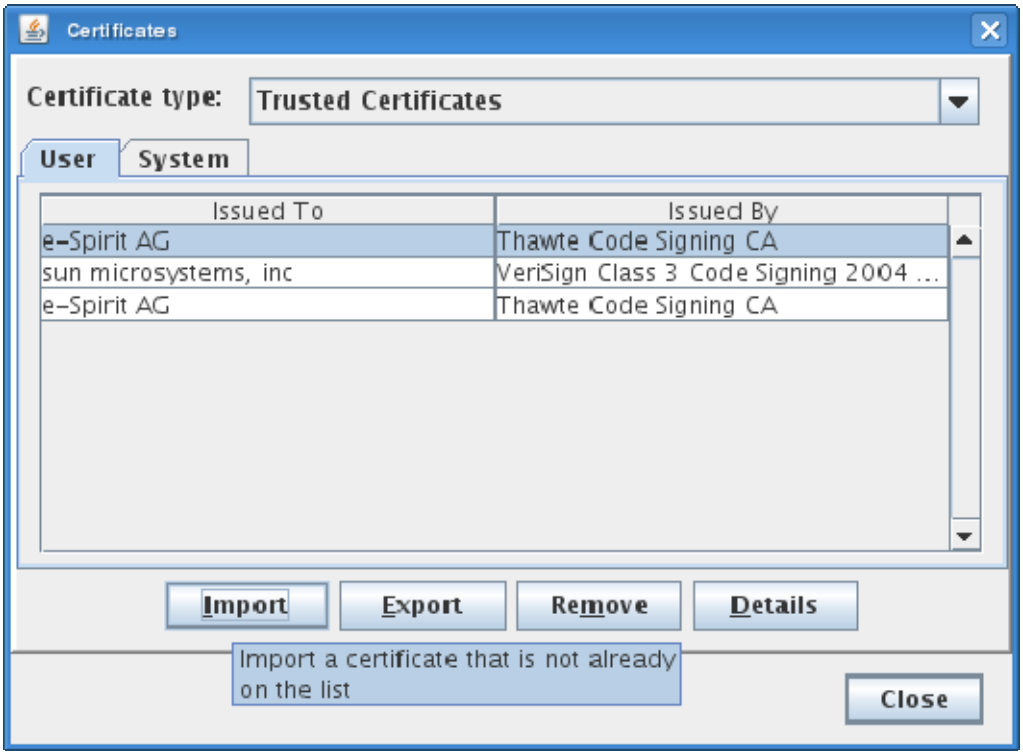


Abbildung 3-3: Java Web Start Zertifikat-Cache / Import von Zertifikaten

Da die zuvor erläuterte Vorgehensweise bei einer Vielzahl von FirstSpirit-Nutzern auf unterschiedlichen Client-Rechnern nicht praktikabel ist, besteht in ***FirstSpirit Version 4.1*** *die komfortable Alternative -- die Verwendung des* ***FirstSpirit*** *eigenen* ***Security-Managers*** *sowie des* ***Security-Classloaders.*** *Die Zuweisung von Rechten auf Modulebene wird über die FirstSpirit Server- und Projektkonfiguration vorgenommen (siehe Kapitel 3.10.1 Seite 62). In der* ***FirstSpirit Version 4.0*** *werden alle Module automatisiert durch den FirstSpirit Security-Manager als vertrauenswürdig eingestuft.*

3.10.1 Verwendung des FirstSpirit Security-Managers/Classloaders

In der FirstSpirit Server- und Projektkonfiguration kann jedes installierte Modul (mit Ausnahme der FirstSpirit System-Module) optional mit Rechten auf lokale System-Ressourcen ausgestattet werden. Über die Schaltfläche „Konfigurieren“ bei zuvor selektiertem Modul besteht die Option, einem Modul, das sicherheitsrelevante Operationen durchführt, z.B. den Zugriff auf die Zwischenablage (java.awt.AWTPermission ClipboardAccess), zu vertrauen. Diesem Modul können Rechte zur Durchführung der Operationen zugewiesen werden. Dies geschieht intern über den FirstSpirit Security-Manager/Classloader. Der Vorteil dieses Verfahrens ist es, dass nicht jedem Modul voller Zugriff durch den FirstSpirit-Serveradministrator gewährt wird bzw. gewährt werden muss. Es müssen nicht 1..* Zertifikate manuell in den Java Web Start Cache importiert werden. Die Konfigurationsoberfläche zum Setzen der Modulrechte in der FirstSpirit Server- und Projektkonfiguration stellt sich wie folgt dar:

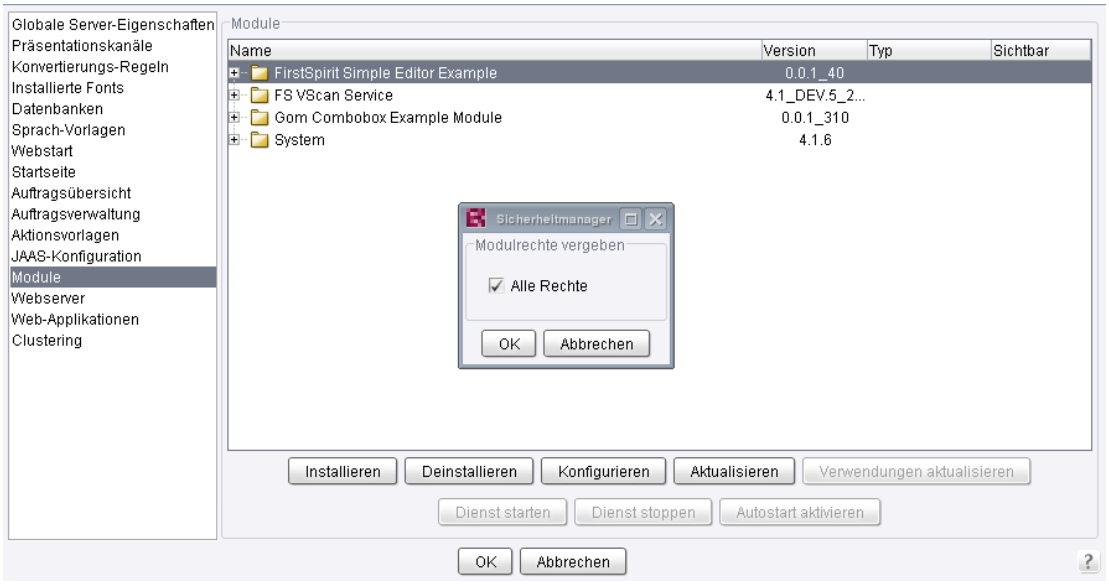


Abbildung 3-4: Setzen der Modulrechte

3.10.2 Zertifikat für Testzwecke erzeugen

Nachfolgend werden beispielhaft die einzelnen Schritte beschrieben, wie sich ein Zertifikat für Testzwecke erzeugen lässt. Die daraus erzeugte Signierung wird von `javaws` als nicht vertrauenswürdig eingestuft, da es sich nur um ein Zertifikat für eigene Tests handelt.

```
# Create a CA certificate that we will use to sign our certificate

# NOTE: make sure the the organisation name for this cert is
different than the keystore/certificates created later on.

openssl req -x509 -new -out CAcert.crt -keyout CAKey.key -days 365


# Generate a server keystore entry

keytool -genkey -keyalg RSA -alias mycert -dname "CN=localhost,
OU=Test AG, O=Test AG, L=Beauchief, S=Dortmund, C=DE" -keystore
keystore.jks -storepass changeit


# Make a certificate request

keytool -certreq -v -alias mycert -keystore keystore.jks -
storepass changeit -file mycert_request.csr


# Sign the certificate request

openssl x509 -req -in mycert_request.csr -CA CA\CAcert.crt -CAkey
CAKey.key -CAcreateserial -out mycert_response.crt -extfile
/etc/ssl/openssl.cnf -extensions v3_ca -days 365
```

```
# View the certificate, Make sure it is version 3
keytool -printcert -file mycert_response.crt


# Add the CA cert to the CA keystore
keytool -import -trustcacerts -alias cacert -file CAcert.crt -
keystore cacerts.jks -storepass changeit


# Add the CA cert to the keystore so that it can find the chain
keytool -import -trustcacerts -alias cacert -file CAcert.crt -
keystore keystore.jks -storepass changeit


# Add the response certificate that has been signed by the CA,
Should get a response of "Certificate reply was installed in
keystore"
keytool -import -alias mycert -file mycert_response.crt -keystore
keystore.jks -storepass changeit
```

3.11 Eine einfache Editor-Komponente

3.11.1 Modul- und Komponenten-Descriptor

```
1.    <!DOCTYPE module SYSTEM "http://www.FirstSpirit.de/module.dtd">
2.    <module>
3.        <name>FirstSpirit Simple Editor Example</name>
4.        <version>@VERSION@</version>
5.        <description>FirstSpirit Simple Editor Module
 Example</description>
6.        <vendor>e-Spirit AG</vendor>
7.          <components>
8.            <editor>
9.                <name>CMS_INPUT_SIMPLE_EDITOR_CONTENT</name>
10.               <description>FirstSpirit Simple Example Editor
 component.</description>
11.
 <class>de.espirit.firstspirit.opt.example.editor.simple.FSEditorContentComp
 onent</class>
12.               <resources>
13.                   <resource version="0.0.1" minVersion="0.0"
 maxVersion="0.0.1">lib/fs-seditor-example.jar</resource>
14.               </resources>
15.           </editor>
16.       </components>
17.   </module>
```

Listing 21: Einfacher Editor – Modul- und Komponenten-Descriptor

3.11.2 Ant build.xml – Komplettes Beispiel

```
1.    <?xml version="1.0" encoding="ISO-8859-1"?>
2.    <!--
3.        to generate fsm module file call
4.        ant -f
 src/de/espirit/firstspirit/opt/example/editor/simple/build.xml [TARGET]
5.
6.        Targets:
7.            jar         builds the jar file.
8.            fsm         builds the module file.
9.            deploy      deploys the module file.
10.
11.   -->
12.   <project name="SimpleEditorExample" default="fsm">
13.
14.       <property environment="env"/>
15.             <property name="env.PROJECT_HOME"
 value="/path/to/your/module/project_home"/>
16.       <property name="path"
 value="de/espirit/firstspirit/opt/example/editor/simple"/>
17.       <property name="classes-dir"
 value="${env.PROJECT_HOME}/classes.modules"/>
18.       <property name="main-jar-file" value="fs-seditor-example.jar"/>
19.       <property name="main-fsm-file" value="fs-seditor-example.fsm"/>
20.       <property name="main-tmp-dir" value="${env.PROJECT_HOME}/tmp/fs-
 seditor-example" />
21.
22.       <path id="class.path">
23.           <pathelement path="${env.PROJECT_HOME}/classes"/>
24.       </path>
25.
26.       <fileset id="classes.main" dir="${classes-dir}">
27.           <include
 name="de/espirit/firstspirit/opt/example/editor/simple/*.class"/>
28.       </fileset>
29.
30.       <target name="main-jar" description="Builds all jar files.">
31.           <mkdir dir="${classes-dir}"/>
32.           <javac debug="on" srcdir="${env.PROJECT_HOME}/src"
 destdir="${classes-dir}" includes="${path}/**" target="1.5" source="1.5">
33.               <classpath refid="class.path"/>
```

```
34.             </javac>
35.             <jar jarfile="${env.PROJECT_HOME}/${main-jar-file}"
 excludes="**/*.dependency" duplicate="preserve" keepcompression="true">
36.                 <fileset refid="classes.main"/>
37.             </jar>
38.     </target>
39.         <target name="deploy" depends="fsm" description="Deploys the
 module file (*.fsm) to the server.">
40.             <copy file="${env.PROJECT_HOME}/${main-fsm-file}"
 todir="${env.CMS_SERVER_HOME}/data/modules" overwrite="true"/>
41.         </target>
42.         <target name="fsm" depends="main-fsm" description="Builds all
 module files (*.fsm)."/>
43.         <target name="main-fsm" depends="main-jar " description="Builds
 the main module file (*.fsm).">
44.             <mkdir dir="${main-tmp-dir}/lib"/>
45.             <mkdir dir="${main-tmp-dir}/META-INF"/>
46.             <copy file="${env.PROJECT_HOME}/${main-jar-file}"
 todir="${main-tmp-dir}/lib" preservelastmodified="true" overwrite="true" />
47.             <copy file="module.xml" tofile="${main-tmp-dir}/META-
 INF/module.xml" overwrite="true">
48.                 <filterset>
49.                     <filter token="VERSION" value="1.1.1"/>
50.                 </filterset>
51.             </copy>
52.             <jar jarfile="${env.PROJECT_HOME}/${main-fsm-file}"
 basedir="${main-tmp-dir}" excludes="**/*.dependency" duplicate="preserve"
 keepcompression="true"/>
53.         </target>
54.     </project>
```

Listing 22: ant build.xml mit den Targets main-jar, main-fsm, deploy

Wird die Modul-Installation über ein „DEPLOY“-Target angestoßen, ist ein Neustart des FirstSpirit-Servers notwendig. Bei der Installation über die Server- und Projektkonfiguration entfällt der Server-Neustart.

3.11.3 Basis-Implementierungsmodell (Klassengrundgerüst)

Interface: FSEditorValue
Package: de.espirit.firstspirit.opt.example.editor.simple
Erweitert das `de.espirit.firstspirit.access.editor.EditorValue<T>`-Interface (`<SIMPLE_EDITOR_CONTENT>T</SIMPLE_EDITOR_CONTENT>`), definiert die XML-Darstelllung (siehe Kapitel 2.9.1.2 Seite 16), in der der Wert der Eingabekomponente gespeichert wird.
`<SIMPLE_EDITOR_CONTENT>...</SIMPLE_EDITOR_CONTENT>`.
`FSEditorContentValueImpl` ist die konkrete Implementierung.

Class 1: de.espirit.firstspirit.opt.example.editor.simple.FSEditorValue

```
1.    package de.espirit.firstspirit.opt.example.editor.simple;
2.
3.    import de.espirit.firstspirit.access.editor.EditorValue;
4.
5.
6.    public interface FSEditorValue extends EditorValue<Object> {
7.
8.        // XML storage TagName
9.        public static final String SIMPLE_EDITOR_CONTENT_ELEMENT =
 "SIMPLE_EDITOR_CONTENT";
10.
11.
12.       public GomFSEditorContent getForm();
13.   }
```

Listing 23: de.espirit.firstspirit.opt.example.editor.simple.FSEditorValue

Das Interface `EditorValue` wird typisiert, d. h. der zu verwaltende Wertetyp wird über die (Java 5) Generics-Funktionalität innerhalb der Implementierung festgelegt. Die Implementierung verwaltet sprachabhängig die jeweiligen Werte und bietet eine generische Methode für den Zugriff auf diese Werte. Ausgeliefert wird der Werte-Wrapper des Typs, mit dem der EditorValue typisiert wurde.

Klasse: FSEditorContentValueImpl
Package: de.espirit.firstspirit.opt.example.editor.simple
Bietet grundlegende Funktionalitäten `(parseValue, formatValue)` für Editoren bzw. für deren XML-Darstellungs-String der spezifizierten Sprache.
`<SIMPLE_EDITOR_CONTENT>value</SIMPLE_EDITOR_CONTENT>`

Class 2: de.espirit.firstspirit.opt.example.editor.simple.FSEditorContentValueImpl

```
1.    package de.espirit.firstspirit.opt.example.editor.simple;
2.
3.    import de.espirit.common.base.Logging;
4.    import de.espirit.firstspirit.access.Language;
5.    import
 de.espirit.firstspirit.access.editor.value.InvalidValueException;
6.    import
 de.espirit.firstspirit.access.editor.value.InvalidValueRuntimeException;
7.    import de.espirit.firstspirit.access.search.Request;
8.    import de.espirit.firstspirit.access.search.Match;
9.    import de.espirit.firstspirit.access.search.PatternClause;
10.   import
 de.espirit.firstspirit.client.access.editor.AbstractEditorValue;
11.   import org.jetbrains.annotations.NotNull;
12.   import org.w3c.dom.Element;
13.
14.   import java.util.HashMap;
15.   import java.util.List;
16.   import java.util.ArrayList;
17.
18.
19.   public class FSEditorContentValueImpl extends
 AbstractEditorValue<Object> implements FSEditorValue {
20.
21.
22.       /**
23.        * Constructs a new FSEditorContentValueImpl.
24.        */
25.       public FSEditorContentValueImpl() {
26.       }
27.
28.
29.       //-- AbstractEditorValue ---//
30.   ═══════════════════════════════════════════════════
```

```
31.
32.
33.
34.         /**
35.          * Parse the value from the given name-keyed map of attributes for
 the specified language. Called on reload
36.          * (JavaClient: F5 pressed)
37.          *
38.          * @param language   The language to parse for.
39.          * @param attributes The map of attributes.
40.          */
41.         @Override
42.         protected Object parseValue(final Language language, final
 HashMap<String, Element> attributes) {
43.             final Element element =
 attributes.get(SIMPLE_EDITOR_CONTENT_ELEMENT);
44.
45.             return element != null ? parseString(element) : null;
46.         }
47.     ================================================================
48.
49.
50.
51.         /**
52.          * Convert the contained value for the specified language to an
 XML representation in form of a sequence of attributes.
53.          * Called for each language on CTRL+S pressed or language tab
 change.
54.          *
55.          * @param language The language the value is stored for.
56.          * @return An XML representation of the value.
57.          */
58.         @Override
59.         protected String formatValue(final Language language) {
60.             final Object value = internalGet(language);
61.
62.             if (isEmpty(value)) {
63.                 return '<' + SIMPLE_EDITOR_CONTENT_ELEMENT + "/>";// empty
 set
64.             }
65.
66.             Logging.logDebug("### formatValue() , language:" + language +
 " <" + SIMPLE_EDITOR_CONTENT_ELEMENT + '>' + value + "</" +
```

```
SIMPLE_EDITOR_CONTENT_ELEMENT + '>', FSEditorContentValueImpl.class);
67.
68.             return '<' + SIMPLE_EDITOR_CONTENT_ELEMENT + '>' + value +
"</" + SIMPLE_EDITOR_CONTENT_ELEMENT + '>';
69.         }
70.
71.
72.         //--- EditorValue ---//
73.     ═══════════════════════════════════════════════════════════════
74.
75.
76.
77.         /**
78.          * Get the type of value the editor takes. The returned value must
not be null and corresponds to the generic type T.
79.          *
80.          * @return The value's type.
81.          */
82.         @NotNull
83.         public Class<Object> getValueType() {
84.             Logging.logDebug("### getValueType()",
FSEditorContentValueImpl.class);
85.             return Object.class;
86.         }
87.     ═══════════════════════════════════════════════════════════════
88.
89.
90.
91.         /**
92.          * Find matches for the given request.
93.          *
94.          * @param language The language to seach in.
95.          * @param request  The request to find matches for.
96.          * @return A list of editor specific match description objects.
97.          */
98.         @NotNull
99.         public List<Match> getMatches(final Language language, final
Request request) {
100.            final List<Match> list = new ArrayList<Match>();
101.            list.add(new Match(new PatternClause("the SearchString"), 0,
1)); // dummy search to fit the @NotNull annotation
102.
103.            Logging.logDebug("### getMatches()",
```

```
 FSEditorContentValueImpl.class);
104.            return list;
105.        }
106.    ================================================================
107.
108.
109.
110.        /**
111.         * @deprecated use specific attribute accessor
112.         */
113.        @Deprecated
114.        public Object getValue(final Language language) {
115.            Logging.logDebug("### getValue()",
 FSEditorContentValueImpl.class);
116.            return get(language);
117.        }
118.    ================================================================
119.
120.
121.
122.        /**
123.         * @deprecated use specific attribute accessor
124.         */
125.        @Deprecated
126.        public void setValue(final Language language, final Object value)
 {
127.            Logging.logDebug("### setValue()",
 FSEditorContentValueImpl.class);
128.            if (value == null) {
129.                clear(language);
130.            } else {
131.                try {
132.                    set(language, value);
133.                } catch (InvalidValueException e) {
134.                    throw new InvalidValueRuntimeException(e);
135.                }
136.
137.            }
138.        }
139.
140.
141.        //--- FSEditorValue ---//
142.    ================================================================
```

```
143.
144.
145.
146.        /**
147.         * @return the gui main form widget
148.         */
149.        @Override
150.        public GomFSEditorContent getForm() {
151.            return (GomFSEditorContent) super.getForm();
152.        }
153.    ================================================================
154.
155.
156.
157.        /**
158.         * Getter for property '{@link GomFSEditorContent#_maxRows}
 maxRows' from {@link GomFSEditorContent}.
159.         * Adds an attribute to the default TAG
 "<CMS_INPUT_SIMPLE_EDITOR_CONTENT maxRows="10">"
160.         *
161.         * @see GomFSEditorContent
162.         * @return Value for property 'maxRows'.
163.         */
164.        public int getMaxRows() {
165.            Logging.logDebug("### getMaxRows()",
 FSEditorContentValueImpl.class);
166.            if (getForm().getMaxRows() != null) {
167.                return getForm().getMaxRows().intValue();
168.            }
169.            return 0;
170.        }
171.
172.    }
```

Listing 24: de.espirit.firstspirit.opt.example.editor.simple.FSEditorContentValueImpl

Klasse: GomFSEditorContent
Package: de.espirit.firstspirit.opt.example.editor.simple
Definition des GOM[9]-Form XML-Identifiers, `CMS_INPUT_SIMPLE_EDITOR_CONTENT`, sowie dessen mögliche Attribute (maxRows) und für die Attribute benötigte Methoden (Getter/Setter).

Class 3: de.espirit.firstspirit.opt.example.editor.simple.GomFSEditorContent

```
1.  package de.espirit.firstspirit.opt.example.editor.simple;
2.
3.  import de.espirit.common.GomDoc;
4.  import
 de.espirit.firstspirit.access.store.templatestore.gom.AbstractGomFormElemen
 t;
5.  import de.espirit.firstspirit.common.number.PositiveInteger;
6.
7.
8.  /**
9.   * Define GOM (GuiObjectModel) Tags and Attributes
10.  */
11. @GomDoc(description = "FS Simple Editor Example", since = "4.0")
12. public class GomFSEditorContent extends AbstractGomFormElement {
13.
14.     // Editor XML-Identifier: integrates the editor component into the
 GOM form template
15.     public static final String TAG =
 "CMS_INPUT_SIMPLE_EDITOR_CONTENT";
16.
17.     // Represents an attribute in the XML-Identifier context
 <CMS_INPUT_SIMPLE_EDITOR_CONTENT maxRows="10">"
18.     private PositiveInteger _maxRows = PositiveInteger.valueOf(10);//
 defaults to 10 rows
19. ═══════════════════════════════════════════════════════════════
20.
21.
22.
23.     /**
```

[9] GOM – GUI Object Model

```
24.          * Constructs a new GomFSEditorContent.
25.          */
26.         public GomFSEditorContent() {
27.         }
28.     ═══════════════════════════════════════════════════════════
29.
30.
31.
32.         /**
33.          * Getter for property 'maxRows'.
34.          * Adds an attribute to the default TAG
 "<CMS_INPUT_SIMPLE_EDITOR_CONTENT maxRows="10">"
35.          *
36.          * @return Value for property 'maxRows'.
37.          */
38.         @GomDoc(description = "Maximum number of textarea rows.", since =
 "4.0")
39.         public PositiveInteger getMaxRows() {
40.             return _maxRows;
41.         }
42.     ═══════════════════════════════════════════════════════════
43.
44.
45.
46.         /**
47.          * Setter for property 'maxRows'.
48.          * Adds an attribute to the default TAG
 "<CMS_INPUT_SIMPLE_EDITOR_CONTENT maxRows="10">"
49.          *
50.          * @param maxRows Value to set for property 'maxRows'.
51.          */
52.         @GomDoc(description = "Set the maximum number of textarea rows.",
 since = "4.0")
53.         public void setMaxRows(final PositiveInteger maxRows) {
54.             _maxRows = maxRows;
55.         }
56.
57.
58.         //--- AbstractGomElement ---//
59.     ═══════════════════════════════════════════════════════════
60.
61.
62.
```

```
63.         /**
64.          * Return the default tag for a gom element.
65.          *
66.          * @return The elements default tag.
67.          */
68.         @Override
69.         protected String getDefaultTag() {
70.             return TAG;
71.         }
72.     ------------------------------------------------------------
73.
74.
75.
76.         /**
77.          * Called on the Serverside of life
78.          *
79.          * @see
80.          * @throws IllegalStateException
81.          */
82.         @Override
83.         public void validate() throws IllegalStateException {
84.             System.err.println("#### GOM validate()");
85.             super.validate();
86.             // e.g. check for not null
87.
88.         }
89.     }
```

Listing 25: de.espirit.firstspirit.opt.example.editor.simple.GomFSEditorContent

Klasse: FSEditorContentComponent
Package: de.espirit.firstspirit.opt.example.editor.simple
Basisklasse der Editor-Komponte, hier werden alle für den Editor und für das FirstSpirit Modul-Konzept benötigten Klassen definiert. Die Klasse initialisiert die Komponente, des Weiteren kann hier auf Ereignisse wie Installation, Deinstallation, Aktualisierung reagiert werden (siehe auch Kapitel 2.6, 2.1, 2.2). Die Klasse muss das **`Editor<ServerEnvironment>`** Interface implementieren.

Class 4: de.espirit.firstspirit.opt.example.editor.simple.FSEditorContentComponent

Das Interface `Editor<ServerEnvironment>` hat Methoden, z.B. `getGomElementClass`, die die entsprechende Klasse des Moduleditors

zurückliefern müssen.

```
1.     package de.espirit.firstspirit.opt.example.editor.simple;
2.
3.     import de.espirit.firstspirit.access.editor.EditorValue;
4.     import
 de.espirit.firstspirit.access.store.templatestore.gom.GomElement;
5.     import
 de.espirit.firstspirit.access.store.templatestore.gom.GomModule.Kind;
6.     import de.espirit.firstspirit.client.access.editor.swing.GuiEditor;
7.     import de.espirit.firstspirit.module.Editor;
8.     import de.espirit.firstspirit.module.ServerEnvironment;
9.     import de.espirit.firstspirit.module.descriptor.EditorDescriptor;
10.    import de.espirit.common.base.Logging;
11.
12.    import java.util.EnumSet;
13.
14.
15.    /**
16.     * FirstSpirit editor compoment.
17.     */
18.    public class FSEditorContentComponent implements
 Editor<ServerEnvironment> {
19.
20.        private static final Class<FSEditorContentComponent> LOGGER =
 FSEditorContentComponent.class;
21.
22.        //--- Editor ---//
23.    ----------------------------------------------------------------------
24.
25.
26.
27.        /**
28.         * Returns editor's {@link
 de.espirit.firstspirit.access.editor.EditorValue} class.
29.         *
30.         * @return editor's {@link
 de.espirit.firstspirit.access.editor.EditorValue} class.
31.         */
32.        public Class<? extends EditorValue> getEditorValueClass() {
```

```
33.             return FSEditorContentValueImpl.class;
34.         }
35.     ═══════════════════════════════════════════════════════════
36.
37.
38.
39.         /**
40.          * Returns editor's {@link
 de.espirit.firstspirit.client.access.editor.swing.GuiEditor} class.
41.          *
42.          * @return editor's {@link
 de.espirit.firstspirit.client.access.editor.swing.GuiEditor} class.
43.          */
44.         public Class<? extends GuiEditor> getGuiEditorClass() {
45.             return FSEditorContentGuiEditor.class;
46.         }
47.     ═══════════════════════════════════════════════════════════
48.
49.
50.
51.         /**
52.          * Returns editor's {@link
 de.espirit.firstspirit.access.store.templatestore.gom.GomElement} class.
53.          *
54.          * @return editor's {@link
 de.espirit.firstspirit.access.store.templatestore.gom.GomElement} class.
55.          */
56.         public Class<? extends GomElement> getGomElementClass() {
57.             return GomFSEditorContent.class;
58.         }
59.     ═══════════════════════════════════════════════════════════
60.
61.
62.
63.         /**
64.          * Returns editor's {@link
 de.espirit.firstspirit.access.store.templatestore.gom.GomElement}, {@link
65.          *
 de.espirit.firstspirit.access.store.templatestore.gom.GomModule.Kind}
 restrictions.
66.          *
67.          * @return editor's {@link
 de.espirit.firstspirit.access.store.templatestore.gom.GomElement}, {@link
```

```
68.          *
 de.espirit.firstspirit.access.store.templatestore.gom.GomModule.Kind}
 restrictions.
69.          */
70.         public EnumSet<Kind> getGomElementRestrictions() {
71.             return EnumSet.allOf(Kind.class);
72.         }
73.
74.
75.         //--- Component ---//
76.     ================================================================
77.
78.
79.
80.         /**
81.          * Initializes this component with the given {@link
 de.espirit.firstspirit.module.descriptor.ComponentDescriptor
82.          * descriptor} and {@link
 de.espirit.firstspirit.module.ServerEnvironment environment}. No other
 method will be called
83.          * before the component is initialized!
84.          *
85.          * @param descriptor useful descriptor information for this
 component.
86.          * @param env       useful environment information for this
 component.
87.          */
88.         public void init(final EditorDescriptor descriptor, final
 ServerEnvironment env) {
89.             // do something
90.             Logging.logDebug("init()", LOGGER);
91.         }
92.     ================================================================
93.
94.
95.
96.         /**
97.          * Event method: called if Component was successfully installed
 (not updated!)
98.          */
99.         public void installed() {
100.             // do something
101.             Logging.logDebug("installed()", LOGGER);
```

```
102.       }
103.   ═══════════════════════════════════════════════════════
104.
105.
106.
107.       /**
108.        * Event method: called if Component was in uninstalling procedure
109.        */
110.       public void uninstalling() {
111.           // do something
112.           Logging.logDebug("uninstalling()", LOGGER);
113.       }
114.   ═══════════════════════════════════════════════════════
115.
116.
117.
118.       /**
119.        * Event method: called if Component was completely updated
120.        *
121.        * @param oldVersionString old version, before component was
 updated
122.        */
123.       public void updated(final String oldVersionString) {
124.           // do something
125.           Logging.logDebug("updated(" + oldVersionString + ")", LOGGER);
126.       }
127.   }
```

Listing 26: de.espirit.firstspirit.opt.example.editor.simple.FSEditorContentComponent

Klasse: FSEditorContentGuiEditor
Package: de.espirit.firstspirit.opt.example.editor.simple
Lädt/hält die initiale Module-/Komponenten-Oberfläche. Speichert Eingaben, reagiert auf GUI-Events (Benachrichtigung `AccessChildTreeModel.StoreListener` bei Änderungen), locked Eingabeelemente.

Class 5: de.espirit.firstspirit.opt.example.editor.simple.FSEditorContentGuiEditor

```
1.    package de.espirit.firstspirit.opt.example.editor.simple;
2.
3.    import de.espirit.common.base.Logging;
```

```
4.    import de.espirit.firstspirit.access.Language;
5.    import
 de.espirit.firstspirit.access.editor.value.InvalidValueException;
6.    import
 de.espirit.firstspirit.client.access.editor.swing.AbstractValueGuiEditor;
7.
8.    import javax.swing.JComponent;
9.    import javax.swing.JPanel;
10.   import javax.swing.JTextArea;
11.   import javax.swing.JTextField;
12.   import javax.swing.JEditorPane;
13.   import javax.swing.event.DocumentEvent;
14.   import javax.swing.event.DocumentListener;
15.   import java.awt.BorderLayout;
16.
17.
18.   /**
19.    * Class FSEditorContentGuiEditor. this class is a good starting point
 to split into View and Control
20.    * <p/>
21.    * Created on Jan 22, 2008
22.    */
23.   public class FSEditorContentGuiEditor extends AbstractValueGuiEditor {
24.
25.       // SetUp the LOGGER
26.       private static final Class<FSEditorContentGuiEditor> LOGGER =
 FSEditorContentGuiEditor.class;
27.
28.       private EditorFrame _component;
29.       private JPanel _fsEditorPanel;
30.       private JTextArea _jTextArea;
31.   ------------------------------------------------------------------------
32.
33.
34.
35.       /**
36.        * Constructs a new FSEditorContentGuiEditor.
37.        */
38.       public FSEditorContentGuiEditor() {
39.       }
40.
41.
42.       //--- AbstractValueGuiEditor ---//
```

```
43.     ================================================================
44.
45.
46.
47.         /**
48.          * Get the EditorValueImpl {@inheritDoc}
49.          */
50.         @Override
51.         protected FSEditorValue getEditorValue() {
52.             return (FSEditorValue) super.getEditorValue();
53.         }
54.     ================================================================
55.
56.
57.
58.         /**
59.          * {@inheritDoc}
60.          */
61.         @Override
62.         public void validate() throws InvalidValueException {
63.             // e.g. validate the given value against the model
64.         }
65.     ================================================================
66.
67.
68.
69.         /**
70.          * Light my fire out to the disk. Implementations should override
 this for saving their current value to the editor.
71.          * Should not be called directly, use {@link #save()} instead.
72.          *
73.          * @param to The language to store for.
74.          * @throws InvalidValueException If the value is invalid for the
 editor.
75.          */
76.         @Override
77.         protected void save(final Language to) throws
 InvalidValueException {
78.             final String toSave = _jTextArea.getText();
79.
80.             if (toSave.length() > 0) {
81.                 Logging.logInfo("saving: " + toSave, LOGGER);
82.                 getEditorValue().set(to, toSave);
```

```
83.             } else {
84.                 Logging.logError("EditorValue for language " + to + " is
 null. Couldnt save text!", LOGGER);
85.             }
86.         }
87.     ---------------------------------------------------------------------
88.
89.
90.
91.         /**
92.          * Implementations should override this for adopting their value
 from the given language.
93.          *
94.          * @param from The language to adopt the value from.
95.          * @since 4.0
96.          */
97.         @Override
98.         protected void adopt(final Language from) {
99.
100.            final Object data = getEditorValue().get(from);
101.            //final FSEditorValue data = (FSEditorValue)
 getEditorValue().get(from);
102.            //final String data = (String) getEditorValue().get(from);
103.
104.            Logging.logDebug("### adopt(final Language from) [name: " +
 getName() + " from: " + from + " data: " + data + "]", LOGGER);
105.
106.            // Prevent possible NPE while previewing section FORM template
107.            if (data != null && _component != null) {
108.                _jTextArea.setText(String.valueOf(data));
109.            }
110.        }
111.
112.
113.        //--- GuiEditor ---//
114.    ---------------------------------------------------------------------
115.
116.
117.
118.        /**
119.         * Implementations should override this method to enable/disable
 the gui components
120.         */
```

```
121.      @Override
122.      public void setLock(final boolean lock) {
123.          super.setLock(lock);
124.          if (_component != null) {
125.              _jTextArea.setEnabled(lock);
126.          }
127.      }
128.
129.
130.      //--- private methods ---//
131.  ================================================================
132.
133.
134.
135.      /**
136.       * Get the editor component.
137.       *
138.       * @return The editor's editor component.
139.       */
140.      public JComponent getEditorComponent() {
141.          if (_component == null) {
142.
143.              // Update gui values from EditorValue object
144.              undo();
145.
146.              // usefull for loading resource bundles
147.              //_bundle = new
 ExtendedResourceBundle(PropertyResourceBundle.getBundle(FSEditorResources.B
 UNDLE_KEY, getHost().getResourceBundle().getLocale()));
148.
149.              // init the component base frame
150.              _component = getFrame(initGui(), 500, true);
151.          }
152.          return _component;
153.      }
154.  ================================================================
155.
156.
157.
158.      /**
159.       * Initiate the components mainpanel.
160.       *
161.       * @return Value for property '_fsEditorPanel'.
```

```
162.         */
163.        JPanel initGui() {
164.            if (_fsEditorPanel == null) {
165.                final JTextArea jTextArea = getJTextArea();
166.
167.                // Listen for component changes and notify
 AccessChildTreeModel.StoreListener
168.                jTextArea.getDocument().addDocumentListener(new
 DocumentListener() {
169.
170.                    public void insertUpdate(final DocumentEvent e) {
171.                        Logging.logDebug("insertUpdate()", LOGGER);
172.
173.                        // do something
174.                        // ...
175.                        setChanged(true);
176.                    }
177.
178.
179.                    public void removeUpdate(final DocumentEvent e) {
180.                        Logging.logDebug("removeUpdate()", LOGGER);
181.
182.                        // do something
183.                        // ...
184.                        setChanged(true);
185.                    }
186.
187.
188.                    public void changedUpdate(final DocumentEvent e) {
189.                        Logging.logDebug("changedUpdate()", LOGGER);
190.
191.                        // do something
192.                        // ...
193.                        setChanged(true);
194.                    }
195.
196.                });
197.
198.                _fsEditorPanel = new JPanel(new BorderLayout());
199.                _fsEditorPanel.add(jTextArea, BorderLayout.CENTER);
200.                Logging.logDebug("initGui()... done.", LOGGER);
201.            }
202.            return _fsEditorPanel;
```

```
203.       }
204.   ═══════════════════════════════════════════════════════
205.
206.
207.
208.       /**
209.        * Getter for property 'JTextArea'.
210.        *
211.        * @return Value for property 'JTextArea'.
212.        */
213.       JTextArea getJTextArea() {
214.           if (_jTextArea == null) {
215.               final GomFSEditorContent form =
 getEditorValue().getForm();
216.
217.               _jTextArea = new JTextArea(form.getMaxRows().intValue(),
 1);
218.               _jTextArea.setAutoscrolls(true);
219.               _jTextArea.setEditable(true);
220.               _jTextArea.setDragEnabled(true);
221.               _jTextArea.setOpaque(true);
222.           }
223.           return _jTextArea;
224.       }
225.
226.   }
```

Listing 27: de.espirit.firstspirit.opt.example.editor.simple.FSEditorContentGuiEditor

3.11.4 Nützliche Interfaces für Editor-Komponenten

Die Implementierung in diesem Bereich ist noch nicht vollständig abgeschlossen. Die Dokumentation erfolgt sobald wie möglich.

3.12 Komponentenlose Modul-Implementierung (JDBC-Connector-Modul)

Um beispielsweise einen MySQL-JDBC-Treiber zur Nutzung innerhalb von FirstSpirit einzubinden oder zu aktualisieren, ist es erforderlich, den FirstSpirit-Server herunterzufahren bzw. einen Neustart durchzuführen. Besteht außerdem die Anforderung, z.B. zwei verschiedene MySQL-JDBC-Treiber für unterschiedliche Datenbankserver mit abweichenden Treiber-Versionen parallel zu betreiben, wird immer der erste von der JVM gefundene Treiber geladen (System-Classloader, siehe auch Kapitel 2.10 Seite 21).

Für den ersten der beiden zuvor beschriebenen Anwendungsfälle bietet sich die Implementierung einer Bibliothek-Modul-Komponente an (siehe Kapitel 3.13 Seite 91) oder, wie in diesem Kapitel beschrieben, als komponentenlose Modul-Umsetzung, wobei der MySQL-Connector hier als Modul-Ressource außerhalb des `<components>`-Elements im Modul-Descriptor definiert wird und somit über den FirstSpirit Modul-Classloader erreichbar ist und nicht im VM-System-Classloader landet (siehe auch Kapitel 2.10 ‚Classloading' Seite 21). Dadurch ergibt sich die Möglichkeit, mehrere Module (bspw. Module MySQL3 und Module MySQL5) mit unterschiedlichen Treiberversionen zu erstellen und diese anschließend parallel zu betreiben, wie im zweiten Anwendungsfall beschrieben. Grundlegend gilt: die Modul-Ressourcen sind systemweit verfügbar. D.h., sie werden in ein temporäres Verzeichnis (eines pro Modul) entpackt und sind dann über den Server-ClassLoader erreichbar. Das führt dazu, dass bei Namensgleichheit von Dateien das erste Modul/Datei/Klasse gewinnt. Das ist in der VM exakt genau so: sind mehrere Jars im ClassPath vorhanden, gewinnt immer die erste gefundene Ressource.

Vorteile der Integration eines Datenbank-Treibers als Modul-Implementierung:

1. Der MySQL-JDBC Treiber kann im laufenden Betrieb des FirstSpirit-Servers nachgeladen werden, der Server muss somit nicht heruntergefahren/neugestartet werden.

2. Es können Treiber in unterschiedlichen Versionen vorliegen und parallel zur Verwendung kommen.

Die Beispiele, die eine JDBC-Modul-Library verwenden, werden aus lizenztechnischen Gründen ohne die entsprechende JDBC-Bibliothek ausgeliefert. Diese muss vom jeweiligen Distributor heruntergeladen und im lib-Verzeichnis des Moduls abgelegt werden. Dazu muss die txt-Datei, die als Platzhalter dient, durch die entsprechende Bibliothek ersetzt werden, z.B. „mysql-connector-java-3.1.14-bin.jar.txt“ im Verzeichnis Library/MySQL03/lib durch „mysql-connector-java-3.1.14-bin.jar.“

Um aus einem JDBC-Treiber eine FirstSpirit-Modul zu erstellen genügen wenige Schritte (hier am Beispiel von „mysql-connector-java-3.1.14-bin.jar "):

1. Erstellen des Modul-Descriptors – module.xml

```
1.    <!DOCTYPE module SYSTEM "http://www.FirstSpirit.de/module.dtd">
2.    <module>
3.      <name>MySQL</name>
4.      <version>3.1.14</version>
5.      <description>JDBC Driver for MySQL (MySQL Connector/J
 3.1.14)</description>
6.      <vendor>MySQL-AB</vendor>
7.      <resources>
8.        <resource>lib/mysql-connector-java-3.1.14-bin.jar</resource>
9.      </resources>
10.     <components/>
11.   </module>
```

Listing 28: Modul-Descriptor ohne Komponenten – module.xml

2. Anlegen der Modul-Verzeichnisstruktur

Abbildung 3-5: Anlegen der Verzeichnisstruktur für das JDBC-Library-Modul

3. Erstellen der Modul-Archiv-Datei

Entweder über das Kommando:

```
$ zip -r MySQL.fsm .
  adding: META-INF/ (stored 0%)
  adding: META-INF/module.xml (deflated 40%)
  adding: lib/ (stored 0%)
  adding: lib/mysql-connector-java-3.1.14-bin.jar (deflated 4%)
```

oder über die grafische Benutzeroberfläche des bevorzugten Zip-Werkzeugs. Sollte es nicht möglich sein, die aus dem Zip-Vorgang resultierende Datei direkt als *.fsm - MySQL.fsm zu deklarieren, kann die `MySQL.zip`-Datei einfach umbenannt werden in `MySQL.fsm`.

4. Das neu erstellte Modul über die FirstSpirit-Anwendung zur Server- und Projektkonfiguration installieren

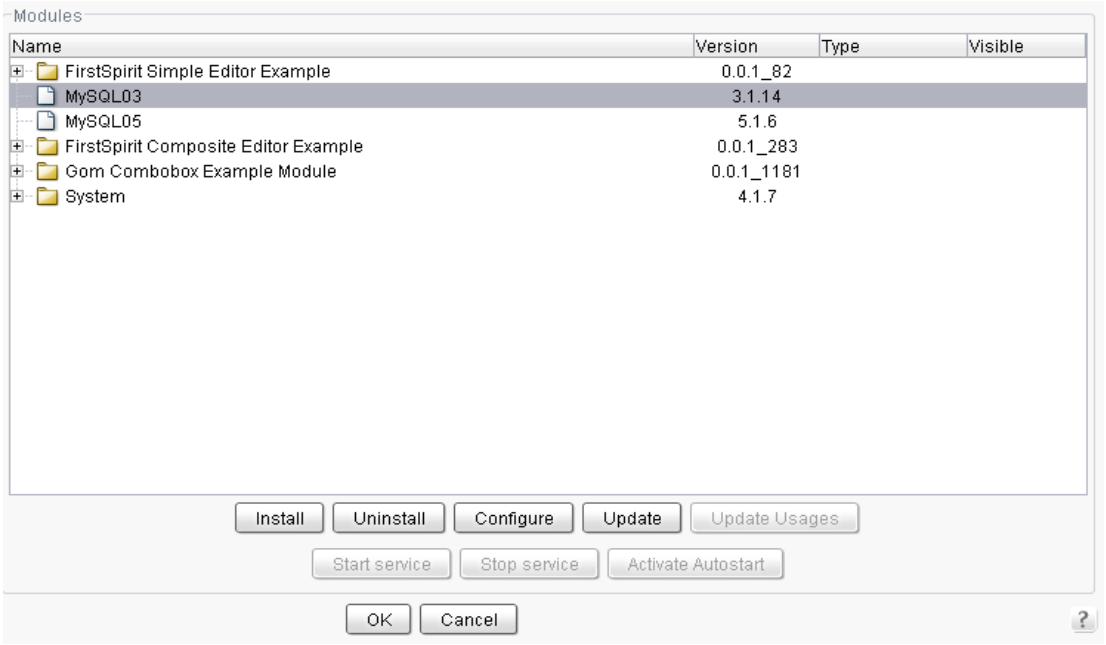


Abbildung 3-6: Komponentenloses Modul – Installation

5. Verwendung des Modul-Treibers

In den Layer-Eigenschaften der entsprechenden Datenbank den Eintrag `module=MySQL03` (Name siehe `module.xml`, Element **<name>***MySQL*</name>**) ergänzen:

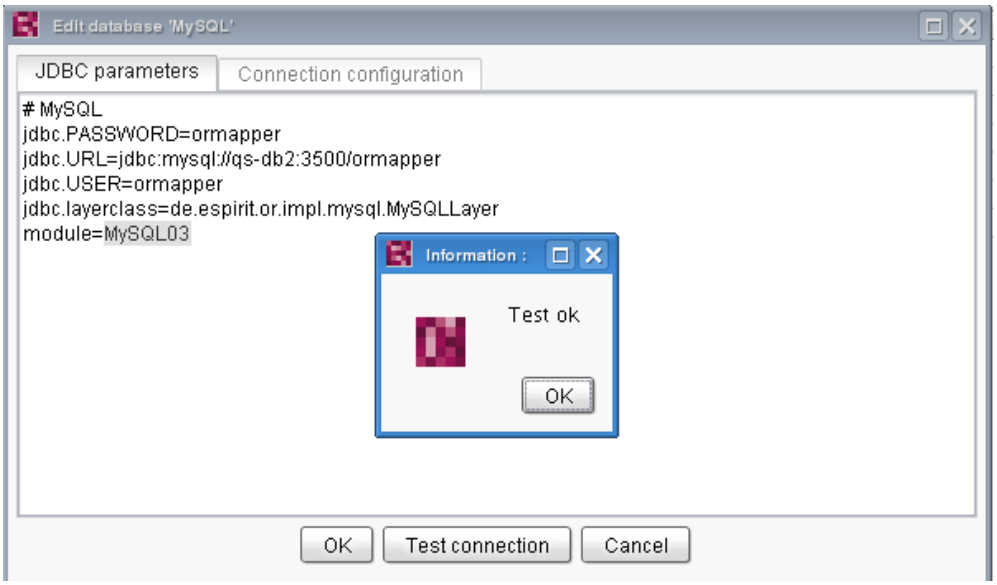


Abbildung 3-7: Verwendung eines Modul-JDBC-Treibers (MySQL) – mysql-connector-java-3.1.14-bin.jar

Um die mysql-connector Version 5 zu nutzen, ist in den Layer-Eigenschaften der entsprechende Datenbank der Eintrag `module=MySQL03` auf `module=MySQL05` zu ändern:

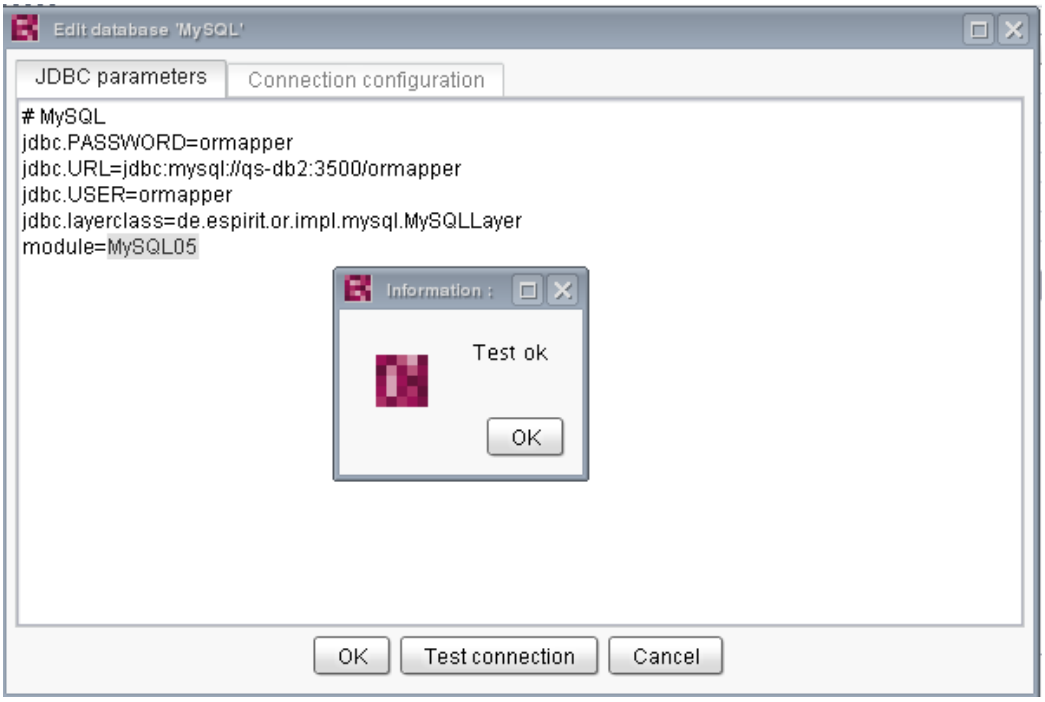


Abbildung 3-8: Verwendung eines Modul-JDBC-Treibers (MySQL) – mysql-connector-java-3.1.14-bin.jar

3.13 Implementierung einer Bibliothek- (Library) Komponente (JDBC-Treiber-Modul)

Hier die gleiche Implementierung wie zuvor unter ‚Komponentenlose Modul-Implementierung (JDBC-Connector-Modul)' beschrieben, nur als FirstSpirit-Modul Library-Komponente:

1. Erstellen des Modul-Descriptors – module.xml

```
1.    <!DOCTYPE module SYSTEM "http://www.FirstSpirit.de/module.dtd">
2.    <module>
3.        <name>MySQL</name>
4.        <version>0.0.1</version>
5.        <description>JDBC Driver for MySQL</description>
6.        <vendor>MySQL-AB</vendor>
7.        <components>
8.        <library>
9.            <name>JDBC Driver for MySQL - Connector/J</name>
10.           <version>3.1.14</version>
11.           <description>JDBC Driver for MySQL (MySQL Connector/J
 3.1.14)</description>
12.           <resources>
13.               <resource>lib/mysql-connector-java-3.1.14-
 bin.jar</resource>
14.           </resources>
15.       </library>
16.       </components>
17.   </module>
```

Listing 29: Modul-Descriptor mit Library-Komponente – module.xml

2., 3., 5. Die weiteren Schritte sind identisch zu Kapitel 3.12 Seite 87.

4. Das neu erstellte Modul über die FirstSpirit-Anwendung zur Server- und Projektkonfiguration installieren

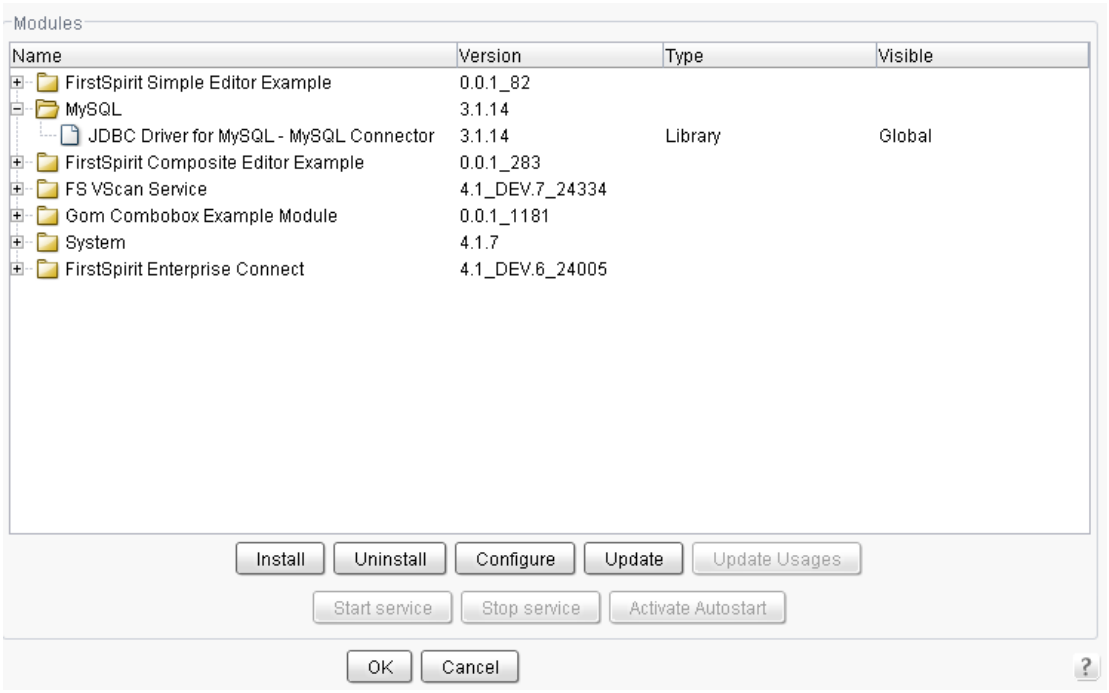


Abbildung 3-9: Installierte JDBC-Modul-Treiber-Bibliothek

Die in diesem Abschnitt beschriebene Variante sollte immer die bevorzugte Art der Implementierung sein.

3.14 Modul-Implementierung mit den Komponenten-Typen – PUBLIC, SERVICE, LIBRARY

Dieses Kapitel soll die Integration eines FirstSpirit Virenscanner-Moduls mittels der FirstSpirit Komponenten-Typen Service, Public und Library anhand einer Beispiel-Implementierung aufzeigen. Die voraussichtlich benötigte Infrastruktur für eine Virenscanner-Implementierung wurde mit dem VScan-Modul geschaffen, sowohl auf Modul-Ebene als auch im FirstSpirit-Server. Die initiale Proof-of-Concept-Implementierung nutzt hierbei einen lokal auf dem FirstSpirit-Server-System installierten Virenscanner (ClamAV / Linux). Dieser wird über einen ProcessBuilder angesprochen, was nur als Beispiel und zur Evaluierung des Konzepts dient. Ein ProcessBuilder ist im Produktionsbetrieb **nicht** geeignet für die Ausführung nativer Virenscanner-Anwendungen. Vielmehr sollte hier ein entfernter Service über Sockets, Pipes oder http (z.B. apache-commons mit Kapselung von ICAP) genutzt werden.

Die bisherige Umsetzung bietet ein `Interface ScanEngine`, welches von der konkreten Virenscanner-Engine implementiert werden muss. Daraus leitet sich der Komponenten-Typ PUBLIC für jede konkrete Virenscanner-Implementierung ab.

3.14.1 Modul-Komponenten und -Konfiguration

Das „VScan“-Modul beinhaltet mehrere Komponenten-Typen, wobei jede dieser Komponenten wiederum Ressourcen beinhaltet.

Abbildung 3-10: Komponenten-Typen des VScan-Moduls in der FirstSpirit Server- und Projektkonfiguration

Jede konkrete Implementierung einer Scanning-Engine enthält aus Modulsicht immer eine **PUBLIC**- und eine **LIBRARY**-Komponente. Wobei die PUBLIC-Komponente in Abbildung 3-10 ClamScanEngine.class das public Interface ScanEngine implementiert. Die Library ist das Archiv der Engine, welches weitere Klassen und Ressourcen enthält.

Die VScanService-Komponente ist konfigurierbar, d.h. für den Komponenten-Descriptor, das Attribut `<configurable/>` muss mit der Klasse für die grafische Konfigurationsoberfläche definiert werden. Die Definition im Modul- bzw. im

Komponenten-Descriptor stellt sich wie folgt dar (siehe auch Kapitel 3.7.1.3 Seite 38):

```
<configurable>de.espirit.firstspirit.opt.vscan.admin.gui.VscanServ
iceConfigPanel</configurable>
```

Die definierte Klasse muss wiederum das typisierte Interface Configuration implementieren:

```
public class VScanServiceConfigPanel implements
de.espirit.firstspirit.module.Configuration<ServerEnvironment>
```

Die folgende Abbildung zeigt die Service-Konfigurationsoberfläche in der FirstSpirit Server- und Projektkonfiguration:

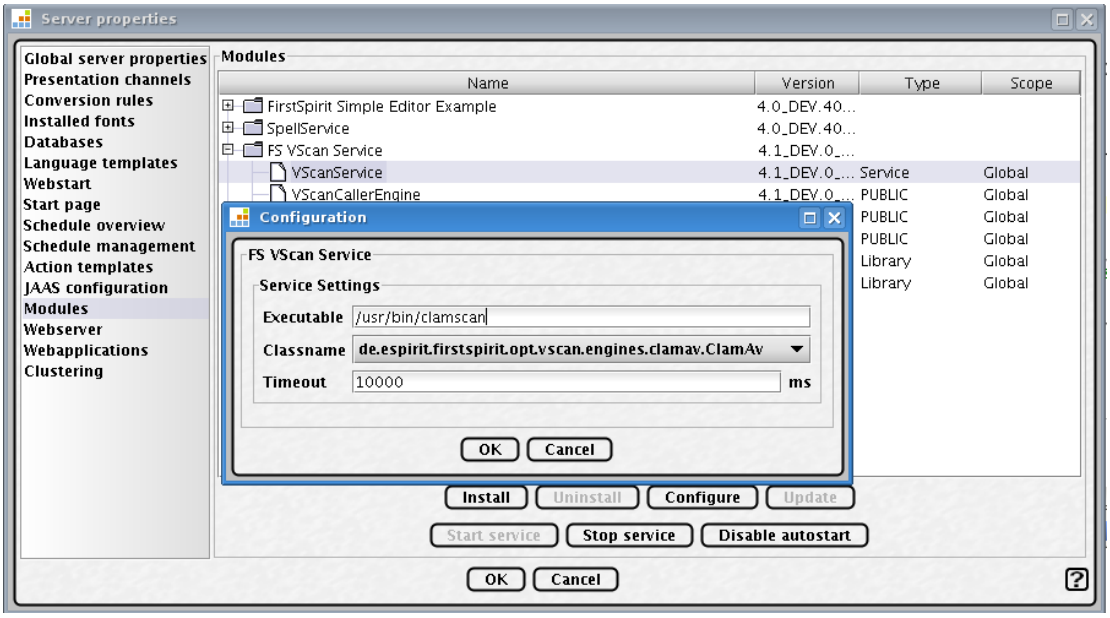


Abbildung 3-11: Service-Konfigurationsoberfläche in der FirstSpirit Server- und Projektkonfiguration

Die ComboBox **Classname** listet alle konkreten Scanning-Engines auf, die das ScanEngine Interface (HotSpot) implementieren.

3.14.2 Schematische Darstellung des Moduls innerhalb von FirstSpirit

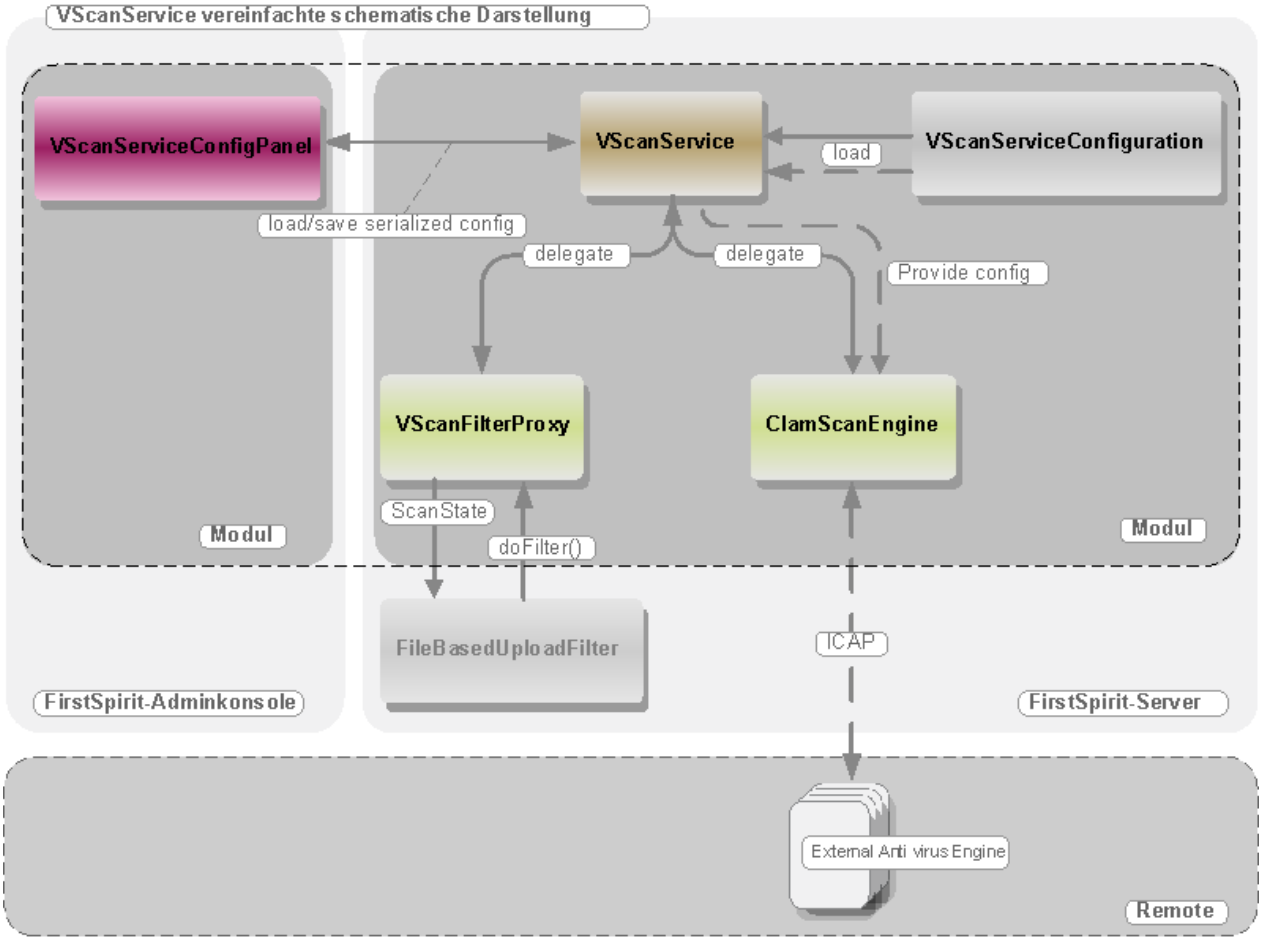


Abbildung 3-12: Kapselung von Modul-Komponenten innerhalb der FirstSpirit Server- und Projektkonfiguration sowie Kommunikationswege der Komponenten und Anbindung an die externe ICAP-Anwendung

3.14.3 Der VScan-Module-Descriptor

Konkret heißt das für das VScan-Modul, es werden drei verschiedene Komponenten-Typen genutzt:

- Über den **`Service`** Komponenten-Typ lässt sich der Virenscanner-Service global (serverweit) starten, stoppen und konfigurieren.
- Der **`Library`** Komponenten-Typ wird hier genutzt, um die verschiedenen Scanner-Engines dem System bekannt zu machen.
- Der **`Public`** Komponenten-Typ implementiert das Interface ScanEngine des VScan-Moduls. Die Descriptor-Datei „module.xml“ befindet sich im Wurzelverzeichnis des Modul-Sourcecodes. Generell besteht hier auch die

Möglichkeit, weitere Scanning-Engines als abhängiges Modul zu implementieren und nicht als weitere Komponente eines bestehenden Moduls. Wird ein neues Modul definiert, muss dieses auch wieder die Public- sowie die Library-Komponente enthalten, darüber hinaus muss eine Abhängigkeit zum VScan-Modul im Modul-Descriptor (module.xml) definiert werden.

```
<dependencies>
    <depends>VScanService</depends>
</dependencies>
```

3.14.4 Vollständiger Modul-Descriptor mit Drei Komponenten-Typen

```
1.    <!DOCTYPE module SYSTEM "../../server/module.dtd">
2.    <module>
3.          <name>FS VScan Service</name>
4.          <version>@VERSION@</version>
5.          <description>Plugable Virus Scanning Service</description>
6.          <vendor>e-Spirit AG</vendor>
7.          <license>...</license>
8.          <components>
9.                <service>
10.                     <name>VScanService</name>
11.                     <description>FirstSpirit Virus Scan
 Service</description>
12.
      <class>de.espirit.firstspirit.opt.vscan.VScanServiceImpl</class>
13.
      <configurable>de.espirit.firstspirit.opt.vscan.admin.gui.VScanServiceC
 onfigPanel</configurable>
14.                     <resources>
15.                           <resource name="libvscan">lib/fs-
 vscan.jar</resource>
16.                           <resource>fs-vscan.conf</resource>
17.                     </resources>
18.               </service>
19.               <public>
20.                     <name>VScanFilterProxy</name>
21.                     <description>The main engine which calls the
 specialized engine implementations.</description>
22.
      <class>de.espirit.firstspirit.opt.vscan.VScanFilterProxy</class>
23.                     <hidden>true</hidden>
```

```
24.                         <dependencies>
25.                             <depends>VScanService</depends>
26.                         </dependencies>
27.                 </public>
28.                 <!-- Scanning Engines Listing -->
29.                 <public>
30.                         <name>ClamAvEngine</name>
31.                         <description>ClamAv core class implementing the
 AvEngine interface</description>
32.
      <class>de.espirit.firstspirit.opt.vscan.engines.clamav.ClamScanEngine<
 /class>
33.                         <dependencies>
34.                             <depends>libclamav</depends>
35.                             <depends>VScanCallerEngine</depends>
36.                         </dependencies>
37.                 </public>
38.                 <library>
39.                         <name>libclamav</name>
40.                         <description>FS VScan Library containing the
 ClamAv virus scanning engine</description>
41.                         <hidden>true</hidden>
42.                         <scope>server</scope>
43.                         <resources>
44.                             <resource name="libclamav">lib/engines/fs-
 clamav.jar</resource>
45.                         </resources>
46.                         <dependencies>
47.                             <depends>ClamAvEngine</depends>
48.                         </dependencies>
49.                 </library>
50.         </components>
51.     </module>
```

Listing 30: Drei Komponenten-Typen Modul-Descriptor

3.14.5 Implementierung der SERVICE Basis-Komponente

Interface: VScanService
Package: de.espirit.firstspirit.opt.vscan
Definition der grundlegenden Service-Eigenschaften, wie z.B. Status-Codes, Laden der Konfiguration.

Interface 1: de.espirit.firstspirit.opt.vscan:VScanService

Klasse: VscanServiceImpl
Package: de.espirit.firstspirit.opt.vscan
Konkrete Implementierung der Service-Komponente. Start-, Stopp-Behandlung, Reagieren auf Ereignisse wie Installation, Aktualisierung, Deinstallation. Als Beispiel ist hier das Kopieren der Konfigurationsdatei `fs-vscan.conf` bei der Installation des Moduls
(`conf/modules/FS VScan Service.VScanService`) zu nennen.

public class VScanServiceImpl **implements** VScanService,
de.espirit.firstspirit.module.Service<VScanService>

Class 6: de.espirit.firstspirit.opt.vscan.VScanServiceImpl

```
1.    package de.espirit.firstspirit.opt.vscan;
2.
3.    import de.espirit.common.base.Logging;
4.    import de.espirit.common.io.IoUtil;
5.    import de.espirit.firstspirit.io.FileHandle;
6.    import de.espirit.firstspirit.io.ServerConnection;
7.
8.    import de.espirit.firstspirit.module.ServerEnvironment;
9.    import de.espirit.firstspirit.module.Service;
10.   import de.espirit.firstspirit.module.ServiceProxy;
11.   import de.espirit.firstspirit.module.descriptor.ServiceDescriptor;
12.   import org.jetbrains.annotations.NotNull;
13.
14.   import java.io.ByteArrayInputStream;
15.   import java.io.File;
16.   import java.io.IOException;
17.   import java.io.InputStream;
18.   import java.io.OutputStream;
19.   import java.io.UnsupportedEncodingException;
```

```
20.    import java.util.ArrayList;
21.    import java.util.Collections;
22.    import java.util.List;
23.    import java.util.Map;
24.    import java.util.Map.Entry;
25.
26.
27.    /**
28.     * $Date: 2008-06-02 17:25:13 +0200 (Mo, 02 Jun 2008) $
29.     *
30.     * @version $Revision: 22746 $
31.     */
32.    public class VScanServiceImpl implements VScanService,
 Service<VScanService> {
33.
34.        private static final Class<?> LOGGER = VScanServiceImpl.class;
35.        public static final String MODULE_CONFIG_FILE = "fs-vscan.conf";//
 NON-NLS
36.        public static final String MODULE_LOG_FILE = MODULE_NAME +
 ".log";// NON-NLS
37.
38.        private ServerEnvironment _environment;
39.        private volatile boolean _running;
40.        private VScanServiceConfiguration _config;
41.        private FileHandle _configFile;
42.    ------------------------------------------------------------------
43.
44.
45.
46.        public VScanServiceImpl() {
47.            _config = new VScanServiceConfiguration();
48.        }
49.    ------------------------------------------------------------------
50.
51.
52.
53.        /** Starts the service. */
54.        public void start() {
55.            Logging.logInfo("Starting " + VScanService.MODULE_SERVICE_NAME
 + "...", LOGGER);// NON-NLS
56.            appendLog("Starting...");// NON-NLS
57.            loadConfiguration();
58.            _running = true;
```

```
59.         }
60.     ================================================================
61.
62.
63.
64.         /** Stops the service. */
65.         public void stop() {
66.             Logging.logInfo("Shutting down " +
 VScanService.MODULE_SERVICE_NAME + "...", LOGGER);// NON-NLS
67.             appendLog("Shutting down...");// NON-NLS
68.             _running = false;
69.         }
70.     ================================================================
71.
72.
73.
74.         /**
75.          * Returns whether the service is running.
76.          *
77.          * @return <code>true</code> if the service is running.
78.          */
79.         public boolean isRunning() {
80.             return _running;
81.         }
82.     ================================================================
83.
84.
85.
86.         /**
87.          * Returns the service interface. Only methods of this interface
 are accessible, so <code>Service</code> instances must
88.          * also implement this interface.
89.          *
90.          * @return service interface class.
91.          */
92.         public Class<? extends VScanService> getServiceInterface() {
93.             return VScanService.class;
94.         }
95.     ================================================================
96.
97.
98.
99.         /**
```

```
100.        * A service proxy is an optional, client-side service
 implementation. It will be instantiated, {@link
101.        * de.espirit.firstspirit.module.ServiceProxy#init(Object,
 de.espirit.firstspirit.access.Connection) initialized} and
102.        * returned by {@link
 de.espirit.firstspirit.access.Connection#getService(String)}. The proxy
 class must have a no-arg
103.        * constructor and must implement the {@link
 de.espirit.firstspirit.module.ServiceProxy} interface, but has not to
104.        * implement the {@link #getServiceInterface() service-interface}
 itself.
105.        *
106.        * @return service proxy class or <code>null</code> if no proxy is
 provided.
107.        */
108.       public Class<? extends ServiceProxy> getProxyClass() {
109.           return null;
110.       }
111.   ------------------------------------------------------------------------
112.
113.
114.
115.       /**
116.        * Initializes this component with the given {@link
 de.espirit.firstspirit.module.descriptor.ComponentDescriptor
117.        * descriptor} and {@link
 de.espirit.firstspirit.module.ServerEnvironment environment}. No other
 method will be called
118.        * before the component is initialized!
119.        *
120.        * @param descriptor useful descriptor information for this
 component.
121.        * @param env       useful environment information for this
 component.
122.        */
123.       public void init(final ServiceDescriptor descriptor, final
 ServerEnvironment env) {
124.           _environment = env;
125.       }
126.   ------------------------------------------------------------------------
127.
128.
129.
```

```
130.       /** Event method: called if Component was successfully installed
 (not updated!) */
131.       public void installed() {
132.           copyConfigFile(getConfigFileName());// NON-NLS
133.           Logging.logDebug(getName() + " installed...", LOGGER);// NON-
 NLS
134.           appendLog("installed");// NON-NLS
135.       }
136.   ═══════════════════════════════════════════════════════════
137.
138.
139.
140.       /** Event method: called if Component was in uninstalling
 procedure */
141.       public void uninstalling() {
142.           Logging.logDebug(getName() + " uinstalling...", LOGGER);//
 NON-NLS
143.       }
144.   ═══════════════════════════════════════════════════════════
145.
146.
147.
148.       /**
149.        * Event method: called if Component was completely updated
150.        *
151.        * @param oldVersionString old version, before component was
 updated
152.        */
153.       public void updated(final String oldVersionString) {
154.           // e.g. merge, diff, bkup the config file or do not overwrite
155.           copyConfigFile(getConfigFileName());// NON-NLS
156.           Logging.logDebug(getName() + " updated...", LOGGER);// NON-NLS
157.           appendLog("updated");// NON-NLS
158.       }
159.   ═══════════════════════════════════════════════════════════
160.
161.
162.
163.       /**
164.        * Copy a config file from the package dir to module config dir ->
 conf/modules/$module_name$.$module_service_name$
165.        *
166.        * @param file the module config file name string
```

```
167.         */
168.        private void copyConfigFile(final String file) {
169.            try {
170.                final FileHandle configFile =
 _environment.getConfDir().obtain(file);
171.
172.                if (configFile.exists()) {
173.                    Logging.logDebug("config file " + configFile.getPath()
 + " exists, do not overwrite", LOGGER);// NON-NLS
174.                } else {
175.                    final InputStream is =
 getClass().getResourceAsStream(file);
176.                    if (is != null) {
177.                        try {
178.                            configFile.save(is);
179.                        } finally {
180.                            IoUtil.saveClose(is);
181.                        }
182.                    }
183.                }
184.            } catch (final IOException e) {
185.                Logging.logError("Error saving file - for " + getName() +
 " compoment", e, LOGGER);// NON-NLS
186.            }
187.        }
188.    ---------------------------------------------------------------------
189.
190.
191.
192.        private void appendLog(final String message) {
193.            final StringBuilder sb = new StringBuilder();
194.            sb.append(Logging.getDateString());
195.            sb.append('\t');
196.            sb.append(message);
197.            sb.append('\n');
198.
199.            InputStream is = null;
200.            try {
201.                //final FileHandle handle =
 _environment.getConfDir().obtain(MODULE_LOG_FILE);
202.                final FileHandle handle =
 _environment.getLogDir().obtain(MODULE_LOG_FILE);
203.                is = new ByteArrayInputStream(sb.toString().getBytes("UTF-
```

```
8"));// NON-NLS
                handle.append(is);
            } catch (final UnsupportedEncodingException e) {
                throw new IllegalStateException(e);// should not happen
            } catch (final IOException e) {
                Logging.logError("Couldn't obtain file: " +
MODULE_LOG_FILE, e, LOGGER);// NON-NLS
            } finally {
                IoUtil.saveClose(is);
            }
        }
    ------------------------------------------------------------------------
```

```
        @SuppressWarnings({"UnusedCatchParameter"})
        private ScanEngine getScanEngine(@NotNull final String className)
throws ClassNotFoundException {
            // replace single call to loadAvEngine by a pre loaded engine
pool on service start
            try {
                final Class<?> clazz =
getClass().getClassLoader().loadClass(className);
                Logging.logInfo("Loading Anti Virus Scanning Engine: " +
className + " ...", LOGGER);// NON-NLS
                if (!ScanEngine.class.isAssignableFrom(clazz)) {
                    throw new IllegalStateException(clazz + " does not
implement " + ScanEngine.class.getName());
                }
                final ScanEngine result = (ScanEngine)
clazz.newInstance();
                result.init(_config);
                if (result.isThreadSafe()) {
                    // todo: pooling
                }
                return result;
            } catch (InstantiationException e) {
                throw new InstantiationError("Could not instantiate virus
scan engine " + className);
            } catch (IllegalAccessException e) {
                throw new IllegalStateException("Access denied - class: "
+ className);
```

```
237.             }
238.         }
239.     ----------------------------------------------------------------------
240.
241.
242.
243.         public void scanFile(@NotNull final File tempFile) throws
 ClassNotFoundException, UploadRejectedException {
244.             try {
245.                 final ScanEngine scanEngine =
 getScanEngine(_config.getClassName());
246.                 appendLog("scanning " + tempFile + " with " +
 scanEngine.getName());// NON-NLS
247.                 scanEngine.scanFile(tempFile);
248.                 appendLog("file " + tempFile + " ok");// NON-NLS
249.             } catch (final UploadRejectedException e) {
250.                 appendLog("file " + tempFile + " not ok - " +
 e.getMessage());// NON-NLS
251.                 throw e;
252.             } catch (final ClassNotFoundException e) {
253.                 Logging.logError("Loading of virus scanning engine failed:
 " + _config.getClassName(), e, LOGGER);// NON-NLS
254.                 throw new ClassNotFoundException("Loading of virus
 scanning engine failed: " + _config.getClassName(), e);
255.             }
256.         }
257.
258.
259.         // VScanServiceConfiguration
260.     ----------------------------------------------------------------------
261.
262.
263.
264.         public void loadConfiguration() {
265.             InputStream is = null;
266.
267.             _config.setEnvironment(_environment);
268.
269.             try {
270.                 _configFile =
 _environment.getConfDir().obtain(MODULE_CONFIG_FILE);
271.                 is = _configFile.load();
272.
```

```
273.                if (_configFile.isFile()) {
274.                    _config.getServiceProperties().load(is);
275.                    _config.init();
276.                    _config.setAvailableEngines(getAvailableEngines());
277.                }
278.            } catch (IOException e) {
279.                Logging.logDebug("Couldn't load config file - " +
 _configFile.getPath(), e, LOGGER);// NON-NLS
280.            } finally {
281.                try {
282.                    if (is != null) {
283.                        is.close();
284.                    }
285.                } catch (IOException e) {
286.                    Logging.logDebug("Couldn't close config file - " +
 _configFile.getPath(), e, LOGGER);// NON-NLS
287.                }
288.            }
289.        }
290.    ================================================================
291.
292.
293.
294.        public void saveConfiguration() {
295.            OutputStream os = null;
296.
297.            try {
298.                if (_configFile.exists() && _configFile.isFile()) {
299.                    IoUtil.copyFile(_configFile.getPath(),
 _configFile.getPath() + ".1");
300.                }
301.
302.                os = _configFile.getOutputStream();
303.
304.
 _config.getServiceProperties().setProperty("fsm.vscan.engine.executable",
 _config.getExecutable().toString());
305.                Logging.logDebug("Setting property -
 fsm.vscan.engine.executable=" + _config.getExecutable().toString(),
 LOGGER);// NON-NLS
306.
307.
 _config.getServiceProperties().setProperty("fsm.vscan.engine.class",
```

```
_config.getClassName());
308.                Logging.logDebug("Setting property -
fsm.vscan.engine.class=" + _config.getClassName(), LOGGER);// NON-NLS
309.
310.
_config.getServiceProperties().setProperty("fsm.vscan.engine.timeout",
String.valueOf(_config.getTimeout()));
311.                Logging.logDebug("Setting property -
fsm.vscan.engine.timeout=" + String.valueOf(_config.getTimeout()),
LOGGER);// NON-NLS
312.
313.                _config.getServiceProperties().store(os, "Last
modified");// NON-NLS
314.
315.            } catch (IOException e) {
316.                Logging.logDebug("Couldn't open config file: " +
_configFile.getPath(), e, LOGGER);// NON-NLS
317.            } finally {
318.                try {
319.                    if (os != null) {
320.                        os.close();
321.                    }
322.                } catch (IOException e) {
323.                    Logging.logDebug("Closing of config file output stream
failed.", e, LOGGER);// NON-NLS
324.                }
325.            }
326.        }
327.    ----------------------------------------------------------------------
328.
329.
330.
331.        /** {@inheritDoc} */
332.        @NotNull
333.        public String getName() {
334.            return MODULE_NAME;
335.        }
336.    ----------------------------------------------------------------------
337.
338.
339.
340.        /**
341.         * Get the module configuration file name
```

```
342.         *
343.         * @return the config file name string
344.         */
345.        @NotNull
346.        private static String getConfigFileName() {
347.            return MODULE_CONFIG_FILE;
348.        }
349.    ------------------------------------------------------------------------
350.
351.
352.
353.        /** {@inheritDoc} */
354.        public boolean isEnabled() {
355.            return _running;
356.        }
357.    ------------------------------------------------------------------------
358.
359.
360.
361.        /** {@inheritDoc} */
362.        public VScanServiceConfiguration getServiceConfiguration() {
363.            loadConfiguration();
364.            return _config;
365.        }
366.    ------------------------------------------------------------------------
367.
368.
369.
370.        public void setServiceConfiguration(final
 VScanServiceConfiguration config) {
371.            _config = config;// todo: clear pool
372.            saveConfiguration();
373.        }
374.
375.    }
```

Listing 31: de.espirit.firstspirit.opt.vscan.VScanServiceImpl

Klasse: VScanServiceConfiguration
Package: de.espirit.firstspirit.opt.vscan

Modell für die notwendigen Konfigurations-Einstellungen des Services. Getter und Setter zu jeder Konfigurationseigenschaft. Lädt bei Initialisierung die persistenten Einstellungen aus der `fs-vscan.conf` Datei. Besitzt eine `save()`-Methode, um die geladenen oder in der Konfigurationsoberfläche geänderten Einstellungen in die Konfigurationsdatei zu schreiben. Jedes Key/Value-Paar in der `fs-vscan.conf` entspricht hier einer Member-Variable der Klasse und benötigt hierfür jeweils einen Getter und Setter. Die `init()`-Methode muss ggf. um die weiteren benötigten Konfigurationsoptionen erweitet werden. Laden und Speichern der Konfigurationsoptionen erfolgt über
VScanServiceImpl#saveConfiguration() und
VScanServiceImpl#loadConfiguration. Die Konfigurationsklasse implementiert hier das Interface Serializable, das diese vom serverseitigen Service zur FirstSpirit Server- und Projektkonfiguration übertragen wird. Der Service kontrolliert den Lifecycle der Konfiguration.

```
public class VScanServiceConfiguration implements Serializable
```

Class 7: de.espirit.firstspirit.opt.vscan.VScanServiceConfiguration

Klasse: VscanServiceConfigPanel
Package: de.espirit.firstspirit.opt.vscan

Die Service-Konfigurationsoberfläche, zu erreichen über die FirstSpirit Server- und Projekteigenschaften. Alle notwendigen Felder der `VscanServiceConfiguration` können hier geladen werden – die Methode `getGui()` initialisiert die Konfigurationsoberfläche, wobei die Methode hasGui() hartcodiert true zurückliefern muss.

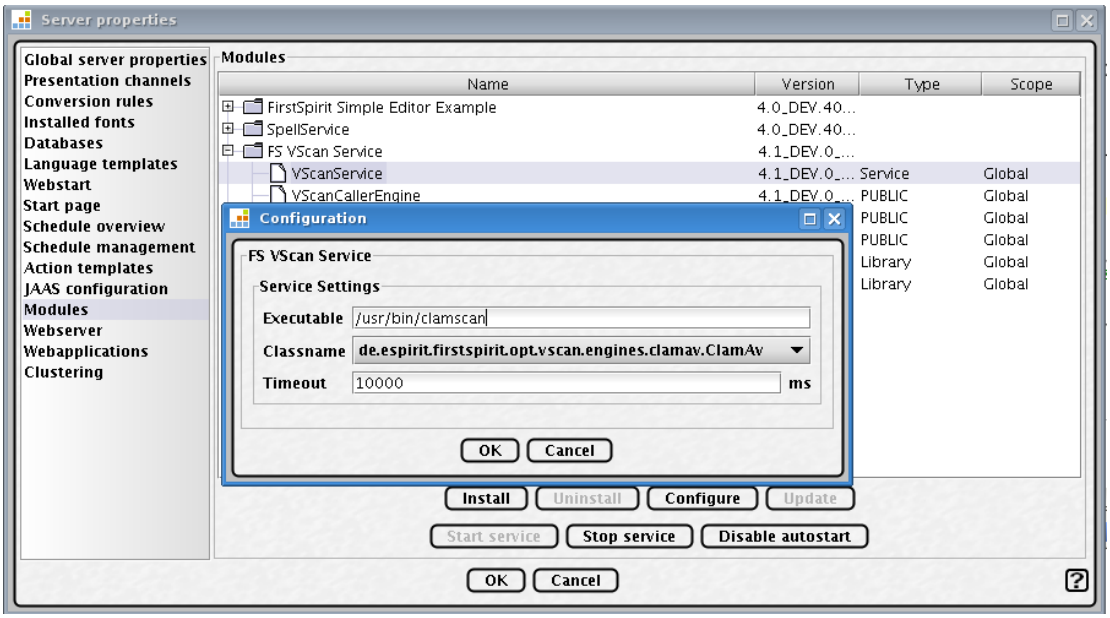


Abbildung 3-13: VscanServiceConfigPanel – Die Modul-Konfigurationsoberfläche

Class 8: de.espirit.firstspirit.opt.vscan.VScanServiceConfigPanel

```
1.  package de.espirit.firstspirit.opt.vscan.admin.gui;
2.
3.  import de.espirit.firstspirit.access.ServiceNotFoundException;
4.  import de.espirit.firstspirit.common.gui.CMSDialog;
5.  import de.espirit.firstspirit.io.ServerConnection;
6.
7.  import de.espirit.firstspirit.module.Configuration;
8.  import de.espirit.firstspirit.module.ServerEnvironment;
9.  import de.espirit.firstspirit.opt.vscan.VScanService;
10. import de.espirit.firstspirit.opt.vscan.VScanServiceConfiguration;
11. import de.espirit.firstspirit.opt.vscan.resources.ModuleResources;
12. import de.espirit.firstspirit.server.ManagerNotFoundException;
13. import info.clearthought.layout.TableLayout;
```

```
14.
15.    import javax.swing.BorderFactory;
16.    import javax.swing.JComboBox;
17.    import javax.swing.JComponent;
18.    import javax.swing.JLabel;
19.    import javax.swing.JPanel;
20.    import javax.swing.JTextField;
21.    import java.awt.Frame;
22.    import java.net.URI;
23.    import java.util.List;
24.    import java.util.Set;
25.
26.
27.    /**
28.     * $Date: 2008-06-02 17:25:13 +0200 (Mo, 02 Jun 2008) $
29.     *
30.     * @version $Revision: 22746 $
31.     */
32.    public class VScanServiceConfigPanel implements
 Configuration<ServerEnvironment> {
33.
34.        private VScanServiceConfiguration _config;
35.        private ServerEnvironment _env;
36.        private JPanel _component;
37.        private JTextField _engineExecutableField;
38.        private JTextField _engineTimeoutField;
39.        private JComboBox _enginesClassList;
40.        private List<String> _availableEngines;
41.    ------------------------------------------------------------
42.
43.
44.
45.        public VScanServiceConfigPanel() {
46.        }
47.    ------------------------------------------------------------
48.
49.
50.
51.        /**
52.         * Returns <code>true</code> if this component has a gui to show
 and change it's configuration.
53.         *
54.         * @return <code>true</code> if this component has a gui to show
```

```
 and change it's configuration.
55.          */
56.         public boolean hasGui() {
57.             return true;
58.         }
59.     ----------------------------------------------------------------
60.
61.
62.
63.         /**
64.          * Returns the configuration gui.
65.          *
66.          * @param masterFrame basic frame component, for creating new
 windows
67.          * @return configuration gui or <code>null</code>.
68.          */
69.         public JComponent getGui(final Frame masterFrame) {
70.             if (_component == null) {
71.
72.                 final double border = 5;
73.                 final double rowsGap = 5;
74.                 final double[][] size = {{border, TableLayout.FILL,
 border}, {border, TableLayout.PREFERRED, rowsGap, TableLayout.PREFERRED,
 rowsGap, TableLayout.PREFERRED, border}};
75.                 final TableLayout tbl = new TableLayout(size);
76.
77.                 final JPanel panel = new JPanel();
78.                 panel.setOpaque(false);
79.
panel.setBorder(BorderFactory.createTitledBorder(VScanService.MODULE_NAME))
 ;
80.                 panel.setLayout(tbl);
81.
82.                 panel.add(getEnginePanel(), "1, 1, 1, 1");
83.
84.                 _component = (JPanel) masterFrame.add(panel);
85.                 //_component = panel;
86.
87.                 clearGuiValues();
88.                 initGuiValues();
89.             }
90.
91.             return _component;
```

```
92.         }
93.     ------------------------------------------------------------------
94.
95.
96.
97.         private JComponent getEnginePanel() {
98.             final double border = 5;
99.             final double rowsGap = 5;
100.            final double colsGap = 5;
101.            final double[][] size = {{border, TableLayout.PREFERRED,
 colsGap, TableLayout.FILL, colsGap, TableLayout.MINIMUM, border}, {border,
 TableLayout.PREFERRED, rowsGap, TableLayout.PREFERRED, rowsGap,
 TableLayout.PREFERRED, border}};
102.            final TableLayout tbl = new TableLayout(size);
103.
104.            final JPanel enginePanel = new JPanel();
105.            enginePanel.setOpaque(false);
106.
 enginePanel.setBorder(BorderFactory.createTitledBorder(ModuleResources.getS
 tring("fs-
 resource.module.vscan.admin.gui.VScanServiceConfigPanel_EngineSetup")));
107.            enginePanel.setLayout(tbl);
108.
109.            enginePanel.add(new JLabel(ModuleResources.getString("fs-
 resource.module.vscan.admin.gui.VScanServiceConfigPanel_EngineExecutable"))
 , "1, 1, 1, 1");
110.            //enginePanel.add(new
 JLabel(ModuleResources.loadIconResource("de.espirit.firstspirit.opt.vscan.r
 esources.icons", "core.png")), "1, 1, 1, 1");// NON-NLS
111.            //enginePanel.add(new
 JLabel(ModuleResources.loadIconResource("core.png")), "1, 1, 1, 1");// NON-
 NLS
112.            _engineExecutableField = new JTextField();
113.            enginePanel.add(_engineExecutableField, "3, 1, 5, 1");
114.
115.
116.            _enginesClassList = new JComboBox(loadEnginesList());
117.            enginePanel.add(new JLabel(ModuleResources.getString("fs-
 resource.module.vscan.admin.gui.VScanServiceConfigPanel_EngineClassname")),
 "1, 3, 1, 3");
118.            enginePanel.add(_enginesClassList, "3, 3, 5, 3");
119.
120.
```

```
121.            enginePanel.add(new JLabel(ModuleResources.getString("fs-
 resource.module.vscan.admin.gui.VScanServiceConfigPanel_EngineTimeout")),
 "1, 5, 1, 5");
122.            _engineTimeoutField = new JTextField();
123.            enginePanel.add(_engineTimeoutField, "3, 5, 3, 5");
124.            enginePanel.add(new JLabel("ms"), "5, 5, 5, 5");
125.
126.            return enginePanel;
127.        }
128.    ================================================================
129.
130.
131.
132.        /**
133.         * Loads the current configuration from appropriate {@link
 de.espirit.firstspirit.module.ServerEnvironment#getConfDir()
134.         * conf directory}.
135.         */
136.        @SuppressWarnings({"UnusedCatchParameter"})
137.        public void load() {
138.            VScanService service = null;
139.
140.            try {
141.                service = (VScanService)
 _env.getConnection().getService(VScanService.MODULE_SERVICE_NAME);
142.                if (!service.isEnabled()) {
143.                    showServiceStartDialog();
144.                }
145.            } catch (IllegalStateException e) {
146.                showServiceStartDialog();
147.            } catch (ServiceNotFoundException e) {
148.                showServiceStartDialog();
149.            } catch (ManagerNotFoundException e) {
150.                showServiceStartDialog();
151.            }
152.
153.            if(service == null) {
154.                service = (VScanService)
 _env.getConnection().getService(VScanService.MODULE_SERVICE_NAME);
155.            }
156.
157.            _config = service.getServiceConfiguration();
158.            _config.setEnvironment(_env);
```

```
159.       }
160.   ------------------------------------------------------------------------
161.
162.
163.
164.
165.       /** Stores the current configuration. */
166.       public void store() {
167.           final VScanService service = (VScanService)
 _env.getConnection().getService(VScanService.MODULE_SERVICE_NAME);
168.
169.
 _config.setExecutable(URI.create(_engineExecutableField.getText().trim()));
170.
 _config.setClassName(_enginesClassList.getSelectedItem().toString());
171.
 _config.setTimeout(Long.valueOf(_engineTimeoutField.getText().trim()));
172.           //_config.save();
173.           service.setServiceConfiguration(_config);
174.       }
175.   ------------------------------------------------------------------------
176.
177.
178.
179.       private void clearGuiValues() {
180.           _engineExecutableField.setText("");
181.           _engineTimeoutField.setText("");
182.       }
183.   ------------------------------------------------------------------------
184.
185.
186.
187.       private void initGuiValues() {
188.
189.
 _engineExecutableField.setText(_config.getExecutable().toString().trim());
190.
 _engineTimeoutField.setText(String.valueOf(_config.getTimeout()).trim());
191.
192.           for (final Object engineClasses : _availableEngines) {
193.               if
 (_config.getClassName().equals(engineClasses.toString())) {
194.
```

```
_enginesClassList.setSelectedItem(engineClasses.toString());
195.                }
196.            }
197.        }
198.    ----------------------------------------------------------------------
199.
200.
201.
202.        private Object[] loadEnginesList() {
203.            _availableEngines = _config.getAvailableEngines();
204.
205.            if (_availableEngines.isEmpty()) {
206.                _availableEngines.add(ModuleResources.getString("fs-
resource.module.vscan.admin.gui.VScanServiceConfigPanel_ErrorNoEngineAvaila
ble"));
207.            }
208.
209.            return _availableEngines.toArray();
210.        }
211.    ----------------------------------------------------------------------
212.
213.
214.
215.        /**
216.         * Returns all parameter names.
217.         *
218.         * @return all parameter names.
219.         */
220.        public Set<String> getParameterNames() {
221.            return _config.getParameterNames();
222.        }
223.    ----------------------------------------------------------------------
224.
225.
226.
227.        /**
228.         * Returns a specific parameter or <code>null</code> if it's not
available.
229.         *
230.         * @param name parameter name.
231.         * @return parameter or <code>null</code>.
232.         */
233.        public String getParameter(final String name) {
```

```
234.            return _config.getParameter(name);
235.        }
236.    ----------------------------------------------------------------------
237.
238.
239.
240.        /**
241.         * Initializes this component with the given {@link
 de.espirit.firstspirit.module.ServerEnvironment environment}. This
242.         * method is called before the instance is used.
243.         *
244.         * @param moduleName    module name
245.         * @param componentName component name
246.         * @param env           useful environment information for this
 component.
247.         */
248.        public void init(final String moduleName, final String
 componentName, final ServerEnvironment env) {
249.            _env = env;
250.        }
251.    ----------------------------------------------------------------------
252.
253.
254.
255.        /**
256.         * Returns ComponentEnvironment
257.         *
258.         * @return ComponentEnvironment
259.         */
260.        public ServerEnvironment getEnvironment() {
261.            return _env;
262.        }
263.    }
```

Listing 32: de.espirit.firstspirit.opt.vscan.VScanServiceConfigPanel

Interface: ScanEngine
Package: de.espirit.firstspirit.opt.vscan
Definition der grundlegenden Methoden, die von einer VirusScan-Engine zur Verfügung gestellt werden müssen (siehe auch `ClamScanEngine.java` und `ScanEngine`#JavaDoc), damit diese ohne weiteres in die bestehende Infrastruktur eingebunden werden kann.

Interface 2: de.espirit.firstspirit.opt.vscan.ScanEngine

3.14.6 Die PUBLIC-Komponente des Moduls

Klasse: VScanFilterProxy
Package: de.espirit.firstspirit.opt.vscan

Erweitert die Klasse `FileBasedUploadFilter`, welche wiederum `UploadFilter` implementiert. Daraus ergibt sich der schon zuvor angesprochene Komponenten-Typ `Public` (siehe auch Kapitel 2.9.1.7 Seite 19). Alle Aufrufe der Klasse werden an VscanServiceImpl delegiert.

Class 9: de.espirit.firstspirit.opt.vscan.VScanFilterProxy

```
1.  package de.espirit.firstspirit.opt.vscan;
2.
3.  import de.espirit.common.base.Logging;
4.  import
 de.espirit.firstspirit.server.mediamanagement.FileBasedUploadFilter;
5.
6.  import java.io.File;
7.  import java.io.IOException;
8.
9.
10. /**
11.  * $Date: 2008-05-26 13:52:11 +0200 (Mo, 26 Mai 2008) $
12.  *
13.  * @version $Revision: 22535 $
14.  */
15. public class VScanFilterProxy extends FileBasedUploadFilter {
16.
17.     public static final String ENGINE_NAME = "VScanFilterProxy";//
 NON-NLS
18.     public static final Class<?> LOGGER = VScanFilterProxy.class;
19. ------------------------------------------------------------------
20.
21.
22.
23.     public VScanFilterProxy() {
24.         // no arg constructor needed, called from super class
25.     }
26. ------------------------------------------------------------------
27.
28.
29.
```

```
30.         @Override
31.         public void init() {
32.             if (Logging.isDebugEnabled(LOGGER)) {
33.                 Logging.logDebug("VScanFilterProxy.init() - ", LOGGER);//
 NON-NLS
34.             }
35.         }
36.
37.
38.         // -- get VScanService, get VScanServiceConfiguration -- //
39.     ================================================================
40.
41.
42.
43.         /**
44.          * Get the VScanService by the serviceLocator
45.          *
46.          * @return the VScanService
47.          */
48.         private VScanService getVScanService() {
49.             return (VScanService)
 getServiceLocator().getService(VScanService.MODULE_SERVICE_NAME);
50.         }
51.
52.
53.         // -- Filter, Execute, Grant or Reject -- //
54.     ================================================================
55.
56.
57.
58.         /** {@inheritDoc} */
59.         @Override
60.         public void doFilter(final File tempFile) throws IOException {
61.             try {
62.                 getVScanService().scanFile(tempFile);
63.             } catch (ClassNotFoundException e) {
64.                 Logging.logInfo("Loading of virus scanning engine
 failed!", e, LOGGER);// NON-NLS
65.                 throw new IOException("Loading of virus scanning engine
 failed!");
66.             }
67.         }
68.
```

```
69.     }
```

Listing 33: de.espirit.firstspirit.opt.vscan.VScanFilterProxy

3.14.7 Die LIBRARY-Komponente(n) des Moduls

Klasse: ClamScanEngine
Package: de.espirit.firstspirit.opt.vscan.engines.clamav.ClamScanEngine
Die konkrete Implementierung des `ScanEngine`-Interfaces. Diese Klasse setzt alle Stati, enthält die Logik zur Kommunikation mit der externen Scan-Engine und extrahiert aus dem Reply die nötigen Informationen, um einen eindeutigen Status zu setzen (`throw new UploadRejectedException`). Anstelle dieser Implementierung der ClamScanEngine (/usr/bin/clamscan) kann man sich bspw. auch eine HTTP/ICAP-Implementierung vorstellen.

Class 10: de.espirit.firstspirit.opt.vscan.engines.clamav.ClamScanEngine

Prinzipiell muss für die Einbindung weiterer Scanning-Engines nur die nötige(n) Klasse(n) für die spezielle AntiVirus-Anwendungen implementiert werden. Die ClamScanEngine-Klasse soll zeigen, wie es z.B. auch möglich wäre, eine ICAP-Umsetzung zu schaffen, die mittels ICAP-Client mit einer entfernten ICAP-AntiVirus-Anwendung (Symantec, Kaspersky etc.) kommuniziert.
Hierzu ist eine Anpassung anderer Klassen oder Interfaces nicht notwendig.

z.B.:
- de.espirit.firstspirit.opt.vscan.engines.icap.IcapClient

Eine Implementierung, welche die FirstSpirit UploadFilter-API nutzt, muss eine konkrete UploadRejectedException werfen, falls ein Medium nicht den Restriktionen entspricht oder im Falle der ScanEngine ein Virus gefunden wurde.

```
1.     package de.espirit.firstspirit.opt.vscan.engines.clamav;
2.
3.     import de.espirit.common.base.Logging;
4.     import de.espirit.firstspirit.access.AccessUtil;
5.     import de.espirit.firstspirit.opt.vscan.ScanEngine;
6.     import
 de.espirit.firstspirit.access.store.mediastore.UploadRejectedException;
7.     import de.espirit.firstspirit.opt.vscan.VScanService;
8.     import de.espirit.firstspirit.opt.vscan.VScanServiceConfiguration;
```

```
9.
10.     import java.io.File;
11.     import java.io.IOException;
12.     import java.io.StringWriter;
13.     import java.net.URI;
14.
15.
16.     /**
17.      * $Date: 2008-05-26 13:52:11 +0200 (Mo, 26 Mai 2008) $
18.      *
19.      * @version $Revision: 22535 $
20.      */
21.     public class ClamScanEngine implements ScanEngine {
22.
23.         public static final Class<?> LOGGER = ClamScanEngine.class;
24.         public static final String ENGINE_NAME = "ClamAv";// NON-NLS
25.
26.         private URI _executable;
27.     ------------------------------------------------------------------------
28.
29.
30.
31.         public ClamScanEngine() {
32.             Logging.logDebug("ClamScanEngine created", LOGGER);// NON-NLS
33.         }
34.     ------------------------------------------------------------------------
35.
36.
37.
38.         public String getName() {
39.             return ENGINE_NAME;
40.         }
41.     ------------------------------------------------------------------------
42.
43.
44.
45.         public void init(final VScanServiceConfiguration configuration) {
46.             _executable = configuration.getExecutable();
47.             Logging.logDebug("Executable set to: " + _executable,
 LOGGER);// NON-NLS
48.             if ( ! executableExists()) {
49.                 Logging.logError(VScanService.MODULE_NAME + ": " +
 getName() + " executable '" + getExecutable() + "' not found.", LOGGER);//
```

```
NON-NLS
50.                 throw new IllegalStateException(VScanService.MODULE_NAME +
 ": " + getName() + " executable '" + getExecutable() + "' not found.");
51.             }
52.         }
53.     ----------------------------------------------------------------------
54.
55.
56.
57.         public URI getExecutable() {
58.             return _executable;
59.         }
60.     ----------------------------------------------------------------------
61.
62.
63.
64.         public boolean executableExists() {
65.             final File file = new File(getExecutable().toString());
66.             return file.exists() && file.isFile();
67.         }
68.     ----------------------------------------------------------------------
69.
70.
71.
72.         public void scanFile(final File arg) throws
 UploadRejectedException {
73.             if (Logging.isDebugEnabled(LOGGER)) {
74.                 Logging.logDebug("scanning file " + arg, LOGGER);// NON-
 NLS
75.             }
76.             final String[] args = {getExecutable().toString(),
 arg.toString()};
77.             try {
78.                 final StringWriter out = new StringWriter();
79.                 final StringWriter error = new StringWriter();
80.                 AccessUtil.executeProcess(out, error, args);
81.                 final String message = out.toString();
82.                 final String[] stat = message.split(":");
83.                 final String[] cause = message.split("\n");
84.                 if ( ! " OK".equals(stat[1].substring(0, 3))) {
85.                     Logging.logError("File upload rejected - " + message,
 LOGGER);// NON-NLS
86.                     throw new UploadRejectedException(cause[0]);
```

```
87.                 }
88.                 if (Logging.isDebugEnabled(LOGGER)) {
89.                     Logging.logDebug("scanned file " + arg, LOGGER);//
 NON-NLS
90.                 }
91.             } catch (IOException e) {
92.                 Logging.logError("Couldn't start process", e, LOGGER);//
 NON-NLS
93.                 throw new UploadRejectedException("Couldn't start upload
 filter process", e);
94.
95.             }
96.         }
97.     ==========$$$$$$$$$$$$$$$$$$$$==========
98.
99.
100.
101.        public boolean isThreadSafe() {
102.            return true;
103.        }
104.
105.    }
```

Listing 34: de.espirit.firstspirit.opt.vscan.engines.clamav.ClamScanEngine

3.14.8 Konfiguration und Persistenz – fs-vscan.conf

Die zentrale Konfigurationsdatei befindet sich im Archiv `fs-vscan.jar` und wird während des Modul-Installations-Prozesses in das FirstSpirit Modul-Konfigurationverzeichnis des VScan-Service (`conf/modules/FS VScan Service.VScanService`) ausgerollt (Verzeichnisstruktur siehe Kapitel 2.7 und 2.8 ab Seite 12).

3.15 Namensgleichheit bei Modul-Ressourcen (z.B. Zip-Exportdateien)

Existieren verschiedene Module mit namensgleichen Ressourcen, z.B. `myZipExport.zip` in Modul01 und Modul02 z.B. jeweils in den Modulverzeichnissen `files` innerhalb eines FSM-Archives, wird durch den Server-Classloader die erste gefundene Ressource mit dem Dateinamen `myZipExport.zip` geladen, also analog zur Java-VM. Wenn mehrere gleichnamige Jars im Classpath liegen, gewinnt die erste gefundene Ressource. Grundlegend sind Modul-Ressourcen systemweit verfügbar. D.h., sie werden in ein

temporäres Verzeichnis (eines pro Modul) entpackt und sind dann über den Server-ClassLoader erreichbar. Das führt dazu, dass bei Namensgleichheit von Dateien das erste Modul gewinnt bzw. dessen Ressource `myZipExport.zip` „gewinnt“.

Abbildung 3-14: Namensgleiche Modul-Ressourcen

Um diese Limitierung zu umgehen, ist der empfohlene Weg, die Dateien mit ins Jar neben die Klasse zu legen, die diese Dateien (`myZipExport.zip`) benötigt. Mit `MyClass.class.getRessourceAsStream(...)` können diese Ressourcen anschließend konfliktlos geladen werden, in diesem Fall also `ZipImporter.class.getResourceAsStream( myZipExport.zip);`.

Nicht zu verwechseln mit getResource(…); hier wird eine URL zurückgegeben, welche Remote auf dem FirstSpirit-Server nicht zum temporären Verzeichnis umgesetzt wird.

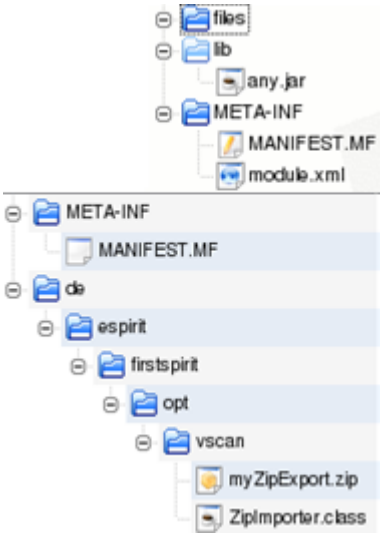


Abbildung 3-15: Laden namensgleicher Modul-Ressourcen

3.16 Erzeugung von FSM-Archiven (.fsm)

```
1.    <target name="main-fsm" depends="main-jar " description="Builds the
 main module file (*.fsm).">
2.            <mkdir dir="/tmp/myModuleName/lib"/>
3.            <mkdir dir="/tmp/myModuleName/META-INF"/>
4.            <copy file="Path to the previous created Jar
 File/myModuleName.jar" todir="/tmp/myModuleName/lib"
 preservelastmodified="true" overwrite="true" />
5.            <copy file="module.xml" tofile="/tmp/myModuleName/META-
 INF/module.xml" overwrite="true">
6.                <filterset>
7.                    <filter token="VERSION" value="1.1.1"/>
8.                </filterset>
9.            </copy>
10.           <jar jarfile="Output dir path/myModuleName.fsm"
 basedir="/tmp/myModuleName" excludes="**/*.dependency" duplicate="preserve"
 keepcompression="true"/>
11.   </target>
```

Listing 35: FSM-Archiv-Erzeugung – build.xml

Ein komplettes Beispiel einer build.xml Datei findet sich in Kapitel 3.11.2 (Seite 66).

3.17 Signieren von Modulen – Jar-Archiv-Klassen

Signierte Module werden bei der Installation überprüft und gegebenenfalls zurückgewiesen. Die Signierungs-Operation lässt sich beispielsweise mit folgendem Ant-Aufruf in die build.xml (Kapitel 3.11.2 Seite 66) integrieren.

```
1.    <property name="signkey" value="yourKeySigningCompany"/>
2.    <signjar jar="themodule.jar" alias="${signkey}" storepass="yourPass"
 keystore="yourKeyFile.jks"/>
```

Listing 36: Signieren von Modulen mit einer Ant-Task

alias	Auswahl des Private-Keys (Zertifikat) anhand des eindeutigen Bezeichners innerhalb der Keystore-Datei.
storepass	Passwort für die Keystore-Integrität (Private-Key Passwort)
keystore	Key-Archive („Schlüsselbund"), Keystore-Datei, die ggf. mehrere Zertifikate enthält.

3.18 Internationalisierung von Modulen – i18n

Generell ist die aktuelle selektierte Sprache des FirstSpirit JavaClients oder der Server- und Projektkonfiguration über `Locale.getDefault()` verfügbar.

Eine Beispiel-Implementierung könnte wie folgt aussehen:

```
1.    package de.espirit.firstspirit.opt.vscan.resources;
2.
3.    import de.espirit.common.base.Logging;
4.
5.    import javax.swing.Icon;
6.    import javax.swing.ImageIcon;
7.    import java.awt.Image;
8.    import java.awt.Toolkit;
9.    import java.io.IOException;
10.   import java.io.InputStream;
11.   import java.util.Locale;
12.   import java.util.MissingResourceException;
13.   import java.util.ResourceBundle;
14.
15.
16.   /**
17.    * $Date: 2008-05-26 15:08:34 +0200 (Mo, 26 Mai 2008) $
18.    *
19.    * @version $Revision: 22537 $
20.    */
21.   public final class ModuleResources {
22.
23.       public static final Class LOGGER = ModuleResources.class;
24.
25.       private static volatile ResourceBundle _bundle;
26.       private static final String LOCALE_RESOURCES_PKG =
 "de.espirit.firstspirit.opt.vscan.resources.locale.Messages";// NON-NLS
27.       private static final String ICON_RESOURCES_PKG =
 "/de/espirit/firstspirit/opt/vscan/resources/icons";// NON-NLS
28.       private static final byte[] BYTE = new byte[0];
29.   ================================================================
30.
31.
32.
33.       private ModuleResources() {
34.       }
35.   ================================================================
36.
37.
38.
39.       public static ResourceBundle getResourceBundle() {
```

```
40.             Logging.logInfo("Loading locale properties for '" +
 Locale.getDefault() + ")", LOGGER);// NON-NLS
41.
42.             if (_bundle != null) {
43.                 return _bundle;
44.             }
45.
46.             //noinspection UnusedCatchParameter
47.             try {
48.                 _bundle = ResourceBundle.getBundle(LOCALE_RESOURCES_PKG,
 Locale.getDefault());
49.             } catch (MissingResourceException e) {
50.                 throw new MissingResourceException("Missing Resource
 bundle: " + Locale.getDefault() + " ", LOCALE_RESOURCES_PKG, "");
51.             }
52.
53.             return _bundle;
54.         }
55.     ================================================================
56.
57.
58.
59.         public static ResourceBundle getResourceBundle(final String
 localeResourceFolder) {
60.             Logging.logInfo("Loading locale properties for '" +
 Locale.getDefault() + "'", LOGGER);// NON-NLS
61.             //noinspection UnusedCatchParameter
62.             try {
63.                 _bundle = ResourceBundle.getBundle(localeResourceFolder,
 Locale.getDefault());
64.             } catch (MissingResourceException e) {
65.                 throw new MissingResourceException("Missing Resource
 bundle: " + Locale.getDefault() + " ", localeResourceFolder, "");
66.             }
67.
68.             return _bundle;
69.         }
70.     ================================================================
71.
72.
73.
74.         @SuppressWarnings({"UnusedCatchParameter"})
75.         public static String getString(final String key) {
```

```
76.             try {
77.                 if (_bundle == null) {
78.                     getResourceBundle();
79.                 }
80.                 return _bundle.getString(key);
81.             } catch (MissingResourceException e) {
82.                 throw new MissingResourceException("Missing Resource: " +
 Locale.getDefault() + " - key: " + key + "  - resources: " +
 LOCALE_RESOURCES_PKG, LOCALE_RESOURCES_PKG, key);
83.             } catch (NullPointerException e) {
84.                 throw new NullPointerException("No bundle set: use
 ModuleResources.getResourceBundle().getString(...)");
85.             }
86.
87.         }
88.     ================================================================
89.
90.
91.
92.         private static InputStream getResourceStream(final String
 iconResourcePkg, final String filename) {
93.             String iconResourcePkg1 = iconResourcePkg;
94.             if (iconResourcePkg1.startsWith("/")) {
95.                 iconResourcePkg1 = iconResourcePkg1.replaceFirst("/", "");
96.             }
97.
98.             final String resname = "/" + iconResourcePkg1.replace('.',
 '/') + "/" + filename;
99.             final Class clazz = ModuleResources.class;
100.
101.             return clazz.getResourceAsStream(resname);
102.         }
103.     ================================================================
104.
105.
106.
107.     }
```

Listing 37: Internationalisierung ModuleResources.java

(siehe Datei "ModulResources.java", enthalten in der ZIP-Datei mit der Beispiel-Implementierung, FirstSpirit Online-Dokumentation, Kapitel „Dokumentation" / „Für Entwickler").

Ein exemplarisches Anwendungsbeispiel für die Nutzung findet sich in der Konfigurationsoberflächen-Klasse (`VScanServiceConfigPanel`, siehe Seite 110, Klassen-Beschreibung VScanServiceConfigPanel).

```
ModuleResources.getString("fs-
resource.module.vscan.admin.gui.VScanServiceConfigPanel_EngineSetu
p")
```

Hierbei werden immer die Ressourcen aus dem Package `de.espirit.firstspirit.opt.vscan.resources.locale` verwendet. Um andere Ressourcen zu nutzen, muss der Pfad hier über die Methode explizit angegeben werden. Hier steht die Methode

```
public static ResourceBundle getResourceBundle(final String
localeResourcePath)
```

zur Verfügung, über die sich ein Package-"Pfad" übergeben lässt. Ein Aufruf würde sich beispielsweise wie folgt darstellen:

```
ModuleResources.getResourceBundle(other.resources.locale).getStrin
g("fs-
resource.module.vscan.admin.gui.VScanServiceConfigPanel_EngineSetu
p")
```

3.19 Icon Ressourcen

Für die Ablage von Bildern ist das Package `de.espirit.firstspirit.opt.vscan.resources` vorgesehen.

```
ModuleResources.loadIcon("myImage.png")
```

Laden der Ressourcen aus dem default-Package für Icons (de.espirit.firstspirit.opt.vscan.resources.icons).

```
ModuleResources.loadIcon("de.espirit.firstspirit.opt.vscan.resources.icons", "myImage.png")
```

Laden der Ressourcen aus einer beliebigen Quelle im JAR order außerhalb des Archivs.

3.20 Text Ressourcen

Mithilfe der Methode loadTextResource können Text-Dateien geladen werden:

```
public static String loadTextResource(final String pkgName, final String filename)
```

3.21 Integration von Eingabekomponenten (Editoren) in den JavaClient

3.21.1 Beispiel GOM-Form

Editoren nutzen für die Integration der Eingabekomponente das FirstSpirit GUI Object Model. Hierbei wird über die Definition eines XML-Identifiers, z.B. `CMS_INPUT_SIMPLE_EDITOR_CONTENT` (siehe auch Kapitel 3.9 Seite 29), in der GOM-Form (`de.espirit.firstspirit.access.store.templatestore.gom.GomFormElement`) die Komponente im FirstSpirit JavaClient eingebunden. Eingabekomponenten in FirstSpirit dienen dazu, die Arbeit (Einpflegen und Bearbeiten von Inhalten) für Redakteure so komfortabel wie möglich zu gestalten. Die gewünschten Eingabekomponenten werden vom Vorlagenentwickler im Registerbereich „Formular" der Vorlage eingebunden, deshalb wird hier der Ausdruck GOM-FORM verwendet (siehe auch Kapitel 3.9 Seite 47)

Für die einfache Editoren-Komponente (Kapitel 3.11 Seite 65) stellt sich die GOM-Form (gui.xml) wie folgt dar:

```
1.     <CMS_MODULE>
2.
3.       <CMS_INPUT_SIMPLE_EDITOR_CONTENT name="fs4see" hFill="yes"
 maxRows="10">
4.         <LANGINFOS>
5.           <LANGINFO lang="*" label="FirstSpirit Editor Example"/>
6.           <LANGINFO lang="DE" label="DE:FirstSpirit Editor Beispiel"/>
7.           <LANGINFO lang="EN" label="EN:FirstSpirit Editor Example"/>
8.         </LANGINFOS>
9.       </CMS_INPUT_SIMPLE_EDITOR_CONTENT>
10.
11.    </CMS_MODULE>
```

Listing 38: Simple Editor GOM-Form

Der Parameter `fs4see` ist hier der eindeutige Bezeichner der Komponente innerhalb des GOM und der Ausgabe-Kanäle (beispielsweise HTML, PDF). Weitere Parameter wie `hFill` sind optional, sie finden sich in den Hauptabschnitten der Online Dokumentation für FirstSpirit V4.0[10], / Vorlagenentwicklung / Formulare.

Form-Elemente müssen ein Fallback-Label definieren ***<LANGINFO lang="*"***

3.22 Modulansicht in der Server -und Projektkonfiguration

Nach erfolgter Installation des zuvor erstellen FSM-Archives stellt sich das installierte Modul in der FirstSpirit Server -und Projektkonfiguration wie folgt dar:

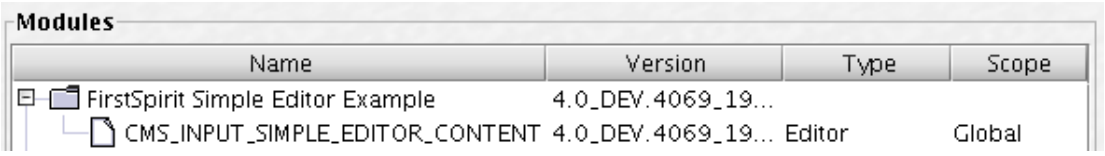


Abbildung 3-16: Modulansicht des installierten Simple Editor Example Moduls in der Server -und Projektkonfiguration

[10] [4] Online Dokumentation für FirstSpirit – ODFS

Abbildung 3-17: Mögliche, für das Simple Editor Example Modul ausführbare Aktionen

Abbildung 3-18: Mögliche, für das Simple Editor Example Modul ausführbare Aktionen in der FirstSpirit Version 4.1

3.23 Ansicht Komponente GOM-Form und Ausgabe-Kanäle (bspw. HTML, PDF)

Um die Editor-Komponenten im FirstSpirit JavaClient zur Verfügung zu stellen, wird in der Vorlagen-Verwaltung (JavaClient) die GOM-Form definiert (siehe Kapitel 3.21.1 Seite 131).

Preview | Properties | Form | html (HTML) | PDF (PDF)

```
<CMS_MODULE>

  <CMS_INPUT_SIMPLE_EDITOR_CONTENT name="fs4see" hFill="yes" maxRows="10">
    <LANGINFOS>
      <LANGINFO lang="*" label="FirstSpirit Editor Example"/>
      <LANGINFO lang="DE" label="DE:FirstSpirit Editor Beispiel"/>
      <LANGINFO lang="EN" label="EN:FirstSpirit Editor Example"/>
    </LANGINFOS>
  </CMS_INPUT_SIMPLE_EDITOR_CONTENT>

</CMS_MODULE>
```

Abbildung 3-19: Definition der GOM-Form für das Simple Editor Example Modul

Preview | Properties | Form | html (HTML) | PDF (PDF)

```
<CMS_HEADER>
</CMS_HEADER>
<div>FirstSpirit4 Editor Example</div>
<p>$CMS_VALUE(fs4see)$</p>
```

Abbildung 3-20: Definition der HTML-Darstellung (Ausgabekanal) für das Simple Editor Example Modul

Preview | Properties | Form | html (HTML) | PDF (PDF)

```
<CMS_HEADER>
</CMS_HEADER>
$CMS_IF(!fs4see.isEmpty)$
  <fo:block font-size="10pt" color="#3864B4" font-weight="bold" space-after="1mm">
   $CMS_VALUE(fs4see)$
  </fo:block>
$CMS_END_IF$
```

Abbildung 3-21: Definition des PDF-Ausgabekanals mittels Formatting Objects Processor innerhalb der FirstSpirit-Template-Syntax

4 Listings

5 Klassen-, Interface-Beschreibungen

6 Abbildungsverzeichnis

7 Referenzen

[1] Developer-API, *http://www.FirstSpirit.de/*, Copyright e-Spirit AG

[2] Access-API, *http://www.FirstSpirit.de/*, Copyright e-Spirit AG

[3] Dokumentation für Administratoren, *http://www.FirstSpirit.de/*, Copyright e-Spirit AG

[4] Online Dokumentation für FirstSpirit – ODFS, *http://www.FirstSpirit.de/*, Copyright e-Spirit AG

[5] Default Policy Implementation and Policy File Syntax, http://java.sun.com/j2se/1.5.0/docs/guide/security/PolicyFiles.html, Copyright Sun Microsystems, Inc. All Rights Reserved.

[6] Java Security, http://java.sun.com/javase/6/docs/technotes/guides/deployment/deployment-guide/security.html, Copyright Sun Microsystems, Inc. All Rights Reserved.

[7] JavaTM 2 Platform Security Architecture, http://java.sun.com/j2se/1.5.0/docs/guide/security/spec/security-spec.doc.html, Copyright © 1997-2002 Sun Microsystems, Inc. All Rights Reserved.

[8] JavaTM Web Start version 6 Frequently Asked Questions, http://java.sun.com/javase/6/docs/thttp://java.sun.com/javase/6/docs/technotes/guides/javaws/developersguide/faq.htmlechnotes/guides/javaws/developersguide/faq.html, March 2006

[9] JavaTM Network Launching Protocol (JNLP) Specification ("Specification"), http://jcp.org/en/jsr/detail?id=56, Copyright 2005 Sun Microsystems, Inc. All rights reserved.

[10] FirstSpirit Modul-Beispiele auf Basis von Eclipse .classpath, Copyright e-Spirit AG

8 Index