FirstSpirit™
Unlock Your Content

UX-Bridge Installationshandbuch

Version	**1.6**
Status	**RELEASED**
Datum	**2014-12-12**
Abteilung	Produktmanagement
Autor/ Autoren	C. Feddersen
Copyright	2014 e-Spirit AG
Dateiname	UX-Bridge_Installation_DE

e-Spirit AG

Barcelonaweg 14
44269 Dortmund | Germany

T +49 231 . 477 77-0
F +49 231 . 477 77-499

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

e-Spirit

Inhaltsverzeichnis

1 Einführung

Das Modul „UX-Bridge“ kommt dem Trend von dynamischen Websites nach. Immer dort, wo eine Vorgenerierung von Inhalten nicht möglich ist, muss dynamisch auf die CMS-Inhalte zugegriffen werden. Dynamisch bedeutet in diesem Fall, dass sich der Inhalt für jeden Website-User und zu jedem Zeitpunkt ändern kann. Die UX-Bridge bietet eine Infrastruktur für die Anforderung einer dynamischen Content-Delivery-Plattform. Somit ergänzt das Modul den hybriden Architekturansatz um eine Standardkomponente für dynamische Content-Auslieferung.

Weitere Informationen finden Sie im Whitepaper, Kapitel 1.3.

1.1 Komponenten

Die UX-Bridge besteht aus einer Reihe von Komponenten, die eine Infrastruktur für die Erstellung von Webapplikationen zur Verfügung stellen.

- UX-Bus
- FirstSpirit-Modul
- UX-Bridge API
- Adapter
- Content-Repository

Der Aufbau der Infrastruktur ist auf verschiedenste Weise möglich. So kann der UX-Bus auf dem FirstSpirit Server, auf einem Standalone Server oder auf dem gleichen Server wie die Webanwendung installiert werden.

Im Folgenden sind zwei gängige Architekturvarianten beschrieben.

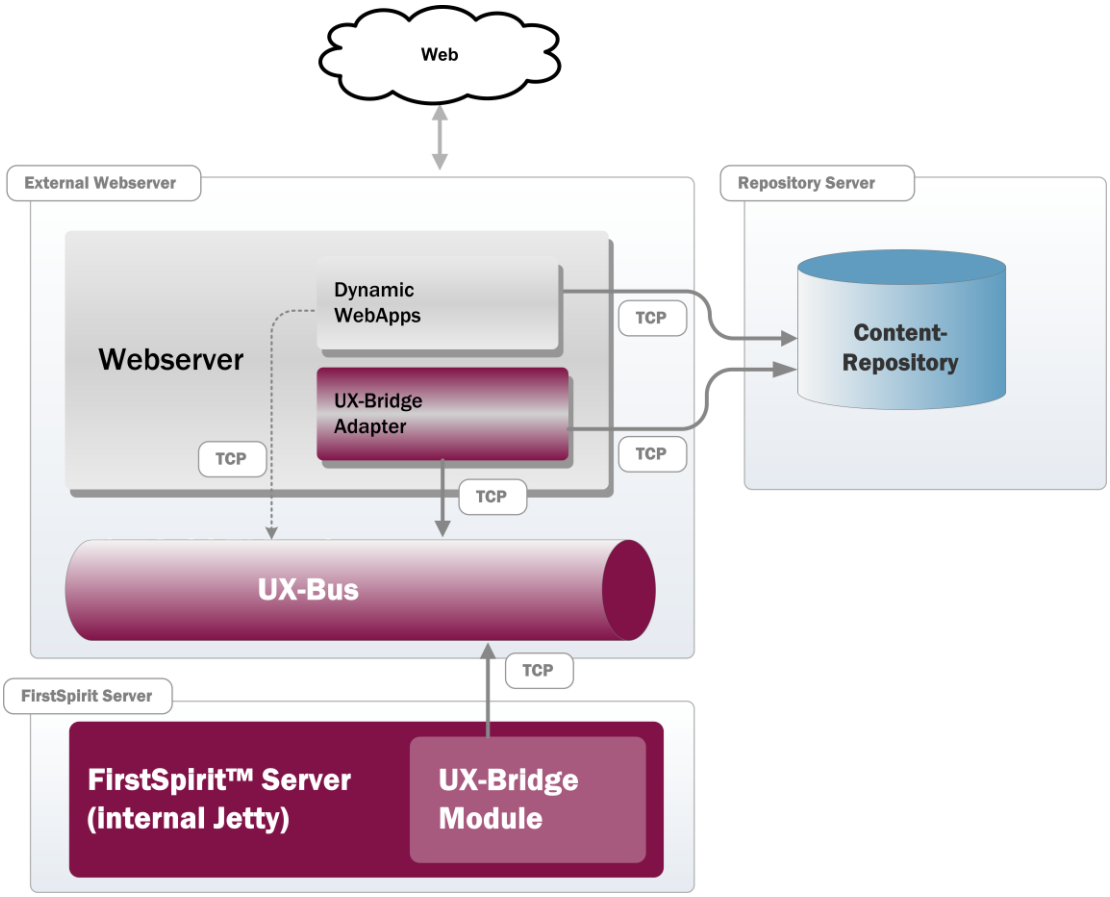


Abbildung 1: Architekturvariante 1

In der ersten Variante befinden sich der UX-Bus und die UX-Bridge Adapter auf der gleichen Maschine wie der Webcontainer, in dem die Webapplikation betrieben wird. Der UX-Bus wird als Standalone-Applikation betrieben. Die UX-Bridge Adapter nutzen den Webcontainer für die Webapplikation mit. Das Content-Repository befindet sich auf einer getrennten Maschine. Der FirstSpirit-Server befindet sich ebenfalls auf einer eigenen Maschine und ist somit von den Livesystemen entkoppelt.

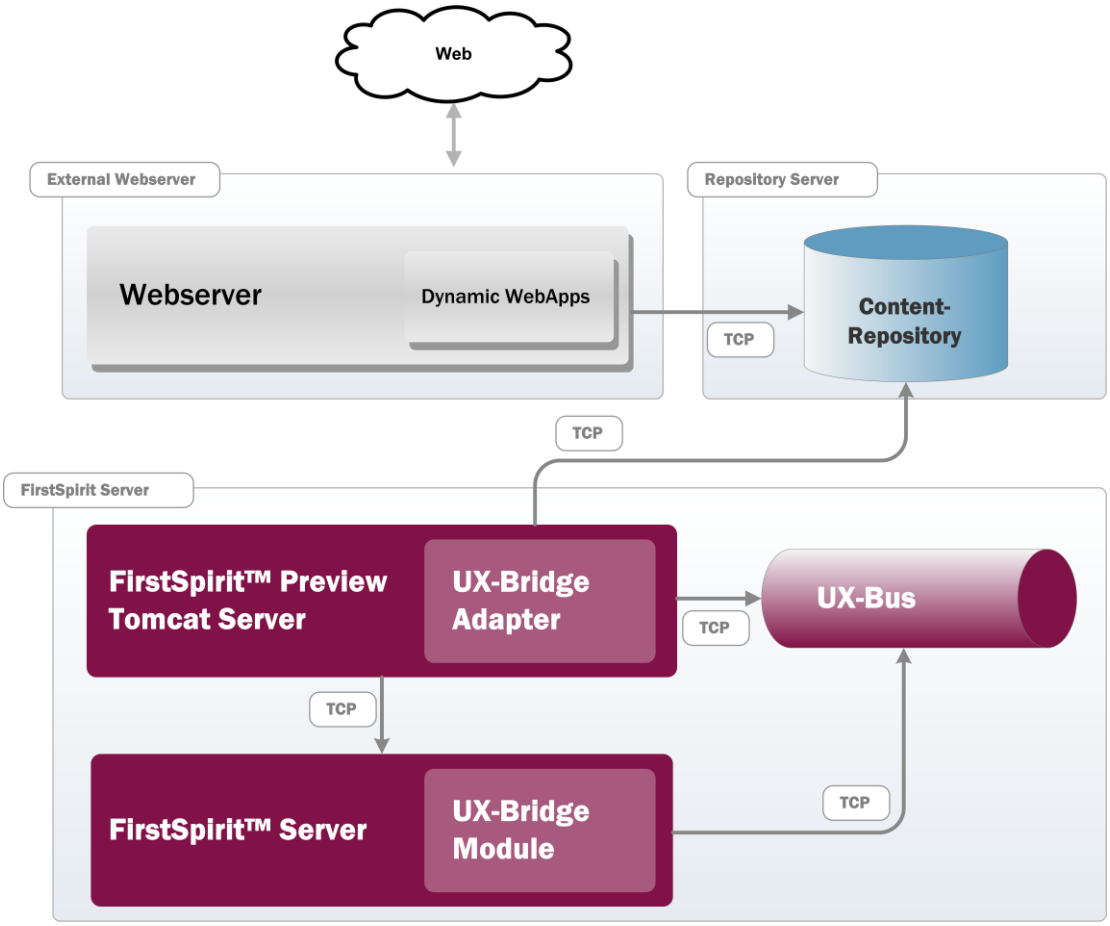


Abbildung 2: Architekturvariante 2

In der zweiten Variante befindet sich der UX-Bus und etwaige UX-Bridge Adapter zusammen mit dem FirstSpirit-Server auf einer Maschine. Dabei werden die UX-Bridge Adapter als Webapplikation in einem Apache Tomcat deployed, der auch für die Vorschau von FirstSpirit genutzt wird.

Andere Varianten sind möglich, die optimale Lösung hängt immer von den Anforderungen im Projekt ab. Entscheidungskriterien sind zum Beispiel:

- Welche Anforderungen gelten bzgl. Verfügbarkeit der einzelnen Komponenten?
- Welche Sicherheitsrichtlinien sind zu berücksichtigen? Muss der UX-Bus auch von Drittapplikationen aus erreichbar sein?
- Welche Technologien werden für die UX-Bridge Adapter, die Webapplikationen und das Content-Repository verwendet

- Wie sieht die Netzanbindung der Komponenten aus?

Weitere Informationen zu den Aspekten Sicherheit und Hochverfügbarkeit finden Sie in den entsprechenden Kapiteln dieser Dokumentation.

1.1.1 UX-Bus

Bei dem UX-Bus handelt es sich um eine Message Oriented Middleware (MOM) zum Austausch von Nachrichten unter den beteiligten Komponenten. Als Message Broker wird Apache ActiveMQ verwendet. Die Konfiguration des Routings erfolgt über das integrierte Apache Camel Framework.

Der UX-Bus leitet die gesendeten Nachrichten an die konfigurierten Endpunkte weiter. So werden die von FirstSpirit erzeugten Nachrichten an den entsprechenden Adapter geleitet. Dieser erzeugt nach der Speicherung der Daten eine Rückantwort, die vom UX-Bus an eine Komponente im FirstSpirit-Server verschickt wird.

Informationen zur Installation des UX-Bus finden Sie im Kapitel 2.1 Installation des UX-Bus, Seite 9.

1.1.2 FirstSpirit-Modul

Diese Komponente muss auf dem FirstSpirit-Server installiert werden. Die Hauptkomponente des Moduls ist ein Service, der Nachrichten erzeugt und an den UX-Bus schickt sowie Antworten von diesem entgegen nimmt. Die Installation und Konfiguration des Modules sind in Kapitel 2.2 Installation FirstSpirit-Modul beschrieben.

1.1.3 UX-Bridge API

Das UX-Bridge API ermöglicht das Verwenden des UXBServices in eigenen Modulen. Dazu müssen das uxbridge-module-api-<version>.jar und JDOM in der Version 1.0 im Classpath liegen. Auf den Service kann dann folgendermaßen zugegriffen werden:

```
UxbService uxbService =
context.getConnection().getService(UxbService.class);
```

1.1.4 Adapter

Adapter bilden die Schnittstelle zu den unterschiedlichen Content-Repositories und

sorgen dafür, dass die Daten in ein Repository geschrieben werden. Da Adapter sowohl von dem verwendeten Repository als auch dem Datenmodel abhängen, sind es projektspezifische Komponenten.

1.1.5 Content-Repository

Das Content-Repository oder auch Live-Repository ist eine Datenhaltungskomponente, die von FirstSpirit gefüllt und von Webapplikationen ausgelesen wird.

Wichtiges Architekturparadigma der UX-Bridge ist hier: „Die Art und Anzahl der Repositories wird nicht vorgegeben“, weil je nach Aufgabenart unterschiedliche Repositories unterschiedlich gut geeignet sind (siehe Whitepaper, Kapitel 2.1.3.2).

Je nach Anforderung des Projektes ist also das geeignete Content-Repository auszuwählen und zu installieren.

2 Installation

Die für den Betrieb benötigten Komponenten können je nach Anforderung auf unterschiedliche Weise installiert werden. In den folgenden Kapiteln wird nun auf die einzelnen Komponenten eingegangen.

Folgende Dateien sind Teil der Auslieferung:

- ux-bus.zip
- ux-bus.tar.gz
- uxbridge-module-api-<version>.jar
- uxbridge-camel-component-<version>.jar
- uxbridge-module-<version>.fsm
- uxbridge-bus-module-<version>.fsm
- uxbridge_tutorial_newsWidget.tar.gz
- uxbridge_tutorial_newsDrilldown.tar.gz
- UX-Bridge_Installation_*.pdf
- UX-Bridge_DeveloperDocumentation _*.pdf
- UX-Bridge_Technical_Datasheet_*.pdf
- third-party-dependencies.txt

2.1 Installation des UX-Bus

2.1.1 Standalone-Betrieb

Der Standalone-Betrieb ist der empfohlene Betriebsmodus des UX-Bus. In diesem Szenario sind alle wesentlichen Komponenten von einander entkoppelt. Außerdem eignet er sich am besten für die Konfiguration von Hochverfügbarkeits-Szenarien.

2.1.1.1 Installation

Der UX-Bus benötigt mindestens ein installiertes Java 6. Die genauen Systemanforderungen entnehmen Sie bitte dem mitgelieferten technischen Datenblatt.

Stellen Sie sicher, dass die Umgebungsvariable JAVA_HOME gesetzt ist. Außerdem sollte JAVA_HOME/bin der Pathvariablen hinzugefügt werden.

Um den UX-Bus zu installieren genügt es, die in Verbindung mit dem Modul ausgelieferte Distribution (UX-Bus_<VERSIONSNUMMER>.zip) in einen selbst gewählten, geeigneten Ordner zu entpacken. Diese Distribution ist bereits so konfiguriert, dass sie in Verbindung mit der UX-Bridge gute Ergebnisse erzielt. Es handelt sich um ein vorkonfiguriertes Apache ActiveMQ mit integriertem Apache Camel, in dem die Routen für den UX-Bus und die Adapter bereits konfiguriert wurden.

Im Ordner „bin“ finden sich die Startskripte. Das Skript „activemq“ kann zum Starten des Busses verwendet werden.

Ausgehend vom Programmordner, in welchem ActiveMQ liegt:

OS X/Linux:

```
./bin/activemq console
```

Windows:

```
bin/activemq
```

Hierbei wird erst einmal die Standardkonfiguration genutzt. Die Konfiguration liegt im Ordner „conf“ und muss natürlich später noch den eigenen Bedürfnissen angepasst werden.

2.1.1.2 Testen der Installation

Sobald der ActiveMQ Bus gestartet wurde, sollte eine Meldung darüber auf der Konsole geloggt werden:

```
INFO  ActiveMQ JMS Message Broker (ID:apple-s-Computer.local-
51222-1140729837569-0:0) has started
```

Mithilfe des Tools netstat kann nun geprüft werden, ob der Bus auch wirklich

erreichbar ist und der Port richtig konfiguriert wurde.

Der Port ist in der Standardkonfiguration 61616 und sollte entsprechend der eigenen Konfiguration angepasst werden.

Zum Testen einfach folgende Zeile in der Konsole ausführen:

Unter Windows:

```
netstat –an|find "61616"
```

Unter Linux:

```
netstat –an|grep 61616
```

2.1.1.3 Installation als Dienst

Für den produktiven Einsatz ist die Konfigurationen als Windows-Dienst bzw. Unix Service empfehlenswert. Dadurch wird sichergestellt, dass der Standalone Server nach einem Neustart des Betriebssystems ebenfalls automatisiert gestartet wird.

Informationen zur Installation als Dienst bzw. Service finden sie unter http://activemq.apache.org/run-broker.html

Bei der Installation unter Windows Server 2008 64bit sind allerdings die Hinweise auf folgender Seite zu beachten: http://blog.bigrocksoftware.com/2010/10/07/commons-daemon-procrun-as-a-java-service-wrapper-for-activemq/

2.1.2 Installation des UX-Bus auf dem FirstSpirit-Server

Der Betrieb des UX-Bus als Modul in einem FirstSpirit-Server wird nur empfohlen, wenn es aus betrieblichen Gründen nicht möglich ist, den UX-Bus als eigenständige Komponente zu installieren, Sie aber Zugriff und Administrationsrechte für den FirstSpirit Server besitzen.
In dieser Betriebsart kommt es in der Regel zu Warnungen im FirstSpirit Serverlog, die darauf zurückzuführen sind, dass der UXBService vor dem UX-Bus gestartet wird. Näheres zu diesen Warnungen finden Sie am Ende dieses Unterkapitels.

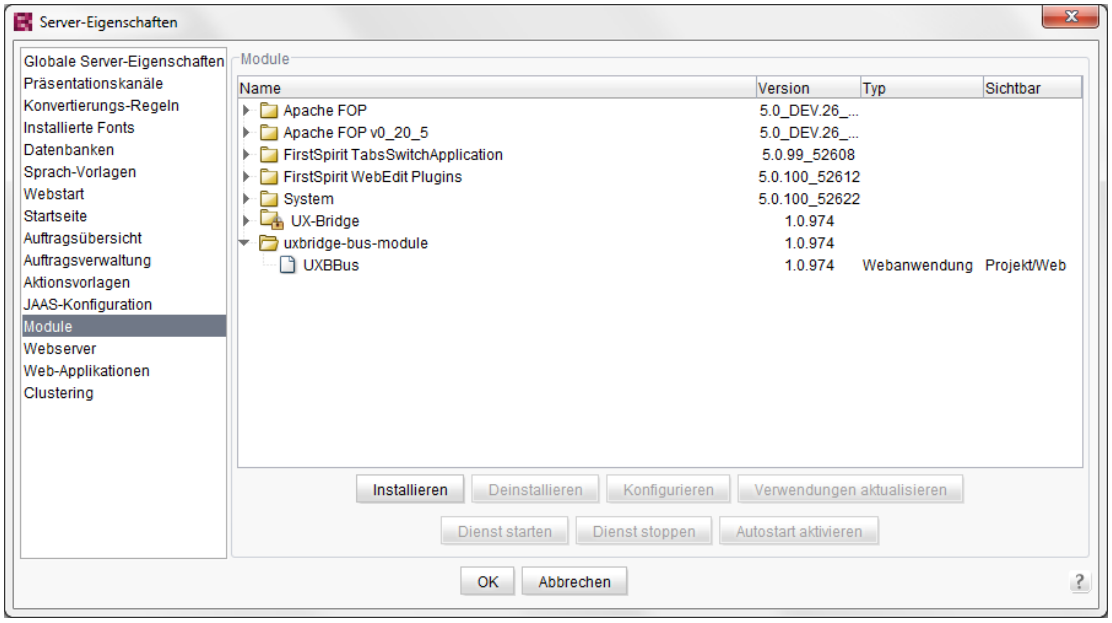
Sollten aber Punkte wie Rückkanal (WebApp -> FirstSpirit), Clustering und Failover

eine Rolle spielen, dann sollte der UX-Bus als Standalone Server betrieben werden.

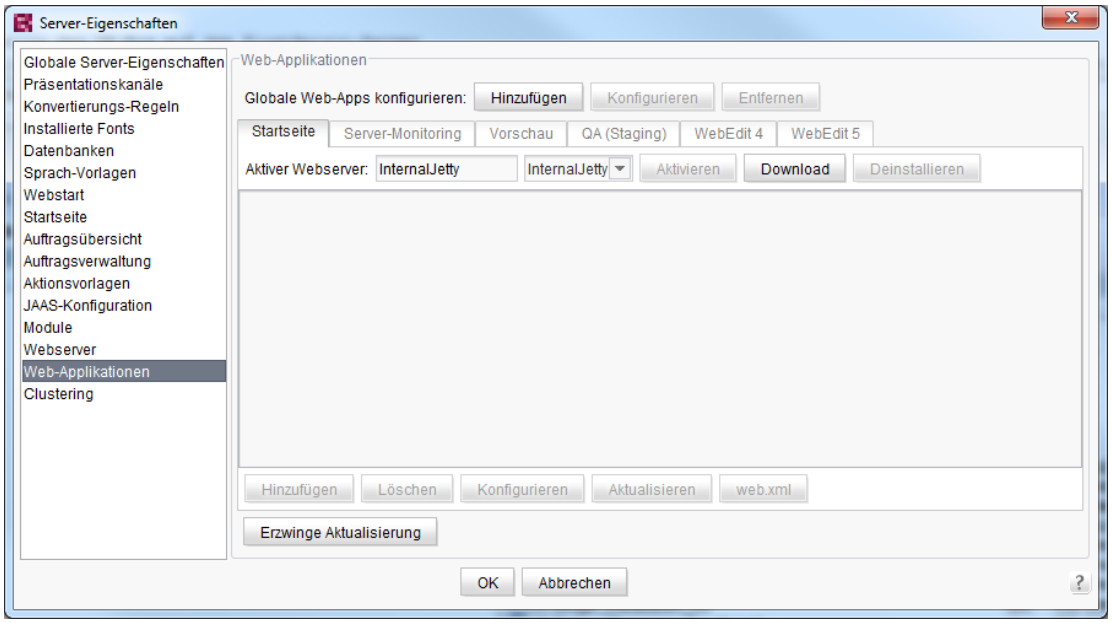
> *Unter FirstSpirit 4.2R4 ist zu beachten, dass der UX-Bus auf dem InternalJetty nicht lauffähig ist und daher entweder auf einem externen Server oder einem anderen internen Server als dem InternalJetty (z.B. Tomcat) installiert werden sollte.*

Wenn sie den UX-Bus in diesem Modus betreiben möchten, gehen sie bitte wie folgt vor:

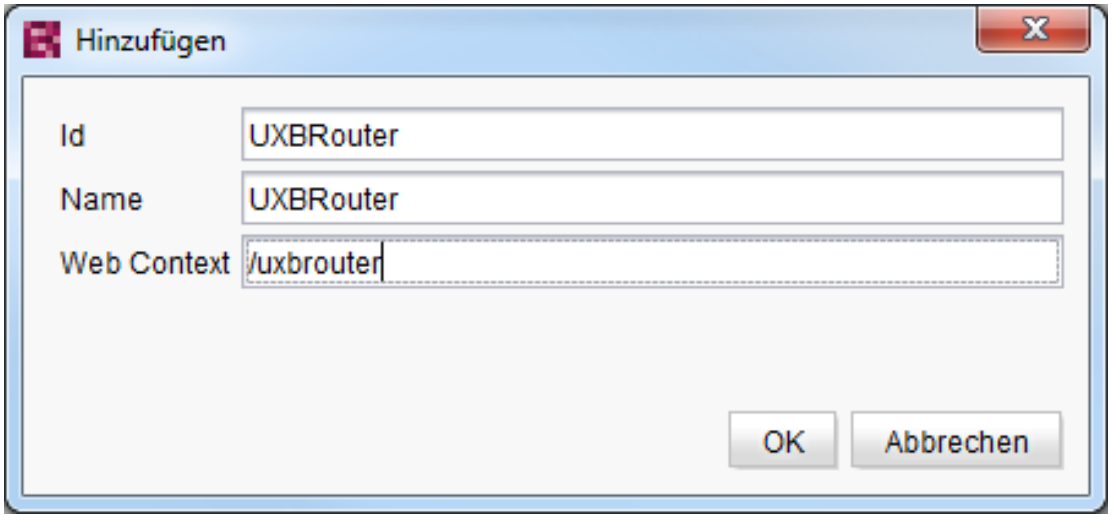
1. Installation des "UX-Bridge Bus" Moduls, dazu muss das mitgelieferte uxbridge-bus-module-<version>.fsm in den Server-Eigenschaften installiert werden.
   Um die Funktionsfähigkeit aller Module sicherzustellen, sollten Sie nach der Installation/Aktualisierung eines Modules den FirstSpirit-Server neustarten.



2. Unter Web-Applikationen muss eine globale Web-Anwendung erstellt werden

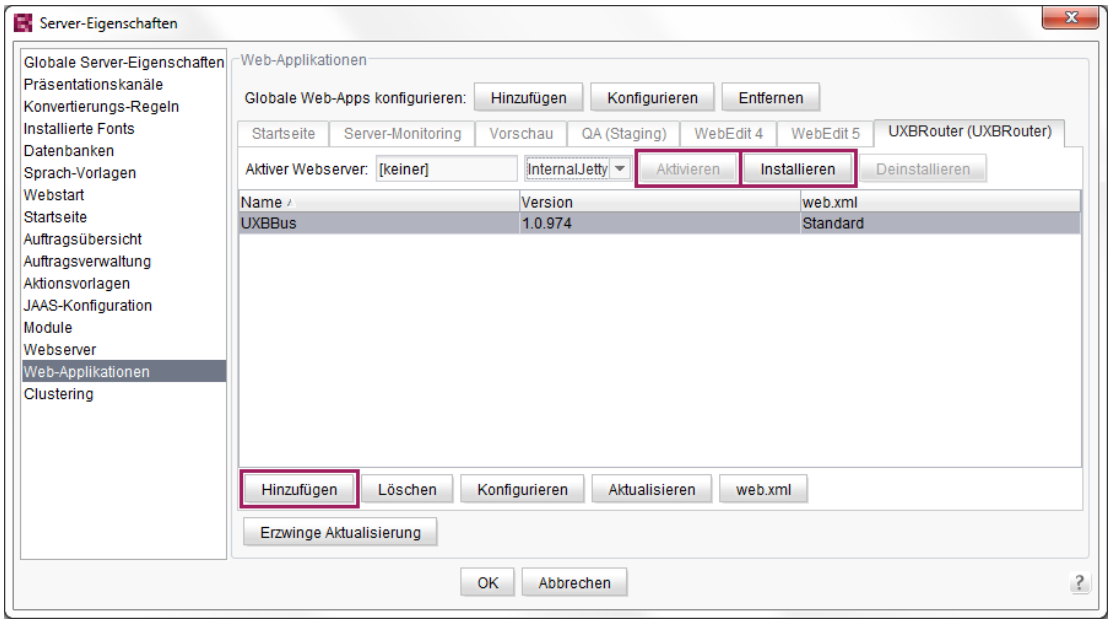


Die hier angegebenen Werte sind als Beispiel zu verstehen und können frei gewählt werden.



3. Durch Klicken auf Hinzufügen wird der UX-Bus auf der Webanwendung installiert. Beim Klicken auf den Button Konfiguration können die Routen angegeben und Einstellungen für den Broker gemacht werden.

4. Im Anschluss sollte als Webserver der „InternalJetty“ ausgewählt und installiert werden. Nach Abschluss der Installation muss auf den Button „aktivieren“ geklickt werden, um den Webserver zu starten.



Danach ist der UX-Bus einsatzbereit.

Weitere Informationen zu dem Thema Web-Applikationen finden sie im Handbuch für Administratoren im Kapitel 7.3.16 Web-Applikationen.

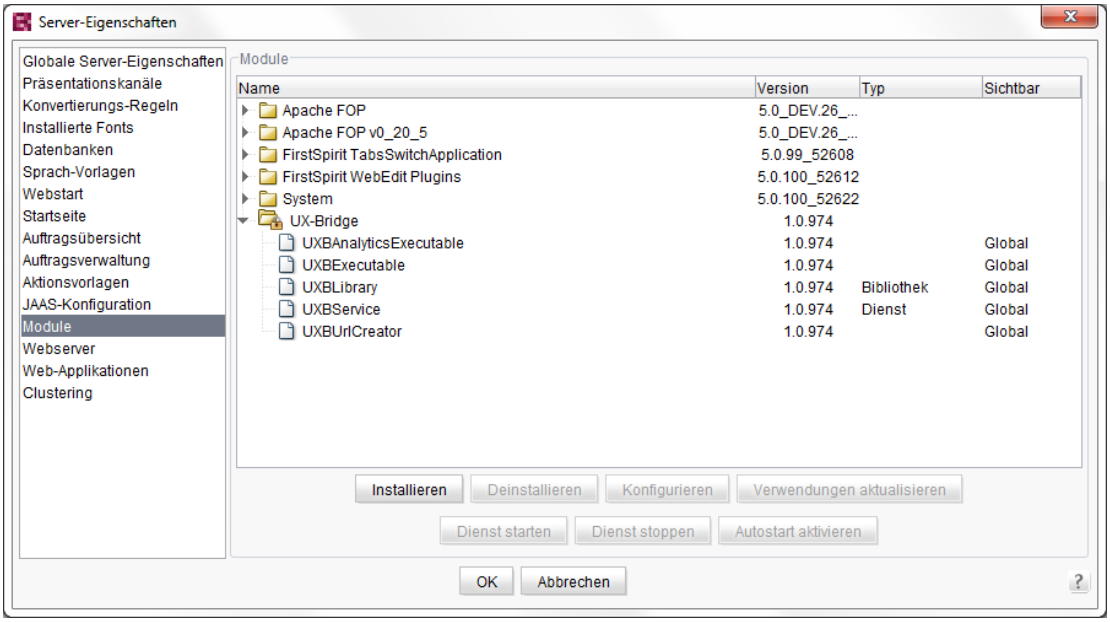
In diesem Betriebsmodus wird der UXBService vor dem UX-Bus gestartet. Da der UXBService direkt versucht, sich nach dem Start mit dem UX-Bus zu verbinden, werden im FirstSpirit Serverlog mehrmals folgende Fehlermeldungen ausgegeben. Nach dem Start aller Komponenten sollte diese Meldung dann nicht mehr

auftauchen. Hinweise für den Fall, dass die Meldung weiter auftaucht, finden sie im Kapitel 3.9 Fehler-Analyse.

```
WARN  20.07.2012 14:00:03.919
(org.apache.camel.component.jms.DefaultJmsMessageListenerContainer
): Could not refresh JMS Connection for destination 'FS_OUT' -
retrying in 5000 ms. Cause: Connection refused: connect
```

2.2 Installation FirstSpirit-Modul

Zur Installation des Moduls UX-Bridge auf dem FirstSpirit Server wählen Sie den Bereich „Module“ in den Servereigenschaften aus und klicken Sie auf den Button „Installieren“. Wählen Sie nun die zu installierende Datei uxbridge-module-<version>.fsm aus. Nach der Installation sollte ein neuer Ordner „UX-Bridge“ hinzugefügt worden sein. Wählen Sie den Eintrag aus, klicken Sie auf „Konfigurieren“, setzen Sie den Haken „Alle Rechte“ und bestätigen Sie Ihre Änderungen.



Schließen Sie nun die Servereigenschaften durch Klicken auf „OK“.

Um die Funktionsfähigkeit aller Module sicherzustellen, sollten Sie nach der Installation/Aktualisierung eines Modules den FirstSpirit-Server neustarten.

Weitere Informationen zur Installation von Modulen finden Sie in der Dokumentation für Administratoren, Kapitel 7.3.14 Module.

2.2.1 FirstSpirit 4.2R4

Wird UX-Bridge in einer FirstSpirit Version unter 5 eingesetzt, müssen aktuelle Versionen (1.6.4) der beiden Jars **slf4j-api.jar** und **slf4j-simple.jar** in den Ordner <fs>/shared/lib/ kopiert werden. Aktuelle Versionen dieser Dateien können von http://www.slf4j.org/ heruntergeladen werden.

Zusätzlich muss eine aktuelle Version (>=11.0.2) der Guava Library z.B. **guava-11.0.2.jar** in den Ordner <fs>/shared/lib/ kopiert werden. Die aktuelle Version dieser Datei kann von https://code.google.com/p/guava-libraries/ heruntergeladen werden.

Dies muss vor der Installation des uxbridge-module-<version>.fsm erfolgen. Außerdem ist ein Neustart des FirstSpirit-Servers notwendig, bevor das Modul installiert werden kann.

2.2.2 Konfiguration des UX-Bridge Services

Der UX-Bridge Service ist die Schnittstelle von FirstSpirit zum UX-Bus, über ihn werden die Nachrichten versendet.

Die Konfiguration des Services ist recht einfach:

1. Server-Eigenschaften -> Module öffnen
2. UX-Bridge aufklicken und UXBService auswählen
3. Konfigurieren klicken

2.2.2.1 XML Reiter

Die Konfiguration wird über eine Spring XML DSL vorgenommen. Weitere Informationen zur Syntax finden Sie unter http://camel.apache.org/spring.html.

Die Konfiguration der Beispielanwendung sieht z.B. folgendermaßen aus:

```
<?xml version="1.0" encoding="UTF-8"?>


<beans xmlns="http://www.springframework.org/schema/beans"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:jms="http://www.springframework.org/schema/jms"
   xsi:schemaLocation="http://www.springframework.org/schema/beans
       http://www.springframework.org/schema/beans/spring-beans-
3.0.xsd
```

```
       http://camel.apache.org/schema/spring

       http://camel.apache.org/schema/spring/camel-spring.xsd
http://www.springframework.org/schema/jms
http://www.springframework.org/schema/jms/spring-jms-3.0.xsd">

<bean id="jms"
class="org.apache.camel.component.jms.JmsComponent">

         <property name="connectionFactory">

                     <bean
class="org.apache.activemq.ActiveMQConnectionFactory">

                     <property name="brokerURL"
value="failover:(tcp://localhost:61616)?maxReconnectAttempts=2&amp
;startupMaxReconnectAttempts=10"/>

                     </bean>

         </property>

</bean>

<bean id="pooledConnectionFactory"
class="org.apache.activemq.pool.PooledConnectionFactory"

           destroy-method="stop">

         <property name="maxConnections" value="10"/>

         <property name="maximumActiveSessionPerConnection"

                   value="500"/>

         <property name="connectionFactory"

                   ref="jmsConnectionFactory"/>

</bean>


<bean id="jmsConfig"
class="org.apache.camel.component.jms.JmsConfiguration">

         <property name="connectionFactory"
ref="pooledConnectionFactory"/>

         <property name="transacted" value="false"/>

         <property name="concurrentConsumers" value="10"/>

         <property name="deliveryPersistent" value="true" />

</bean>


<bean id="jmsConnectionFactory"
class="org.apache.activemq.ActiveMQConnectionFactory">

         <property name="brokerURL"
value="failover:(tcp://localhost:61616)?maxReconnectAttempts=2&amp
;startupMaxReconnectAttempts=10"/>

</bean>

<bean id="activemq"
class="org.apache.activemq.camel.component.ActiveMQComponent">
```

```
          <property name="configuration" ref="jmsConfig"/>
</bean>


<camelContext xmlns="http://camel.apache.org/schema/spring"
id="camelContext" trace="false">
          <package>com.espirit.moddev.uxbridge.service</package>
          <template id="producerTemplate"/>
          <endpoint id="FS-Out"
uri="activemq:topic:FS_OUT"></endpoint>


        <onException>
            <exception>java.lang.Exception</exception>
                    <handled>

<constant>true</constant>
                    </handled>
                    <process ref="uxbExceptionProcessor" />
          </onException>


          <route id="Adapter-Statistics-Response-Route">
                    <from uri="jms:topic:FS_IN"/>
                    <convertBodyTo
type="com.espirit.moddev.uxbridge.api.v1.service.UXBEntity"/>
                    <bean
ref="UxbServiceStatisticsResponseHandler" method="print"/>
          </route>
</camelContext>
<bean id="UxbServiceStatisticsResponseHandler"
class="com.espirit.moddev.uxbridge.service.UxbServiceStatisticsRes
ponseHandler">
          <constructor-arg ref="camelContext"/>
</bean>
<bean id="uxbExceptionProcessor"
class="com.espirit.moddev.uxbridge.inline.UxbExceptionProcessor"/>
</beans>
```

- **Jms**:
  In diesem Bean wird die Apache ActiveMQ Verbindung konfiguriert. Über das Property brokerURL kann neben der Verbindung u.a. auch das Failover-Verhalten konfiguriert werden.

  Informationen zu allen Konfigurationsparametern finden Sie unter den

folgenden Links:

Transport: http://activemq.apache.org/configuring-version-5-transports.html
Connection: http://activemq.apache.org/connection-configuration-uri.html
Failover: http://activemq.apache.org/failover-transport-reference.html
TCP: http://activemq.apache.org/tcp-transport-reference.html

- **CamelContext**:
  Hier werden zwei Routen konfiguriert:
  FS_OUT = Die Route auf die FirstSpirit die Daten schickt
  FS_IN = Die Route auf der FirstSpirit auf neue Nachrichten wartet

- **UxbServiceStatisticsResponseHandler**:
  Dieses Bean ist der Handler für die Nachrichten, die von den Adaptern zurück gesendet werden.

- **uxbExceptionProcessor**
  Dieses Bean dient zur Verarbeitung der während des Transports auftretenden Exceptions.

Zu Entwicklungs- und Testzwecken ist es möglich den UX-Bridge Service so zu konfigurieren, dass er ohne einen laufenden Bus auskommt (zum Thema Bus siehe auch Kapitel 3.1 Routing). Wenn es dem UX-Bridge Service nicht gelingt, sich mit einem Bus zu verbinden, loggt er regelmäßig eine Meldung darüber. Geschieht dies über längere Zeiträume, kann es schwierig werden, die Logs zu lesen. Deshalb kann eine Dummykonfiguration angelegt werden, die dafür sorgt, dass alle Nachrichten als Dateien in einem bestimmten Ordner abgelegt werden.

```
<?xml version="1.0" encoding="UTF-8"?>


<beans xmlns="http://www.springframework.org/schema/beans"
           xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
xsi:schemaLocation="http://www.springframework.org/schema/beans
       http://www.springframework.org/schema/beans/spring-beans-
3.0.xsd
       http://camel.apache.org/schema/spring
       http://camel.apache.org/schema/spring/camel-spring.xsd">
         <camelContext
xmlns="http://camel.apache.org/schema/spring" id="camelContext"
trace="false">
         <package>com.espirit.moddev.uxbridge.service</package>
                     <template id="producerTemplate"/>
                     <endpoint id="FS-Out" uri="file://log/uxb-
```

```
messages"></endpoint>


        <onException>
            <exception>java.lang.Exception</exception>
            <handled>
                    <constant>true</constant>
            </handled>
            <process ref="uxbExceptionProcessor" />
         </onException>
         <route id="Adapter-Statistics-Response-Route">
                    <from uri="direct:in"/>
                    <convertBodyTo
type="com.espirit.moddev.uxbridge.api.v1.service.UXBEntity"/>
         </route>
         </camelContext>
         <bean id="uxbExceptionProcessor"
class="com.espirit.moddev.uxbridge.inline.UxbExceptionProcessor"/
>
</beans>
```

Nachdem der Service neu konfiguriert wurde, muss er neu gestartet werden, um die Änderungen anzuwenden.

In diesem Beispiel werden die Nachrichten im Ordner *<FS-Home>/log/uxb-messages/* abgelegt. Diese Dateien werden allerdings nicht automatisch entfernt, was je nach Menge der gesendeten Nachrichten dazu führen kann, dass der Ordner sehr groß wird. Wenn viele Nachrichten zu erwarten sind, sollte man einen Mechanismus einrichten, welcher alte Nachrichten regelmäßig löscht.

2.2.2.2 Optionen Reiter

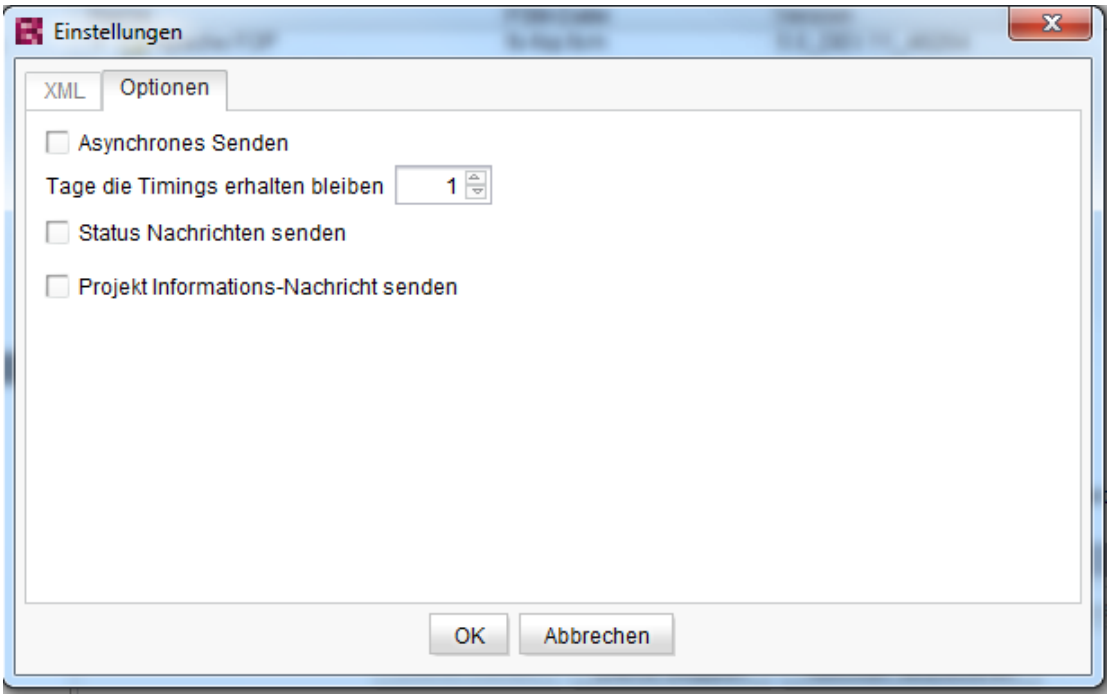
Im zweiten Reiter des UXBService kann definiert werden ob die Nachrichten asynchron oder synchron versendet werden sollen. Durch das Setzen des Hakens werden die Nachrichten asynchron versendet. Dadurch kann eine höhere Performance erreicht werden, da anders als bei der synchronen Kommunikation nicht auf eine Rückantwort gewartet wird. Diese Verbesserung der Performance kann bei großen Datenmengen sehr vorteilhaft sein.

Informationen zu versandten Nachrichten werden zu Zeitmessungszwecken intern in einem Cache gespeichert, bis eine Rückantwort von einem Adapter eingetroffen ist. Über das Eingabefeld „Tage die Timings erhalten bleiben“ kann die Dauer definiert werden, wie lange gewartet werden soll, bevor Nachrichten ohne Rückantwort aus

dem Cache entfernt werden.

Über die Checkbox „Status Nachrichten senden“ kann konfiguriert werden ob die UX-Bridge zu Beginn und nach der Generierung eine weitere Nachricht an den Bus schickt, so das auf diese Events im Adapter gezielt reagiert werden kann (weitere Details s. Entwicklerhandbuch).

Die Checkbox „Projekt Informations-Nachricht senden“ steuert ob zu Beginn der Generierung einmalig eine Nachricht mir Projektinformationen gesendet wird (weitere Details s. Entwicklerhandbuch).



2.3 Installation von Adaptern

Bei Adaptern handelt es sich um Komponenten, die innerhalb des Projektes entwickelt werden. Um die Administration und Wartung einfach zu halten, bietet es sich an, den Adapter als Webapplikation zu entwickeln, die in einen Web-/Servlet-Container bzw. Applikation-Server installiert werden. Die notwendige Infrastruktur ist in vielen Fällen bereits vorhanden. Notwendige Voraussetzung ist dies jedoch nicht, so kann der Adapter durchaus auch als eigenständige Java-Applikation entwickelt werden, die auf einem Server installiert wird.

Weitere Hinweise zur Entwicklung von Adaptern finden Sie in der

DeveloperDocumentation in Kapitel 3.3 Adapter.

3 Konfiguration

Zu Beginn verfügt der UX-Bus über einige Standard-Routen, die aber nach Bedarf angepasst und erweitert werden können.
Je nach Art der Installation, Standalone oder in FirstSpirit, müssen Änderungen an der Konfiguration an unterschiedlichen Stellen gemacht werden. Diese Aspekte werden in den ersten Kapiteln geklärt.

In den darauf folgenden Kapiteln werden Themen zur Hochverfügbarkeit aber auch zu Security, Monitoring und Logging behandelt.

3.1 Routing

Durch das Routing werden Nachrichten vorgefiltert und verteilt. Das Routing wird in einer, durch die Camel-DSL erweiterte, Spring-XML-Datei konfiguriert (http://camel.apache.org/spring.html).

Standardmäßig benötigt FirstSpirit zwei Routen. FS_OUT für das Senden und FS_IN für den Empfang von Nachrichten. Im Routing können dann je nach Bedarf und Anwendungsfall verschiedene Routen definiert werden. Die Nachrichten, die über die FS_* Routen gesendet werden, können gefiltert und/oder verteilt werden.

Die Standard Routen:

```
<beans
   xmlns="http://www.springframework.org/schema/beans"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://camel.apache.org/schema/spring
http://camel.apache.org/schema/spring/camel-spring.xsd
   http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd">


    <camelContext id="camel"
xmlns="http://camel.apache.org/schema/spring">


        <route id="uxbridge-route1">
            <from uri="activemq:topic:FS_OUT"/>
            <to uri="activemq:topic:BUS_OUT"/>
        </route>
        <route id="uxbridge-route2">
```

```
            <from uri="activemq:topic:BUS_IN"/>
            <to uri="activemq:topic:FS_IN"/>
        </route>
    </camelContext>
</beans>
```

Die Routen sind wie die Adapter sehr projektspezifisch. Eine tiefere Einführung dazu findet Sie im Kapitel 3.5 Routing in der DeveloperDocumentation .

3.1.1 Standalone UX-Bus

Das Routing wird in der Datei conf/camel.xml im Installationsverzeichnis des UX-Bus definiert. In der mitgelieferten Distribution sind bereits einige Standardrouten vorgegeben, so dass initial keine Anpassungen notwendig sind. Details zur Konfiguration des Routings finden Sie in Kapitel 3.6 Änderung des Routings auf Seite 38.

3.1.2 FirstSpirit-Server

Wurde der UX-Bus in dem FirstSpirit-Server installiert, dann gehen sie zur Konfiguration der Routen bitte folgendermaßen vor.

1. Starten sie die Server- und Projektkonfiguration
2. Öffnen sie die Server-Eigenschaften
3. Wählen sie im linken Menu den Punkt „Web-Applikation“
4. Wählen sie die erstellt globale Web-Anwendung aus (siehe Kapitel 2.1.2 Installation des UX-Bus auf dem FirstSpirit-Server, Seite 11)
5. Klicken sie auf den Button „Konfigurieren“
6. In dem Dialog können sie nun neue Routen anlegen bzw. bestehende Routen bearbeiten.

Eine Beschreibung der in FirstSpirit zu konfigurierenden Routen finden Sie im Kapitel 3.5.1 Endpunkte in FirstSpirit in der DeveloperDocumentation.

3.2 Logging

Dieses Kapitel beschäftigt sich mit der Konfiguration der Logging Mechanismen von

Apache ActiveMQ. Bei Fehlern sind die Logfiles und die Logausgaben auf der Standardausgabe die ersten Anlaufstellen um nach den Ursachen zu suchen.

Apache ActiveMQ nutzt für das interne Logging die Apache Commons Logging API. Deshalb kann es problemlos mit verschiedensten im Java Umfeld eingesetzten Logging Frameworks eingesetzt werden. In der Standardkonfiguration wird für das Logging Apache log4j eingesetzt. Für detaillierte Informationen zur Konfiguration von log4j kann die Online-Dokumentation von log4j herangezogen werden: http://logging.apache.org/log4j/1.2/

Die Apache ActiveMQ Dokumentation enthält ebenfalls einige Informationen zum Thema über Logging: http://activemq.apache.org/how-can-i-enable-detailed-logging.html

3.2.1 Logging im UX-Bus

Die Standardausgabedatei für das Logging ist data/activemq.log. Bei Problemen sollte sie die erste Anlaufstelle sein. Das Logging kann in der Datei conf/log4j.properties konfiguriert werden, um zum Beispiel das Loglevel den eigenen Bedürfnissen anzupassen. In der Standardkonfiguration wird das Loglevel "INFO" verwendet. Es wird sowohl auf der Konsole als auch in dem Logfile ausgegeben:

```
log4j.rootLogger=INFO, console, logfile
```

Weiterhin kann man einzelne Pakete definieren, welche ein anderes Loglevel nutzen sollen. In der Standardkonfiguration sind es:

```
log4j.logger.org.apache.activemq.spring=WARN
log4j.logger.org.apache.activemq.web.handler=WARN
log4j.logger.org.springframework=WARN
log4j.logger.org.apache.xbean=WARN
log4j.logger.org.apache.camel=INFO
```

Für Debuggingzwecke kann das Loglevel dann beispielsweise durch das Ersetzen der jeweiligen Zeile angepasst werden. Für das Debuglevel also:

```
log4j.rootLogger=DEBUG, console, logfile
```

Hierbei sollte man sich dessen bewusst sein, dass im Debug Modus viel mehr Meldungen erscheinen werden, was es erschwert, die relevanten Meldungen herauszufiltern. Um dies zu verhindern, könnte man die rootLogger Konfiguration auf INFO belassen und nur die Konfiguration für das zu untersuchende Paket auf DEBUG oder sogar TRACE setzen. Also:

```
log4j.logger.mein.eigenes.package=TRACE
```

In der mitgelieferten Distribution reicht es folgende beiden Zeilen in der Datei <activemq>/conf/log4j.properties zu aktivieren, um passende Debugmeldungen für das Routing und die Verarbeitung der Nachrichten zu bekommen:

```
# Or for more fine grained debug logging uncomment one of these
#log4j.logger.org.apache.activemq=DEBUG
#log4j.logger.org.apache.camel=DEBUG
```

3.2.2 LoggingBrokerPlugin

Das LoggingBrokerPlugin ermöglicht es noch detaillierter zu loggen, als log4j es tut und vor allem mehr Informationen darüber zu bekommen, was im UX-Bus passiert. Man kann das Plugin einfach in der conf/activemq.xml Konfigurationsdatei zu den <plugins> hinzufügen:

```
...
  <plugins>
    <loggingBrokerPlugin logAll="true"/>
  </plugins>
...
```

Nähere Erklärungen & weitere Attribute sind in der Apache ActiveMQ Dokumentation enthalten:
http://activemq.apache.org/logging-interceptor.html

3.2.3 Logging in Clients

Bei Fehlern und Problemen auf dem Broker hilft das Logging des UX-Bus. Wenn allerdings Fehler außerhalb des UX-Bus auftreten, sollte man auch hier die Möglichkeit des Loggings haben, um Probleme schnell zu beheben. Probleme können an allen zum Bus verbundenen Clients auftreten. Dazu gehört der UXBService und die implementierten Adapter.

Der UXBService nutzt das über die FirstSpirit API zur Verfügung gestellte log4j als Logging Mechanismus. Es wird die Logging Konfiguration genutzt, welche vom FirstSpirit Server ebenfalls genutzt wird. Um DEBUG Messages für den UXBridgeService zu erhalten, genügt es das DEBUG Logging für den FirstSpirit Server zu aktivieren. Wählen Sie dazu im Server-Monitoring des FirstSpirit-Servers den Punkt Konfiguration -> Protokollierung. Editieren Sie die aktive Konfiguration und ändern Sie die erste Zeile wie folgt.

```
log4j.rootCategory=DEBUG, fs
```

Nach der Speicherung der Konfiguration wird das Loglevel automatisch angepasst.

Adapter werden im Projekt umgesetzt, so dass es im Verantwortungsbereich des Entwicklers liegt, ein entsprechendes Logging vorzusehen und die Verwendung entsprechend zu dokumentieren.

3.3 Hochverfügbarkeit

Wird die UX-Bridge nur zur Befüllung von Content-Repositories verwendet, so werden mit der Standardkonfiguration oft schon die Kriterien bezüglich der Verfügbarkeit erfüllt. Ein eventueller Ausfall des UX-Bus hat keine direkten Auswirkungen auf die Webseite (Livesystem), da die darin enthalten Webapplikationen nur mit dem Content-Repository kommunizieren. Der Betrieb der Webseite ist also auch bei einem zeitweisen Ausfall gewährleistet. Während des Ausfalls können lediglich keine Daten von FirstSpirit in das Content-Repository geschrieben werden.

Kurze Ausfälle, z.B. durch Neustart einer Komponente, Stromausfall, o.ä., werden in der Standardkonfiguration durch Mechanismen wie automatische Reconnects, Redelivery von Nachrichten und dem persistenten Message Storage abgefangen. In diesen Fällen ist also kein Datenverlust zu erwarten.

Fallen Komponenten irreparabel aus, so kann durch ein Neuaufsetzen der Komponente und ein Volldeplyoment der Daten der ursprüngliche Zustand einfach wieder hergestellt werden.

Wird die UX-Bridge als Integrationskomponente genutzt, mit der auch andere Systeme über den Austausch von Nachrichten kommunizieren, sind die Anforderungen an die Verfügbarkeit in der Regel höher. In diesen Fällen kann der Einsatz von Master/Slave Konfigurationen zur Sicherstellung der Hochverfügbarkeit des UX-Buses sinnvoll sein. Bei extrem hohen Zahlen von Nachrichten oder sehr großen Nachrichten stellt die Konfiguration eines Loadbalancing eine weitere Option dar. Beide Mechanismen können natürlich auch in Kombination eingesetzt werden.

Werden in den Content-Repositories auch Daten gespeichert, die nicht von FirstSpirit erzeugt werden können, so sollte ein besonderes Augenmerk auf die Sicherung dieser Daten gelegt werden. Natürlich muss in einem Hochverfügbarkeitsszenario auch das Content-Repository entsprechend redundant ausgelegt sein, um nicht zum Single Point of Failure zu werden. Dies ist besonders kritisch, da ggf. die gesamte Website in ihrer Funktion beeinträchtigt ist.

In den folgenden Kapiteln wird auf die einzelnen Komponenten und Aspekte

eingegangen, die bei der Konfiguration eines Hochverfügbarkeitsszenarios zu betrachten sind. Dabei wird immer wieder der Begriff Broker verwendet, der die Nachrichtenkomponente des UX-Bus bezeichnet. Konkret wird Apache ActiveMQ eingesetzt.

3.3.1 Message Storage

Als Message Store des UX-Bus wird in der mitgelieferten Distribution KahaDB mit aktivierter Persistenz eingesetzt. Falls Sie auch für den Message Storage eine Desaster-Recovery-Strategie umsetzen wollen, finden Sie unter http://activemq.apache.org/replicated-message-store.html entsprechende Hinweise.

Detaillierte Hinweise zur Konfiguration des Message Storage finden Sie unter http://activemq.apache.org/persistence.html.

3.3.1.1 Backup des Message Storage

Bei der Auslieferung ist in der Konfiguration (\conf\activemq.xml) KahaDB zur Nachrichten-Persistenz konfiguriert:

```
<broker>
  <persistenceAdapter>
    <kahaDB directory="${activemq.data}/kahadb"/>
  </persistenceAdapter>
</broker>
```

Um ein Backup der KahaDB zu erstellen sind folgende Schritte notwendig:

1. Freeze des Dateisystems, das die Datenbank enthält, um zu gewährleisten, dass ein konsistenter Snapshot des Journals erstellt wird
2. Backup der Datenbank mit Standard Backup Mechanismen

Weitergehende Informationen finden sich unter:

http://activemq.apache.org/how-do-i-back-up-kahadb.html
http://www.mentby.com/Group/apache-activemq/kahadb-and-backups.html

Wird alternativ zur Kaha Persistenz eine JDBC Persistenz der JMS Nachrichten benutzt (http://activemq.apache.org/persistence.html), so sind die bekannten datenbankspezifischen Backup-Mechanismen zu nutzen.

3.3.2 Failover

Bei einem Ausfall des UX-Bus sollte sichergestellt sein, dass die Arbeit des UXBServices und der Adapter nicht beeinträchtigt werden. Das heißt, dass das Versenden der Nachrichten entweder zu einem späteren Zeitpunkt erneut versucht wird oder der Versand direkt über eine zweite Instanz erfolgt.

Alle am jeweiligen UX-Bus angemeldeten Clients sollten die Möglichkeit haben, sich zu einem anderen UX-Bus neu zu verbinden oder einen erneuten Verbindungsaufbau nach einer gewissen Wartezeit zu versuchen. Um dies zu erreichen, gibt es das failover Protokoll.

In der Standardkonfiguration des UXBService ist das Failover-Protokoll bereits konfiguriert:

```
<property name="brokerURL"
value="failover:(tcp://localhost:61616)"/>
```

In diesem Fall sorgt das Failover-Protokoll dafür, dass bei einem Verbindungsverlust in bestimmten Abständen versucht wird eine neue Verbindung aufzubauen. Hier kann man die verschiedenen Parameter selbst bestimmen, z.B. wie lange das erste Mal gewartet werden soll, um welchen Faktor sich die Wartezeit bei jedem Versuch erhöht und wie lange die maximale Wartezeit sein soll (initialReconnectDelay, backOffMultiplier, maxReconnectDelay).

Eine Failover-Konfiguration mit mehreren UX-Bus Instanzen würde z.B. folgendermaßen aussehen:

```
<property name="brokerURL" value="failover:(tcp://broker1:61616,
tcp://broker2:61616)?randomize=false"/>
```

In diesem Fall wird der broker1 zum Versenden der Nachrichten verwendet. Fällt dieser aus, werden die Nachrichten automatisch über broker2 gesendet.

Beim Implementieren eigener Adapter ist darauf zu achten, dass dort ebenfalls das Failover-Protokoll eingesetzt wird, um auch hier vor Verbindungsabbrüchen oder Ausfällen des UX-Bus geschützt zu sein.

Den größten Vorteil des Failover-Protokolls hat man, wenn man mehrere UX-Bus Instanzen in einem Cluster konfiguriert. (siehe auch Kapitel 3.3.3 Master/Slave Betrieb, Seite 30) Man kann nun mehrere UX-Bus Instanzen angeben mit denen sich das Failover-Protokoll verbinden kann. Wenn die Verbindung zu einem UX-Bus unterbrochen wird, wird automatisch eine Verbindung zu einem anderen UX-Bus hergestellt. Bei zeitkritischen Anwendungen kann zusätzlich mit den Parametern backup und backupPoolSize eine bestimmte Anzahl an Backup Verbindungen

hergestellt werden, welche sofort bei einem Abbruch einer aktiven Verbindung genutzt werden können. Der Vorteil ist, dass die Verbindung schon in einem Standby Zustand besteht und im Notfall nicht erst hergestellt werden muss.

Ein alternative Möglichkeit, um Failover beim Clustering zu nutzen, ist nur einen UX-Bus anzugeben und in der Konfiguration des UX-Bus zu veranlassen, dass die angemeldeten Clients automatisch eine Liste aller verfügbaren UX-Bus erhalten, zu denen sie bei Problemen eine Verbindung aufbauen können. Dafür gibt es die Eigenschaften „updateClusterClients“ und „rebalanceClusterClients“ in der „TransportConnector“-Konfiguration des UX-Bus.

Nähere Informationen zu den Failover-Parametern können Sie in der Apache ActiveMQ Dokumentation finden: http://activemq.apache.org/failover-transport-reference.html

3.3.3 Master/Slave Betrieb

Um die Ausfallsicherheit zu erhöhen, bietet sich der Master/Slave Betrieb an. Dabei fungiert eine Apache ActiveMQ Instanz als Master und eine oder mehrere Instanzen als Slave. Die Clients senden ihre Anfragen immer an den Master, erst bei einem Ausfall des Masters wird eine der Slave Instanzen zum neuen Master und übernimmt dessen Aufgaben. Die Nachrichten werden nun von dieser Instanz bearbeitet.

Dem gegenüber steht der Einsatz eines Network of Brokers (Kapitel 3.3.4 Network of Brokers Seite 31), bei dem alle Apache ActiveMQ Instanzen gleichberechtigt sind.

In beiden Fällen ist es wichtig das Failover im Client, wie in Kapitel 3.3.2 Failover auf Seite 29 beschrieben, zu konfigurieren.

3.3.3.1 Shared storage Master/Slave

Beim Clustering-Modus „Shared storage Master/Slave“ wird dieselbe KahaDB zum persistieren der Nachrichten von mehreren Brokern genutzt. Es kann immer nur ein Master auf die Persistenz zugreifen, während die Slaves geblockt werden. Die Slaves prüfen in regelmäßigen Abständen ob sie Zugriff auf die Persistenz bekommen. Wenn der Master Broker nun ausfällt und herunterfährt; bekommt der Slave Broker den Zugriff auf die Persistenz. Erst in diesem Moment fährt der Slave Broker komplett hoch und erlaubt es Clients eine Verbindung aufzubauen.

Es gibt 2 Arten diese geteilte Persistenz zu realisieren:

Eine Möglichkeit ist es, eine relationale Datenbank zu nutzen, in welcher die Broker die Daten speichern. Ein Vorteil dieser Methode ist, dass Sicherungen von Datenbanken schnell und einfach erstellt werden können.

Die zweite Möglichkeit ist, auf ein gemeinsam genutztes Dateisystem zuzugreifen, auf welchem die Apache ActiveMQ Broker dann die KahaDB ablegen. Die einzige Anforderung an dieses Dateisystem ist, dass es einen Locking Mechanismus gibt, welcher sicherstellt, dass immer nur ein Broker auf die KahaDB zugreifen kann.

Um ein „shared storage Master/Slave“-Clustering zu konfigurieren genügt es zwei Broker zu konfigurieren, welche auf dieselbe Persistenz zugreifen. Die Broker sollten aufgrund der Ausfallsicherheit auf unterschiedlichen Maschinen laufen. Die Clients bekommen beide Brokeradressen im Failover-Protokoll übergeben, und wenn ein Master-Broker ausfällt, wird sofort eine Verbindung zum neuen Master-Broker hergestellt. Dieser greift auf die Daten derselben Persistenz zu und ist somit aktuell. In jedem der Broker muss der Persistence-Adapter so angepasst werden, dass sie alle dasselbe Verzeichnis verwenden.

```
<persistenceAdapter>
    <kahaDB directory="/sharedFileSystem/sharedBrokerData"/>
</persistenceAdapter>
```

Näherer Informationen zur Konfiguration finden sie in der offiziellen Dokumentation unter http://activemq.apache.org/shared-file-system-master-slave.html.

Man sollte beachten, dass dieser Modus anfällig für Fehler in Verbindung mit dem Dateisystem ist. Beim Ausfall der Maschine, auf welchem sich das Dateisystem befindet, haben alle Broker, die darauf zugreifen sollen, keinen Zugriff mehr auf die Persistenz und müssen im schlimmsten Fall heruntergefahren werden. Wenn die eigene Anwendung vor Fehlern in Verbindung mit dem Dateisystem geschützt werden soll, sollte ein Dateisystem gewählt werden, welches ebenfalls auf mehrere Maschinen verteilt wird und welches beim Ausfall einer Maschine weiterhin funktionsfähig ist.

3.3.4 Network of Brokers

Ein „Network of Brokers“ besteht aus mindestens zwei verschiedenen Brokern, welche jeweils einen eigenen Persistenzstore haben. In diesem Fall ist es nicht zwingend notwendig, dass jeder Broker die gleichen Nachrichten (bzw. alle Nachrichten) vorliegen hat. Ein Anwendungsfall für diese Architektur ist die Lastenverteilung.

Der Einsatz eines Netzwerkes von Brokern für die Verteilung der Last wird in den meisten Einsatzszenarien nicht nötig sein. Ein einzelner UX-Bus kann in der Regel das zu erwartete Nachrichtenaufkommen problemlos verarbeiten.

Wird trotzdem ein Loadbalancing gewünscht, so würde eine Client-Konfiguration z.B. folgendermaßen aussehen:

```
failover:(tcp://master1.IP:61616,tcp://master2.IP:61617)?randomize
=true
```

Bei master1 und master2 handelt es sich um ein „Network of Brokers“, die Verteilung der Nachrichten erfolgt nach dem Zufallsprinzip.

Diese Architektur lässt sich am besten an einem Beispiel verdeutlichen:

Broker A bekommt eine Nachricht in der Queue A und speichert sie im eigenen Store. Solange am Broker B kein Client nach Nachrichten aus Queue A lauscht, werden diese Nachrichten auf Broker B nicht benötigt, weshalb er sie auch nicht bekommt. In dem Moment, wo ein Client von Broker B nach Nachrichten aus der Queue A lauscht, wird von Apache ActiveMQ sichergestellt, dass auch Broker B diese Nachrichten erhält, um sie an den Client auszuliefern.

Die Informationen, welche Clients auf welchen Queues an welchen Brokern lauschen, werden durch so genannte „Advisory Messages“ unter den verschiedenen Brokern ausgetauscht. So weiß jeder Broker zu jeder Zeit, welche Nachrichten für andere Broker interessant sind und welche nicht.

In der Konfiguration einzelner Broker kann man auch Filter für verschiedene Nachrichten definieren oder andere Broker von Nachrichten ausschließen.

Aus Sicht des Client handelt es sich hier um eine Hochverfügbarkeit der Broker, nicht aber der Nachrichten die verschickt werden. Grund dafür ist die Tatsache, dass jeder Broker eine eigene Datenbank für die Nachrichten hat. Fällt ein Broker aus, werden die Nachrichten dieses Brokers erst wieder verteilt, wenn der Broker neu gestartet wurde.

Weiterführend Informationen zu diesem Thema finden sie unter:

http://activemq.apache.org/networks-of-brokers.html und
http://activemq.apache.org/how-do-distributed-queues-work.html

3.3.5 Einsatz mit FirstSpirit GenerationServer

Durch den Einsatz eines oder mehrerer GenerationServer kann die Generierungslast

vom FirstSpirit Master Server entkoppelt werden. Der GenerationServer kann neben der Generierung der statischen Seiten, auch die Generierung der Nachrichten übernehmen. Das senden der Nachrichten geschieht weiterhin über den UX-Bus Service, der auf dem FirstSpirit Master Server läuft.

3.3.6 Hochverfügbarkeit der Content-Repositories

Um Hochverfügbarkeit zu gewährleisten, ist neben der korrekten Konfiguration der UX-Bridge Infrastruktur, ebenfalls die Konfiguration des Content Repositories entscheidend.

Die Wahl des Content Repositories wird projektspezifisch getroffen. Konsultieren sie bitte die Dokumentation des verwendeten Repositories, um mehr über Hochverfügbarkeit zu erfahren.

3.3.7 Hochverfügbarkeit der Adapter

Adapter werden projektspezifisch implementiert. Bei der Entwicklung müssen die Anforderungen bzgl. Hochverfügbarkeit entsprechend berücksichtigt werden. In vielen Szenarien können die Anforderungen durch mehrere Instanzen des Adapters und ein entsprechendes Routing abgedeckt werden.

3.4 Security

In der Standardkonfiguration vom UX-Bus ist es jeder Anwendung erlaubt, sich zum Bus zu verbinden, um Nachrichten zu senden und zu empfangen. Je nach Einsatzgebiet der UX-Bridge und der abhängigen Anwendungen mag dies nicht erwünscht sein, vor allem wenn sensible Daten übertragen werden.

Der UX-Bus sollte nicht von außen erreichbar sein, sondern die Nachrichten nur über ein internes Netzwerk verschicken, damit eine möglichst hohe Sicherheit gewährleistet werden kann. Hierbei sollte darauf geachtet werden, dass Verbindungen nach außen und von außen geblockt werden. Es wird empfohlen, dies über entsprechende Firewall-Regeln auf den Rechnern sicherzustellen. Alternativ oder zusätzlich kann dies über Security-Einstellungen im UX-Bus erreicht werden. Eine Möglichkeit ist die Verwendung der unten erwähnten "messageAuthorizationPolicy. Nähere Details sind in der Apache ActiveMQ Dokumentation zu finden (http://activemq.apache.org/security.html).

Der UX-Bus kann vor nicht authentifiziertem und nicht autorisiertem Zugriff geschützt werden. Für einfache Anwendungsfälle ist es ausreichend, das „simple authentication plugin“ von Apache ActiveMQ zu nutzen, das es erlaubt, die Zugangsdaten in einer Konfigurationsdatei abzulegen. Das Apache ActiveMQ „JAAS plug-in“ ermöglicht es darüber hinaus, die standardisierten, einfach zu konfigurierenden Java Login Module zu nutzen, die eine Authentifizierung über verschiedene Quellen erlauben, wie z.B. LDAP, Property-Dateien, usw.. Zusätzlich können auch eigene JAAS Login-Module geschrieben werden, die zur Authentifizierung oder Autorisierung Mechanismen wie Kerberos, NTLM, NIS, usw. nutzen. Es ist auch möglich, den ActiveMQ-Broker mit zertifikat-basierter Verschlüsselung (z.B. SSL) zu betreiben.

Soll eine noch feingranularere Kontrolle eingerichtet werden, so bietet ActiveMQ die Möglichkeiten der „operation-level“-Authentifizierung und der „message-level“-Authentifizierung. Über die „operation-level“-Authentifizierung kann z.B. festgelegt werden, welcher Benutzer in welche/von welcher Destination schreiben/lesen kann. Im Gegensatz zur Authentifizierung und Autorisierung auf Broker-Level ermöglicht die „message-level“-Autorisierung es, nur bestimme Nachrichten in der Destination zuzulassen z.B. nur Nachrichten für einen bestimmten Empfänger.

Über die Apache ActiveMQ plug-in API ist es außerdem möglich neue Security Plugins zu schreiben, die an die eigenen Bedürfnisse angepasst sind.

Eine weiterführende Dokumentation ist unter http://activemq.apache.org/security.html zu finden.
Ebenfalls empfehlenswert ist auch die Fuse MQ Enterprise Doku: http://fusesource.com/docs/mqent/7.0/security/front.html
Folgende Stichworte können bei der Suche helfen:
simpleAuthenticationPlugin, jaasAuthenticationPlugin, authorizationPlugin, messageAuthorizationPolicy, jaasCertificateAuthenticationPlugin

Ein weiterer, sicherheitsrelevanter Punkt ist der Zugriff über JMX. Dieser ist bei ActiveMQ standardmäßig geschützt und kann über die Dateien “jmx.access” und “jmx.password” konfiguriert werden (http://activemq.apache.org/jmx.html).

3.5 Monitoring

Dieses Kapitel behandelt verschiedene Möglichkeiten des Monitorings von Apache ActiveMQ. Da es in High-End Applikationen immer auf eine gute Performance und geringe Fehleranfälligkeit ankommt, ist es wichtig Engpässe schnell und zuverlässig lokalisieren zu können und sich schnell ein zuverlässiges Bild von dem aktuellen

Zustand der Infrastruktur machen zu können, falls Probleme auftreten.

3.5.1 Apache ActiveMQ Webconsole

Apache ActiveMQ bringt bereits von Haus aus Möglichkeiten zum Administrieren und Monitoren der Anwendung mit. So gibt es die Webconsole, welche unter http://localhost:8161/admin erreichbar ist, sobald sie aktiviert ist. "localhost" muss gegebenenfalls durch die Adresse der Maschine, auf welcher der UX-Bus läuft, ersetzt werden.

Details zur Aktivierung und Konfiguration der Webconsole sind in der Online Dokumentation zu finden: http://activemq.apache.org/web-console.html

Es kommt vor allem darauf an, dass nicht jeder uneingeschränkten Zugriff auf die Webconsole hat. Deshalb sollte in der conf/jetty.xml Datei die Zeile

```
<property name="authenticate" value="false" />
```

durch die Zeile

```
<property name="authenticate" value="true" />
```

ersetzt werden.

Die berechtigten User und deren Zugangsdaten können dann in der Datei conf/jetty-realm.properties eingetragen werden (der Standarduser ist „admin“ mit Username/Password: admin/admin).

Die Webconsole bietet einfache Funktionen, um Informationen über den Broker zu betrachten und einfache Statistiken angezeigt zu bekommen. Es werden unter anderem Informationen über Queues, Topics, Subscriber, Connections und das Netzwerk angezeigt. Außerdem gibt es eine Funktion zum Senden von Nachrichten. Dies ermöglicht einfaches Testen von Anwendungen ohne vorher extra Code zu schreiben. Die Webconsole lässt sich intuitiv bedienen, ist selbsterklärend und wird deshalb an dieser Stelle nicht näher erläutert.

3.5.2 Apache ActiveMQ Kommandozeilentools

Mit Apache ActiveMQ werden außerdem einige nützliche Kommandozeilen Tools ausgeliefert, welche grundlegende Admin-Funktionen bereitstellen, unter anderem auch einige Funktionen zum Monitoring. Mit dem Tool "activemq-admin" lassen sich zum Beispiel mit dem Parameter "query" verschiedenste Informationen zu Broker, Destination, Connector und Connection abfragen und auslesen. Mithilfe des

Parameters "browse" können Nachrichten untersucht werden.

Eine genaue Beschreibung des Funktionsumfangs aller Möglichkeiten ist in der offiziellen Apache ActiveMQ Dokumentation zu finden: http://activemq.apache.org/activemq-command-line-tools-reference.html

3.5.3 Monitoring mit Hilfe von JMX

Eine andere Möglichkeit des Monitorings ist der Gebrauch der JMX-API. (Java Management Extensions). Einige Parameter der oben erwähnten Kommandozeilentools (z.B. "query") nutzen ebenfalls die JMX-API um Informationen über Broker, Queues, usw. zu erhalten. Details zur Konfiguration von JMX entnehmen Sie bitte der Apache ActiveMQ Dokumentation: http://activemq.apache.org/jmx.html

Wenn der Apache ActiveMQ Broker so konfiguriert ist, dass er JMX Verbindungen zulässt, kann man beispielsweise eigene Hilfsprogramme schreiben, welche eine JMX Verbindung herstellen und Brokerinformationen auslesen.

Hierzu wird auf die offizielle JMX API (http://docs.oracle.com/javase/7/docs/technotes/guides/jmx/spec.html) und die Apache ActiveMQ API, welche zusätzliche Funktionalität bereitstellt (http://activemq.apache.org/maven/5.6.0/activemq-core/apidocs/) verwiesen. Das Interface "BrokerViewMBean" aus der Apache ActiveMQ API ist ein guter Einstiegspunkt, um Daten des Brokers über JMX auszulesen.

Die MBeans von Apache ActiveMQ liegen in dem Package „org.apache.activemq“ und können mit Tools wie der JConsole, JVisualVM oder ähnlichen ausgelesen werden. Informationen zu Apache Camel und den Routen finden sie in dem MBean org.apache.camel.

Über das JMX Monitoring erhält man Informationen wie

- Wie viele Nachrichten wurden verschickt
- Wie viele Nachrichten wurde geroutet oder sind verfallen
- Wie viele Verbindungen (UXBService, Adapter) gibt es zum UX-Bus

Im Fehlerfall können dadurch über das Monitoring wertvolle Informationen erlangt werden.

3.5.4 Monitoring mit Hilfe von Advisory Messages

Advisory Messages werden von Apache ActiveMQ genutzt um die Kommunikation von Brokern untereinander zu ermöglichen und Informationen über Queues, Consumer, usw. auszutauschen. Um diese Nachrichten für eigene Monitoring Zwecke zu nutzen, kann zum Beispiel eine eigene Anwendung geschrieben werden, welche die Advisory Messages mithört. Details hierzu sind in der Apache ActiveMQ Dokumentation enthalten: http://activemq.apache.org/advisory-message.html

3.5.5 Monitoring im Auftrag

Der UXB-Service stellt selbst auch eine Möglichkeit zum Monitoring zur Verfügung. Jede Nachricht, die an den UX-Bus gesendet wird, erhält bei der Erstellung einen Zeitstempel. Wenn ein Adapter die Nachricht verarbeitet, wird ein weiterer Zeitstempel gesetzt, sowie der Status nach der Verarbeitung (ok/fail). Im Auftrag kann eine Methode aufgerufen werden, die den UXB-Service abfragt und eine Liste von Nachrichten, deren Durchlaufzeit und deren Status zurück liefert. Dazu muss dem Auftrag am Ende eine Script-Aktion hinzugefügt werden:

```
#! executable-class

com.espirit.moddev.uxbridge.inline.UxbResultHandler
```

Da die Verarbeitung der Nachrichten asynchron erfolgt und sie je nach Adapter und Anzahl der Nachrichten unterschiedlich lange dauern kann, ist es möglich über einen Scriptparameter (messageTimeout) die Zeit in Sekunden anzugeben, die der Service maximal auf die Antwortnachrichten warten soll, bevor die Durchlaufzeiten ausgewertet werden.

```
…
INFO 25.07.2012 10:38:44.811 Time for
#uxb/pressreleasedetails/UXB/DE/704 (mongodb): 220ms

INFO 25.07.2012 10:38:44.812 Time for
#uxb/pressreleasedetails/UXB/EN/704 (mongodb): 173ms

INFO 25.07.2012 10:38:44.813 48/48 deployed successfully (overall:
210ms, monogodb: 183ms, postgres: 238ms).

INFO 25.07.2012 10:38:44:813 finished task 'GetTimings' – schedule
entry 'UX-Bridge (News)' (id=6191)
```

Die Ergebnisse der Nachrichten, die erst nach der spezifizierten Wartezeit abgearbeitet wurden, werden im Speicher gehalten. Die Dauer bis diese automatisch gelöscht werden, kann in den Einstellungen des Services (s. Kapitel 2.2.2.2 Optionen Reiter) konfiguriert werden.

3.6 Änderung des Routings

Soll das Routing geändert werden, weil zum Beispiel neue Adapter oder Content-Repositories hinzugekommen sind, muss die Konfiguration wie in Kapitel 3.1 Routing Seite 23 beschrieben angepasst werden.

Nach den Änderungen muss der UX-Bus neugestartet werden, damit die neuen oder geänderten Routen aktiviert werden. Hierbei sollte beachtet werden, dass ein Neustart des UX-Busses einen Verlust von Nachrichten zur Folge haben kann.

Wenn während des Neustarts ein Deployment läuft und die Zeit des UX-Bus Neustarts länger dauert, als die Zeit, die der Failover Transport benötigt, um die maximale Anzahl an Reconnect Versuchen zu erreichen, dann bricht das Deployment des entsprechenden Content Elements ab und es wird fortgefahren. Dieses Element wird dann später nicht im Content Repository vorhanden sein. Um diese Gefahr zu minimieren, empfiehlt es sich, durch vorherige Tests den Failover Transport so zu konfigurieren, dass ein normaler Neustart des UX-Buses ohne Datenverlust durchgeführt werden kann.

In der Standardkonfiguration werden die default-Parameter von Failover verwendet. Diese sind in der Online Dokumentation beschrieben: http://activemq.apache.org/failover-transport-reference.html

Die Parameter maxReconnectAttempts und startupMaxReconnectAttempts wurden auf die folgenden Werte gesetzt:

```
maxReconnectAttempts=10
startupMaxReconnectAttempts=2
```

Die erste Wartezeit beträgt 10ms und wird 10 Versuche lang verdoppelt. Also ergibt sich eine Gesamtwartezeit von:

10+20+40+80+160+320+640+1280+2560+5120=10230ms=10,23s

Wenn der Server also länger als 10s nicht verfügbar ist, geht ein Datensatz verloren.

Während der UX-Bus neu startet, ist ein Neustart des UXB-Services nicht nötig, da der Failover Transport nach dem Neustart des UX-Bus automatisch die Verbindung wiederherstellt. Es wird empfohlen, in selbst implementierten Adaptern ebenfalls den Failover Transport einzusetzen, damit die Verbindung bei Abbrüchen wieder aufgebaut wird und der Adapter nicht neu gestartet werden muss.

3.7 Backup

Da die UX-Bridge aus unterschiedlichen Komponenten besteht, muss das Backup abhängig von der Komponente durchgeführt werden.

3.7.1 Backup des FirstSpirit Projekts

Hinweise, wie FirstSpirit Projekte gesichert und wiederhergestellt werden können, sind in der FirstSpirit Admin Dokumentation zu finden.

3.7.2 Backup des Message Brokers

Auf dem Message Broker befinden sich meistens nicht allzu viele Nachrichten. Sobald eine Nachricht, welche in einer Queue liegt, abgerufen wird, verlässt diese die Queue. Der Message Broker übergibt die Verantwortlichkeit, was mit der Nachricht weiter geschieht, dem Adapter. Die Nachrichten, welche noch nicht vom Adapter „abgeholt“ wurden, liegen auf dem Broker. Wenn es gewünscht ist, diese zu sichern, kann das durch ein Backup der KahaDB geschehen, falls die Nachrichten und die KahaDB persistiert werden. Details hierzu sind im Kapitel 3.3.1.1 Backup zu finden.

3.7.3 Backup des Content Repositories

Das Vorgehen, um die Daten in dem Content Repository zu sichern, ist abhängig von der eingesetzten Lösung. Je nach eingesetztem System sollten die Hinweise aus der entsprechenden Dokumentation befolgt werden, um ein Backup zu erstellen.

3.8 Desaster-Recovery

Nach Systemausfällen oder kritischen Fehlern können Daten beschädigt werden oder verloren gehen. Im Notfall gilt es, diese Daten möglichst schnell und zuverlässig wiederherzustellen.

Bei Ausfällen, in denen Nachrichten verloren gegangen sind, ist es am einfachsten aus FirstSpirit heraus ein Volldeployment durchzuführen, wobei der aktuelle Stand des Projekts von der UX-Bridge in das Content-Repositories geschrieben wird. So kann man sich sicher sein, konsistente, aktuelle und fehlerfreie Daten im Repository zu haben, mit denen weitere Anwendungen wieder arbeiten können.

Sollte es, aus welchem Grund auch immer, nicht möglich sein, die Daten durch ein

Volldeployment zu erneuern, dann müssen die Daten manuell wiederhergestellt werden.

Hierbei ist das Vorgehen abhängig von der ausgefallenen Komponente.

3.8.1 Fehler in FirstSpirit

Sollte für den FirstSpirit Server eine Desaster-Recovery notwendig sein, wenden Sie sich bitte an den Helpdesk der e-Spirit AG, um entsprechende Instruktionen zu erhalten.

3.8.2 Fehler im UX-Bus

Wenn der Fehler in der Bus-Infrastruktur auftritt und diese beschädigt wurde, ist es in den meisten Fällen nicht notwendig, gesicherte Daten manuell wiederherzustellen. Im schlimmsten Fall gehen höchstens die Nachrichten verloren, welche noch nicht auf dem Bus verarbeitet wurden und auch nicht persistiert wurden. In der Standardkonfiguration ist die Persistenz allerdings aktiviert.

Auf Seite der Apache ActiveMQ wird die KahaDB genutzt, um Nachrichten zu speichern und zu verarbeiten. Deshalb gilt es im Falle eines Datenverlustes, die KahaDB wiederherzustellen. Das ist wiederum nur möglich, wenn der Broker so konfiguriert ist, dass die KahaDB persistiert wird. Ansonsten sind die Daten, welche sich zum Zeitpunkt des Crashs auf dem UX-Bus befanden, verloren. Im Normalfall sorgt Apache ActiveMQ selbst dafür, dass die persistierten Daten beim neuen Starten der Anwendung wiederhergestellt werden. Bei eigenen Konfigurationen, zum Beispiel in Verbindung mit einer relationalen Datenbank oder der KahaDB auf einem verteilten Dateisystem, können die Sicherungsmechanismen, die für die jeweilige Technik üblich sind, eingesetzt werden. Außerdem sei auf das Kapitel 3.2 der vorliegenden Dokumentation hingewiesen, welches sich mit dem Thema Hochverfügbarkeit beschäftigt.

3.8.3 Fehler im Adapter

Wenn der Adapter abgestürzt ist, sollten keine großen Probleme auftreten, denn ein Datenverlust ist nur in Extremsituationen zu erwarten. Im Normalfall bleiben die Nachrichten einfach so lange auf dem Broker, bis der Adapter wieder hochgefahren ist und diese Nachrichten abholt.

Lediglich die zur Zeit des Crashs verarbeitete Nachricht ist wahrscheinlich, abhängig von der Implementierung des Adapters, verloren. In diesem Fall kann in FirstSpirit

ein erneutes Deployment der Daten durchgeführt werden.

3.8.4 Fehler im externen Content Repository

Wenn Daten im externen Repository wiederhergestellt werden müssen, müssen dort die jeweils notwendigen Schritte durchgeführt werden. Diese unterscheiden sich je nach eingesetzter Anwendung. Einzelheiten sollten in der jeweiligen Dokumentation nachgeschlagen werden. Ein typischer Anwendungsfall könnte das Wiederherstellen einer Sicherung einer relationalen Datenbank sein.

3.9 Fehler-Analyse

Dieser Abschnitt soll den Administrator bzw. Entwickler bei der Fehlersuche und Fehlerbehebung unterstützen.

Die UX-Bridge besteht aus mehreren Komponenten, von der jede die Ursache für eine Störung sein kann.

Die folgende Checkliste kann abgearbeitet werden, wenn Inhalte nicht wie erwartet auf der Webseite erscheinen bzw. in das Content-Repository überführt werden.

1. FirstSpirit Auftrag prüfen, der das Deployment vornimmt
2. Prüfen, ob die Nachrichten richtig geroutet werden
3. Prüfen, ob der Adapter richtig funktioniert
4. Logs des Content-Repositories prüfen

3.9.1 FirstSpirit Auftrag prüfen

3.9.1.1 Fehler in UX-Bridge Aufgabe im Auftrag

3.9.1.1.1 Symptom

Die Aufgabe wird mit einer Fehlermeldung abgeschlossen und die Inhalte wurden nicht in das Content-Repository überführt.

Beispiel der Fehlermeldung:

```
INFO  18.07.2012 12:29:31.180 {seID=6191}
```

```
(de.espirit.firstspirit.server.scheduler.ScheduleManagerImpl):
starting task 'UX-Bridge – Activate Generation' - schedule entry
'UX-Bridge (News)' (id=6191)

ERROR 18.07.2012 12:29:31.185 {seID=6191}
(de.espirit.firstspirit.server.scheduler.ScriptTaskExecutor):
error during script execution :
de.espirit.firstspirit.access.ServiceNotFoundException: Service
com.espirit.moddev.uxbridge.api.v1.service.UxbService' not found

de.espirit.firstspirit.access.ServiceNotFoundException: Service
com.espirit.moddev.uxbridge.api.v1.service.UxbService' not found

at

…
```

3.9.1.1.2 Ursache

Der UX-Bridge Dienst ist nicht gestartet.

3.9.1.1.3 Lösung

Starten Sie den UX-Bridge Dienst in den Servereigenschaften und stellen Sie sicher, dass der Autostart für diesen Dienst aktiviert ist.

3.9.1.2 Fehler im UX-Bridge Generierungsauftrag

3.9.1.2.1 Symptom

Der Generierungsauftrag für den Ausgabekanal der UX-Bridge wird mit Fehlern abgeschlossen.

3.9.1.2.2 Ursache

Hier sind zwei Ursachen denkbar:

1. Es handelt sich um Fehler in den Vorlagen, die vom Vorlagenentwickler behoben werden müssen
2. Es tauchen Fehlermeldungen folgender Art auf:

```
INFO  18.07.2012 12:40:45.943 {seID=6191}
(com.espirit.moddev.uxbridge.service.UxbServiceImpl): UXB Service:
calling Producer
```

```
ERROR 18.07.2012 12:40:51.098 {seID=6191}
(com.espirit.moddev.uxbridge.service.UxbServiceImpl): Could not
connect to broker!
```

In diesem Fall konnte die Verbindung zum UX-Bus nicht aufgebaut werden.

3.9.1.2.3 Lösung

- Stellen Sie sicher, dass der UX-Bus verfügbar ist. Starten Sie danach den UX-Bridge Dienst neu. Optimaler Weise sollte der UX-Bus vor dem FirstSpirit-Server gestartet werden.
- Treten die Fehlermeldungen trotz aktiver UX-Bus Instanz auf, so prüfen Sie bitte die Konfiguration des UXB-Services. Ist der Hostname/IP des UX-Bus korrekt? Korrigieren Sie ggf. die Konfiguration und starten Sie den UX-Bridge Dienst (UXBService) neu.
- Prüfen Sie, ob die Kommunikation eventuell durch eine Firewall blockiert wird.

3.9.1.3 In der UX-Bridge Statistics Report Aufgabe treten Warnungen auf

3.9.1.3.1 Symptom

Die Aufgabe UX-Bridge Statistics Report im Auftrag wird mit Warnungen abgeschlossen.

Beispiel:

```
INFO  18.07.2012 12:47:14.694 {seID=6191}
(de.espirit.firstspirit.server.scheduler.ScheduleManagerImpl):
starting task 'UX-Bridge Statistics Report' - schedule entry 'UX-
Bridge (News)' (id=6191)

WARN  18.07.2012 12:47:14.736 {seID=6191}
(com.espirit.moddev.uxbridge.service.UxbServiceImpl): Deployment
for target expired: #uxb/pressreleasesdetails/UXB/EN/256
(postgres)

INFO  18.07.2012 12:47:14.736 {seID=6191}
(com.espirit.moddev.uxbridge.service.UxbServiceImpl): Time for
#uxb/pressreleasesdetails/UXB/EN/704 (mongodb): 580ms
```

3.9.1.3.2 Ursache

Hier sind zwei mögliche Ursachen denkbar:

1. Der Adapter konnte die Nachricht nicht richtig verarbeiten, so dass er keine Nachricht über die erfolgte Aktualisierung des Content-Repositories gesendet hat.

2. Der Adapter konnte die Nachricht verarbeiten, die Nachricht über die erfolgte Aktualisierung wurde aber nicht innerhalb des definierten Zeitraumes verschickt. Siehe dazu auch Kapitel 3.9.5.2 False positives in UX-Bridge Statistics Report Aufgabe S. 46.

3.9.1.3.3 Lösung

Prüfen Sie, ob der Adapter die Nachricht richtig geroutet und verarbeitet hat. Siehe dazu Kapitel 3.9.3 Prüfen ob die Nachrichten richtig geroutet werden S. 44.

3.9.2 Prüfen ob die Nachrichten richtig geroutet werden

Der UX-Bus loggt in der Standardkonfiguration keinerlei Informationen bzgl. des Routings von Nachrichten.

Im Kapitel 3.2 Logging, ab Seite 24, erhalten Sie Informationen, wie Sie ein entsprechendes Logging konfigurieren können.

3.9.3 Prüfen ob Adapter richtig funktioniert

Es ist möglich, dass der Adapter die Nachrichten nicht richtig verarbeiten kann. Der Entwickler des Adapters sollte in seine Implementierung entsprechende Logausgaben schreiben, damit solche Probleme im laufenden Betrieb erkannt werden können.

3.9.4 Das Content-Repository prüfen

Je nach verwendetem Content-Repository können dessen Logfiles und eventuell vorhandene Administrationstools bei der Fehlersuche unterstützen. Da diese Komponenten projektspezifisch gewählt werden, können wir an dieser Stelle keine zusätzlichen Hinweise geben.

3.9.5 Sonstige Fehler

3.9.5.1 FirstSpirit Server meldet, dass keine JMS Verbindung zum UX-Bus aufgebaut werden konnte

3.9.5.1.1 Symptom

Im fs-server.log sind Fehlermeldungen zu finden, dass keine JMS-Verbindung zum UX-Bus aufgebaut werden konnte.

Beispiel:

```
WARN  17.07.2012 16:30:46.483
(org.apache.camel.component.jms.DefaultJmsMessageListenerContainer
): Setup of JMS message listener invoker failed for destination
'FS_IN' - trying to recover. Cause: java.io.EOFException

WARN  17.07.2012 16:30:46.486
(org.apache.camel.component.jms.DefaultJmsMessageListenerContainer
): Setup of JMS message listener invoker failed for destination
Adapter-Statistics-Response-Route' - trying to recover. Cause:
java.io.EOFException

WARN  17.07.2012 16:30:46.488
(org.apache.camel.component.jms.DefaultJmsMessageListenerContainer
): Could not refresh JMS Connection for destination 'ROUTEIN' -
retrying in 5000 ms. Cause: Could not connect to broker URL:
tcp://localhost:61616. Reason: java.net.ConnectException:
Connection refused

WARN  17.07.2012 16:30:46.489
(org.apache.camel.component.jms.DefaultJmsMessageListenerContainer
): Could not refresh JMS Connection for destination Adapter-
Statistics-Response-Route' - retrying in 5000 ms. Cause: Could not
connect to broker URL: tcp://localhost:61616. Reason:
java.net.ConnectException: Connection refused

WARN  17.07.2012 16:30:46.489
(org.apache.activemq.transport.failover.FailoverTransport):
Transport (tcp://127.0.0.1:61616) failed, reason:
java.io.EOFException, attempting to automatically reconnect

ERROR 17.07.2012 16:30:51.666
(org.apache.activemq.transport.failover.FailoverTransport): Failed
to connect to [tcp://localhost:61616] after: 10 attempt(s)
```

3.9.5.1.2 Ursache

Beim Start des FirstSpirit-Servers war der UX-Bus nicht erreichbar oder die Verbindung ist im laufenden Betrieb abgebrochen.

3.9.5.1.3 Lösung

- Stellen Sie sicher, dass der UX-Bus verfügbar ist. Starten Sie danach den UX-Bridge Dienst (UXBService) neu. Optimaler Weise sollte der UX-Bus vor dem FirstSpirit-Server gestartet werden.
- Treten die Fehlermeldungen trotz aktiver UX-Bus Instanz auf, so prüfen Sie bitte die Konfiguration des UXB-Services. Ist der Hostname/IP des UX-Bus korrekt? Korrigieren Sie ggf. die Konfiguration und starten Sie den UX-Bridge Dienst neu.
- Prüfen Sie, ob die Firewall die Kommunikation blockiert

3.9.5.2 False positives in UX-Bridge Statistics Report Aufgabe

3.9.5.2.1 Symptom

Änderungen werden im Content-Repository vorgenommen, die UX-Bridge Statistics Report Aufgabe liefert dennoch Warnungen für das Objekt zurück

3.9.5.2.2 Ursache

Das Versenden der Nachrichten über den UX-Bus erfolgt asynchron, genau wie das Schreiben in das Content-Repository. Dadurch kann es länger als 5 Sekunden dauern, bevor der Adapter die Daten geschrieben und eine Antwort an FirstSpirit gesendet hat.

3.9.5.2.3 Lösung

Editieren Sie die Aufgabe und erhöhen sie den Wert im Scriptparameter „messageTimeout“. Weiter Informationen zu diesem Thema finden sie im Kapitel 3.5.5 Monitoring im Auftrag.

3.9.6 Konfigurieren des UXBService führt zu Exception

3.9.6.1 Symptom

Wenn in der Admin Konsole der UXBService ausgewählt wurde und anschließend auf „Konfigurieren“ geklickt wird, wird folgende Exception geworfen:

```
ERROR Tue Jul 24 14:37:30 CEST 2012
(de.espirit.firstspirit.client.AbstractGuiHost)
ExceptionHandler.uncaughtException() -
java.lang.SecurityException: Unable to create temporary file

java.lang.SecurityException: Unable to create temporary file
java.lang.SecurityException: Unable to create temporary file
    at java.io.File.checkAndCreate(null:-1)
    at java.io.File.createTempFile0(null:-1)
    at java.io.File.access$100(null:-1)
    at java.io.File$1.createTempFile(null:-1)
    at sun.misc.IOUtils.createTempFile(null:-1)
    at
javax.imageio.stream.FileCacheImageInputStream.<init>(null:-1)
    at
com.sun.imageio.spi.InputStreamImageInputStreamSpi.createInputStre
amInstance(null:-1)
    at javax.imageio.ImageIO.createImageInputStream(null:-1)
    at javax.imageio.ImageIO.read(null:-1)
    at
de.javasoft.plaf.synthetica.painter.ImagePainter.<init>(ImagePaint
er.java:234)
    at
de.javasoft.plaf.synthetica.painter.ScrollBarPainter.paintScrollBa
rThumbBackground(ScrollBarPainter.java:175)
    at
de.javasoft.plaf.synthetica.painter.SyntheticaPainter.paintScrollB
arThumbBackground(SyntheticaPainter.java:569)
    at
javax.swing.plaf.synth.ParsedSynthStyle$DelegatingPainter.paintScr
ollBarThumbBackground(null:-1)
…

Trace message truncated for length over 10K
```

3.9.6.2 Ursache

Das UX-Bridge Modul besitzt nicht die nötigen Rechte um Änderungen an der Konfigurationsdatei vorzunehmen.

3.9.6.3 Lösung

Den Wurzelknoten des Moduls „UX-Bridge“ auswählen und auf „Konfigurieren“ klicken. Im erscheinenden Dialog den Haken für „Alle Rechte“ setzen.

Anschließend alle Fenster mit „OK“ bestätigen und den FirstSpirit Server neu starten. Nun besitzt das Modul die erforderlichen Rechte. Nach dieser Änderung muss die Serverkonfiguration geschlossen und erneut geöffnet werden. Erst dann können Sie Änderungen an der Konfiguration vornehmen.

4 Glossar

Adapter Projektspezifische Schnittstelle zwischen UX-Bus und Content-Repository

Broker Nachrichtenkomponente des UX-Bus, hier Apache ActiveMQ

UX-Bus Zentrale Infrastrukturkomponente zur Verteilung von Inhalten

UXBService (auch UX-Bridge Service) Schnittstelle von FirstSpirit zum UX-Bus

5 Rechtliche Hinweise

Das Modul „UX-Bridge Installationshandbuch“ ist ein Produkt der e-Spirit AG, Dortmund, Germany.

Für die Verwendung des Modules gilt gegenüber dem Anwender nur die mit der e-Spirit AG vereinbarte Lizenz.

Details zu möglicherweise fremden, nicht von der e-Spirit AG hergestellten, eingesetzten Software-Produkten, deren eigenen Lizenzen und gegebenenfalls Aktualisierungs-Informationen, finden Sie in der Datei „third-party-dependencies.txt, die mit dem Modul ausgeliefert wird.