FirstSpirit™
Unlock Your Content

UX-Bridge Installation Manual

Version	**1.6**
Status	**RELEASED**
Date	**2014-12-12**
Department	Product Management
Author(s)	C. Feddersen
Copyright	2014 e-Spirit AG
File name:	UX-Bridge_Installation_EN

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

Table of contents

1 Introduction

The "UX-Bridge" module is a response to the trend of dynamic websites. Whenever content cannot be pre-generated, the CMS content has to be accessed dynamically. In this case, "dynamically" means that the content can change for every website user and at any point in time. UX-Bridge provides an infrastructure to meet the need for a dynamic content delivery platform. Consequently, the module expands on the hybrid architecture approach by adding a standard component for dynamic content delivery.

Additional information can be found in the white paper, Section 1.3.

1.1 Components

The UX-Bridge consists of a series of components that provide an infrastructure for creating web applications.

- UX-Bus
- FirstSpirit module
- UX-Bridge API
- Adapters
- Content repository

The infrastructure can be set up in a wide variety of ways. The UX-Bus can be installed on the FirstSpirit Server, a standalone server, or on the same server as the web application.

Two common architecture variants are described in the following.

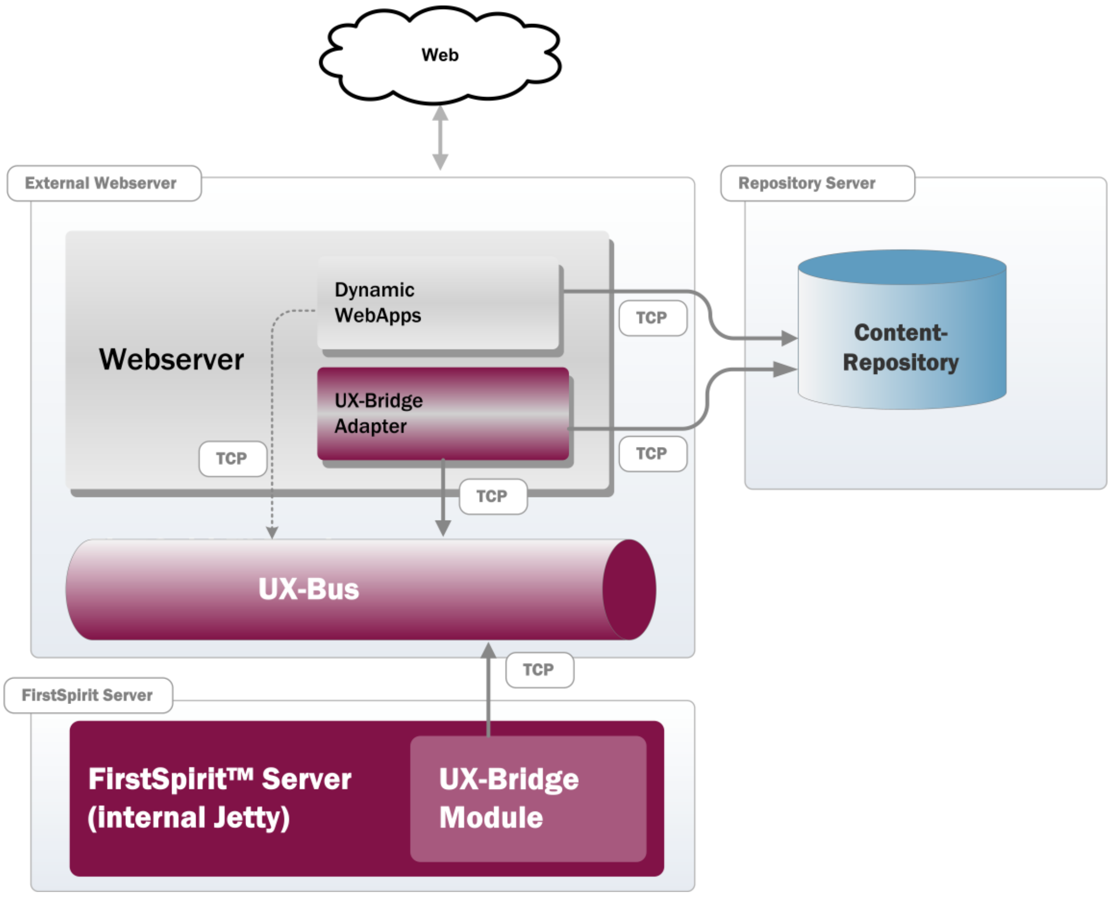


Figure 1: Architecture variant 1

In the first variant, the UX-Bus and the UX-Bridge adapters are on the same machine as the web container in which the web application is operated. The UX-Bus is operated as a standalone application. The UX-Bridge adapters jointly use the web container for the web application. The content repository is on a separate machine. The FirstSpirit server is also on its own machine, and therefore is decoupled from the live systems.

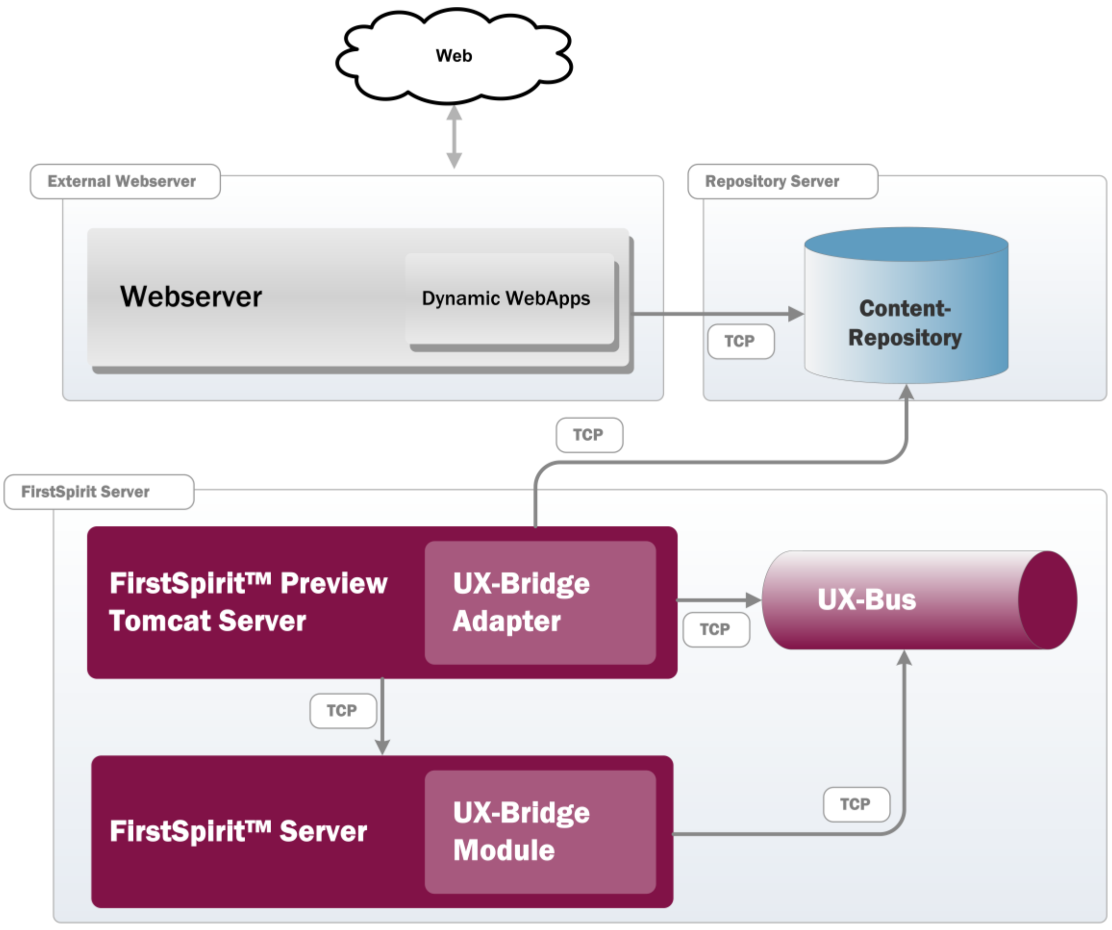


Figure 2: Architecture variant 2

In the second variant, the UX-Bus and any UX-Bridge adapters, along with the FirstSpirit server, are on one machine. Here, the UX-Bridge adapters are deployed as a web application in an Apache Tomcat, which is also used for the FirstSpirit preview.

Other variants are possible; the optimum solution always depends on the requirements in the project. Deciding factors, for example, include:

- What requirements apply concerning the availability of the individual components?
- What security guidelines have to be taken into account? Does the UX-Bus have to be accessible from third-party applications as well?
- What technologies are used for the UX-Bridge adapters, web applications, and content repository?

- What does the network connection of the components look like?

For additional information on its security and high availability features, refer to the corresponding sections in this documentation.

1.1.1 UX-Bus

The UX-Bus is a message-oriented middleware (MOM) for exchanging messages among the participating components. Apache ActiveMQ is used as the message broker. The routing is configured using the integrated Apache Camel framework.

The UX-Bus forwards the sent messages to the configured end points. Thus the messages generated by FirstSpirit are directed to the corresponding adapter. After saving the data, this adapter generates a response, which is sent from the UX-Bus to a component in FirstSpirit Server.

For information about installing the UX-Bus, refer to Section 2.1 Installation of the UX-Bus, page 9.

1.1.2 FirstSpirit module

These components have to be installed on the FirstSpirit server. The module's main component is a service that generates messages, sends them to the UX-Bus, and receives responses from the bus. Section 2.2 Installing the FirstSpirit module describes how to install and configure the module.

1.1.3 UX-Bridge API

The UX-Bridge API makes it possible to use UXBService in custom modules. To do this, the uxbridge-module-api-<version>.jar and JDOM version 1.0 must be included in the class path. The service can be accessed as follows:

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

1.1.4 Adapters

Adapters form the interface to the various content repositories and ensure that the data is written to a repository. Since adapters depend on the repository used as well as on the data model, they are project-specific components.

1.1.5 Content repository

The content repository or live repository is a data storage component which is filled by FirstSpirit and read out by web applications.

Here, an important paradigm of the UX-Bridge architecture is: "The type and number of repositories is not specified", since, depending on the kind of task, different repositories may be more or less suitable (see the Whitepaper, Section 2.1.3.2).

Thus the suitable content repository is to be selected and installed depending on the project requirements.

2 Installation

The necessary components for operation can be installed in different ways, depending on what is required. The following sections cover the individual components in more detail.

The following files are part of the delivery:

- 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 of the UX-Bus

2.1.1 Standalone operation

Standalone operation is the recommended operating mode for the UX-Bus. In this scenario, all essential components are decoupled from each other. Additionally, it is best suited for configuring high-availability scenarios.

2.1.1.1 Installation

The UX-Bus requires having at least Java 6 installed. For the exact system requirements, please refer to the provided technical data sheet.

Make sure that the environment variable is set to JAVA_HOME. Also, JAVA_HOME/bin should be added to the path variables.

The UX-Bus is installed simply by unzipping the distribution (UX-Bus_<VERSION NUMBER>.zip) delivered in conjunction with the module into an appropriate folder of your choice. This distribution is already configured to achieve good results in conjunction with the UX-Bridge. It is a pre-configured Apache ActiveMQ with integrated Apache Camel, in which the routes for the UX-Bus and the adapters were already configured.

The start scripts are in the "bin" folder. The "activemq" script can be used to start the bus.

Proceeding from the program folder, where ActiveMQ is located:

OS X/Linux:

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

Windows:

```
bin/activemq
```

The default configuration is used initially in this case. The configuration is in the "conf" folder and later will, of course, need to be adapted in line with your own requirements at a later point.

2.1.1.2 Testing the installation

As soon as the ActiveMQ bus has been started, a message to this effect should be logged on the console:

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

Using the netstat tool, you can now check whether the bus can in fact be accessed and whether the port was configured correctly.

In the default configuration, the port is 61616; it should be adjusted in accordance with your own configuration.

To test it, simply execute the following line in the console:

Under Windows:

```
netstat -an|find "61616"
```

Under Linux:

```
netstat -an|grep 61616
```

2.1.1.3 Installation as service

Configurations as a Windows Service or Unix Service is recommended for productive implementation. This ensures that the standalone server is also started automatically after restarting the operating system.

For information about installing as a service, visit http://activemq.apache.org/run-broker.html

When installing under Windows Server 2008 64-bit, however, follow the instructions on the site: http://blog.bigrocksoftware.com/2010/10/07/commons-daemon-procrun-as-a-java-service-wrapper-for-activemq/

2.1.2 Installing the UX-Bus on the FirstSpirit server

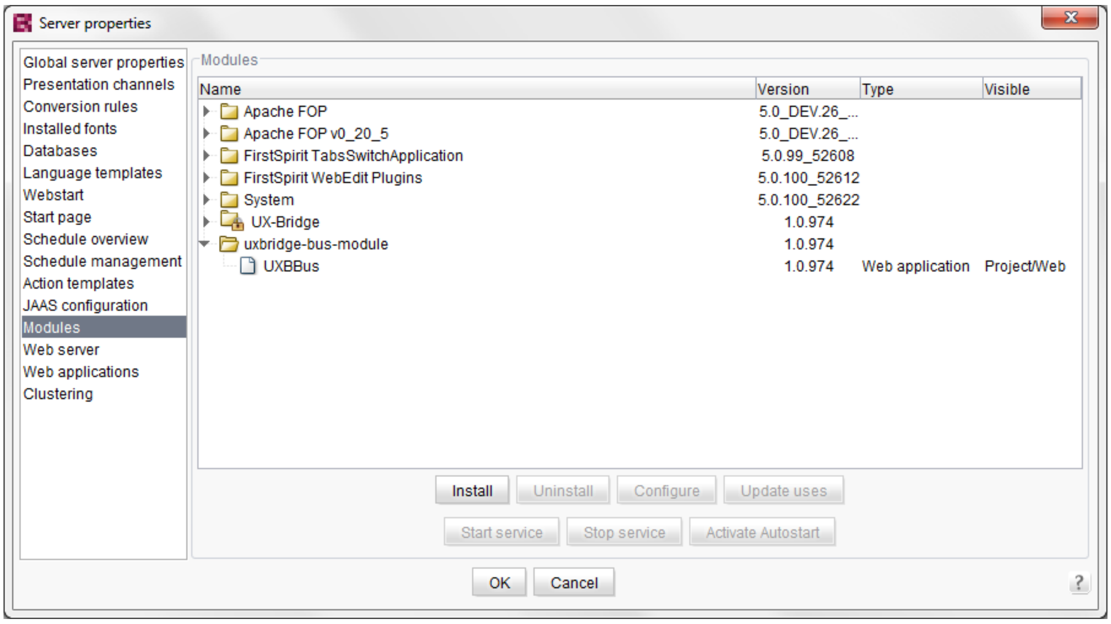
Operation of the UX-Bus as a module on a FirstSpirit server is recommended only if operational reasons make it impossible for you to install the UX-Bus as a standalone component, but you have access and administration rights for the FirstSpirit server. In this operating mode, warnings usually appear in the FirstSpirit server log which are a result of the UXBService being started before the UX-Bus. More information about these warnings is available at the end of this subsection.

However, if points such as Back channel (WebApp -> FirstSpirit), Clustering, and Failover will be included, the UX-Bus should be operated as a standalone server.

Under FirstSpirit 4.2R4, note that the UX-Bus cannot run on InternalJetty, and therefore should be installed on either an external server or an internal server other than InternalJetty (for example, Tomcat).

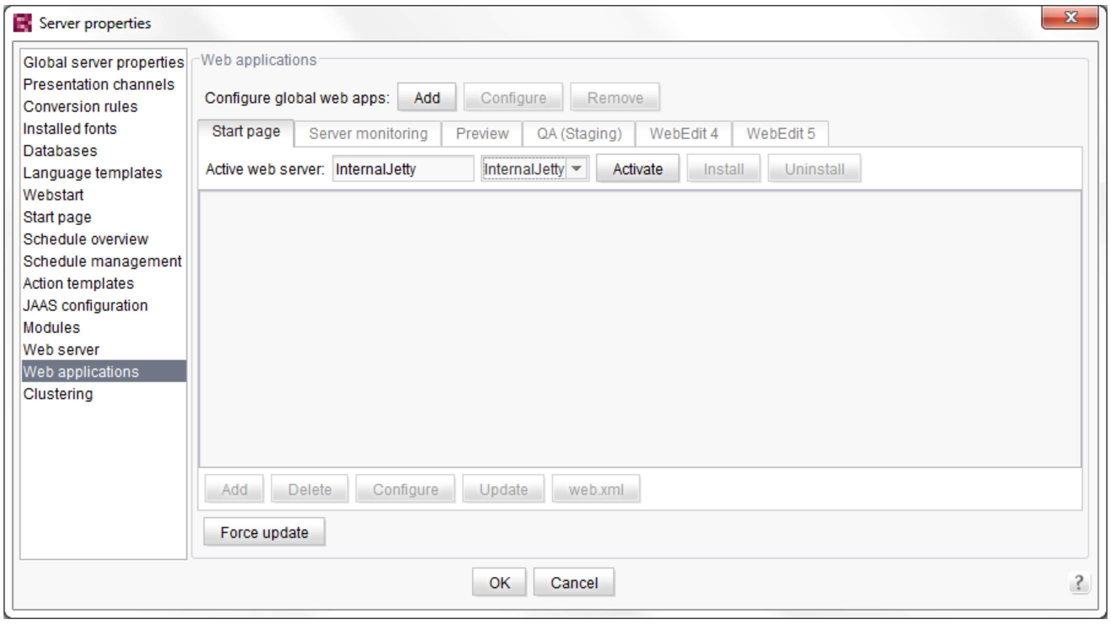
If you would like to operate the UX-Bus in this mode, please proceed as follows:

1. Install the "UX-Bridge bus" module; to do so, the provided uxbridge-bus-module-<version>.fsm has to be installed in the server properties. To ensure that all modules are functional, restart the FirstSpirit server after installing/updating a module.



Name	Version	Type	Visible
Apache FOP	5.0_DEV.26_...		
Apache FOP v0_20_5	5.0_DEV.26_...		
FirstSpirit TabsSwitchApplication	5.0.99_52608		
FirstSpirit WebEdit Plugins	5.0.100_52612		
System	5.0.100_52622		
UX-Bridge	1.0.974		
uxbridge-bus-module	1.0.974		
UXBBus	1.0.974	Web application	Project/Web

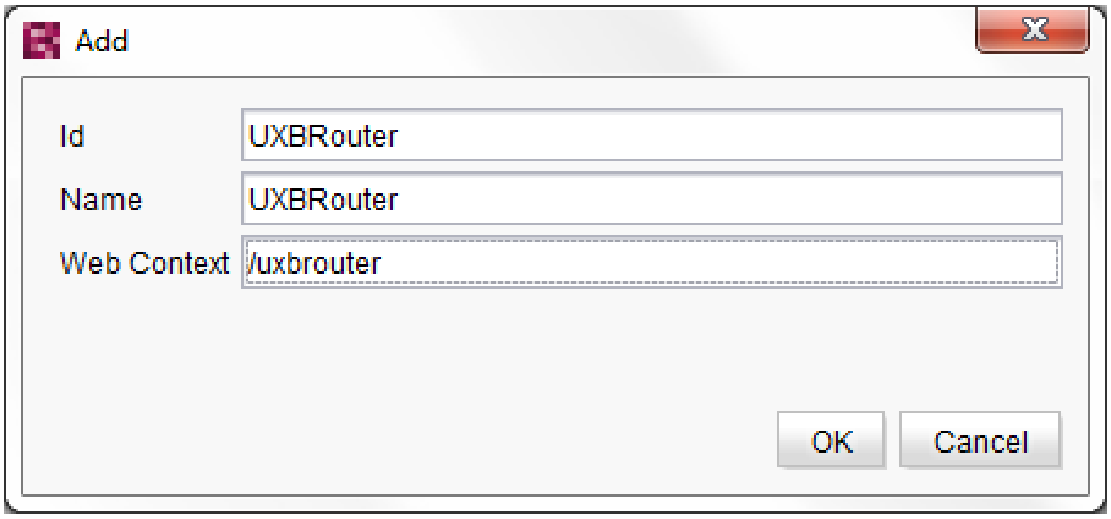
2. A global web application has to be created under Web applications



Configure global web apps:

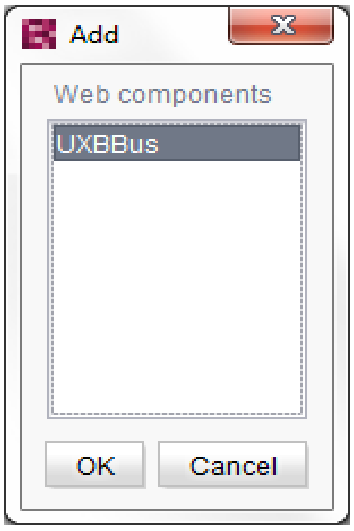
Active web server: InternalJetty   InternalJetty

The values shown here only serve as an example and can be chosen at will.

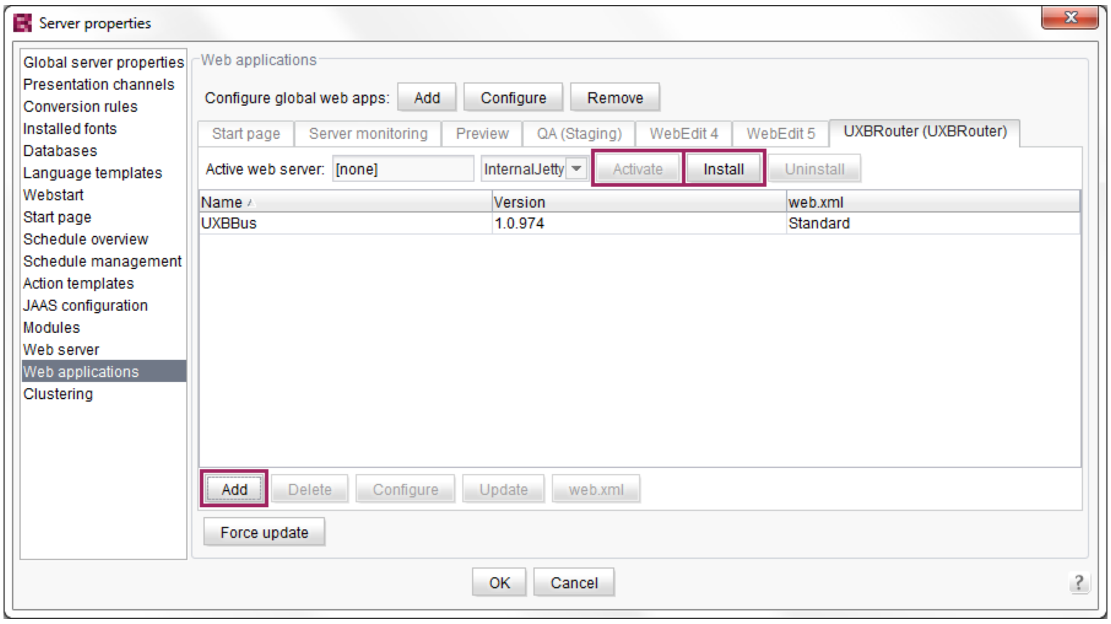


Id	UXBRouter
Name	UXBRouter
Web Context	/uxbrouter

3. Clicking Add installs the UX-Bus on the web application. You can specify the routes and configure the settings for the broker by clicking the Configuration button.



4. "InternalJetty" should then be selected and installed as the web server. After completing the installation, click the "Activate" button to start the web server.



Active web server: [none] InternalJetty

Name	Version	web.xml
UXBBus	1.0.974	Standard

The UX-Bus is then ready for use.

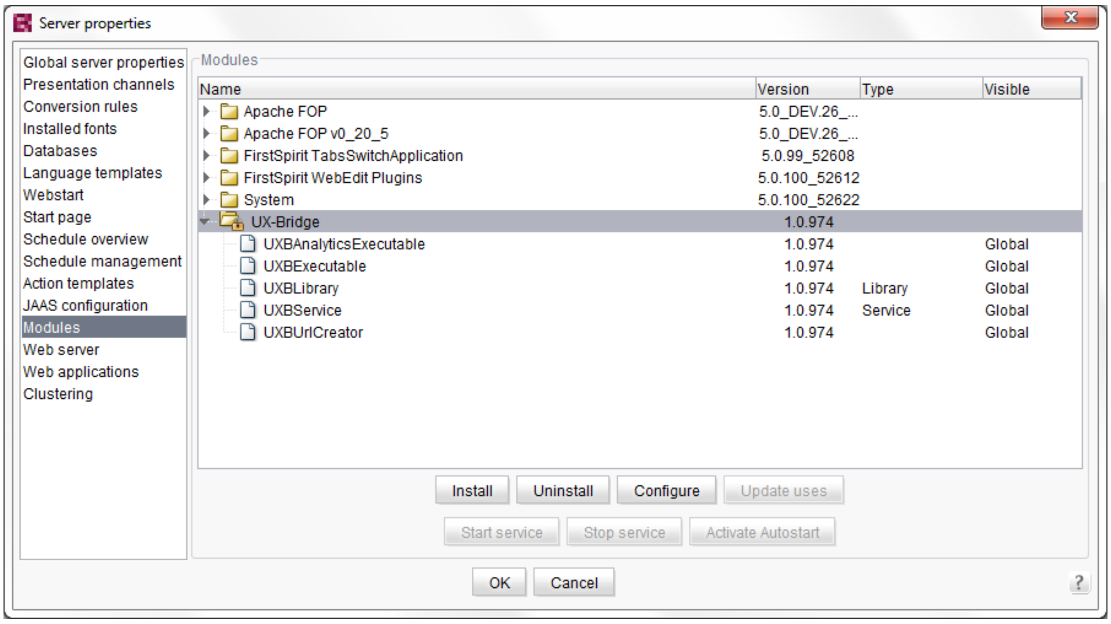
For more information on the topic of web applications, refer to the "Handbook for Administrators", Section 7.3.16, "Web applications".

In this operating mode, UXBService is started before the UX-Bus. Since UXBService attempts to connect to the UX-Bus directly after starting, the following error messages are issued repeatedly in the FirstSpirit server log. After all components are started, this message should no longer appear. If the message continues to appear, refer to the instructions in Section 3.9 Error analysis.

```
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 Installing the FirstSpirit module

To install the UX-Bridge module on the FirstSpirit server, select the "Modules" area under the server properties and click the "Install" button. Now select the file uxbridge-module-<version>.fsm to be installed. After installation, the new "UX-Bridge" folder should have been added. Select the entry, click "Configure", check "All rights", and confirm your changes.



Name	Version	Type	Visible
Apache FOP	5.0_DEV.26_...		
Apache FOP v0_20_5	5.0_DEV.26_...		
FirstSpirit TabsSwitchApplication	5.0.99_52608		
FirstSpirit WebEdit Plugins	5.0.100_52612		
System	5.0.100_52622		
UX-Bridge	1.0.974		
UXBAnalyticsExecutable	1.0.974		Global
UXBExecutable	1.0.974		Global
UXBLibrary	1.0.974	Library	Global
UXBService	1.0.974	Service	Global
UXBUrlCreator	1.0.974		Global

Now close the server properties by clicking "OK".

To ensure that all modules are functional, restart the FirstSpirit server after installing/updating a module.

For more information about installing modules, refer to Section 7.3.14, "Modules", in the documentation for administrators.

2.2.1 FirstSpirit 4.2R4

If UX-Bridge is used in a FirstSpirit version lower than 5, current versions (1.6.4) of the two jar files, **slf4j-api.jar** and **slf4j-simple.jar**, will need to be copied into the <fs>/shared/lib/ folder. Current versions of these files can be downloaded from http://www.slf4j.org/.

In addition, a current version (11.0.2 or higher) of Guava Library, such as **guava-11.0.2.jar**, must be copied to the <fs>/shared/lib/ folder. The current version of this file can be downloaded from https://code.google.com/p/guava-libraries/.

This must be done before installing the uxbridge-module-<version>.fsm. The FirstSpirit server will also need to be restarted before the module can be installed.

2.2.2 Configuring the UX-Bridge service

The UX-Bridge service is the interface from FirstSpirit to the UX-Bus; it is used to send messages.

The service is very easy to configure:

1. Open Server Properties -> Modules
2. Click UX-Bridge and select UXBService
3. Click Configure

2.2.2.1 XML tab

The configuration is carried out using a Spring XML DSL. For more information about syntax, visit http://camel.apache.org/spring.html.

The following is an example of the configuration for the sample application:

```
<?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**:
  The Apache ActiveMQ link is configured in this Bean. With the brokerURL property you can configure not only the link, but also the Failover behavior, among other things.

  Information about all configuration parameters can be found using the following 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**:
  Two routes are configured here:
  FS_OUT = The route by which FirstSpirit sends the data
  FS_IN = The route in which FirstSpirit waits for new messages

- **UxbServiceStatisticsResponseHandler**:
  This Bean is the handler of the messages sent back by the adapters.

- **uxbExceptionProcessor:**
  This Bean is used to process exceptions that occur during transport.

For development and test purposes, the UX-Bridge service can be configured without a running bus (for information on buses, refer also to Section 3.1 Routing). If the UX-Bridge service is unable to connect with a bus, it regularly logs a related report. If this occurs over an extended period of time, it may be difficult to read the log entries. Therefore, a dummy configuration can be created that is used to save all messages as files in a specific folder.

```
<?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>
```

After the service has been reconfigured, it will have to be restarted in order to implement the changes.

In this example, the messages are saved in the folder *<FS-Home>/log/uxb-messages/*. However, these files are not deleted automatically, which can cause the folder to become quite large, depending on the number of messages sent. If a high volume of messages is expected, you should set up a way to regularly delete old messages.

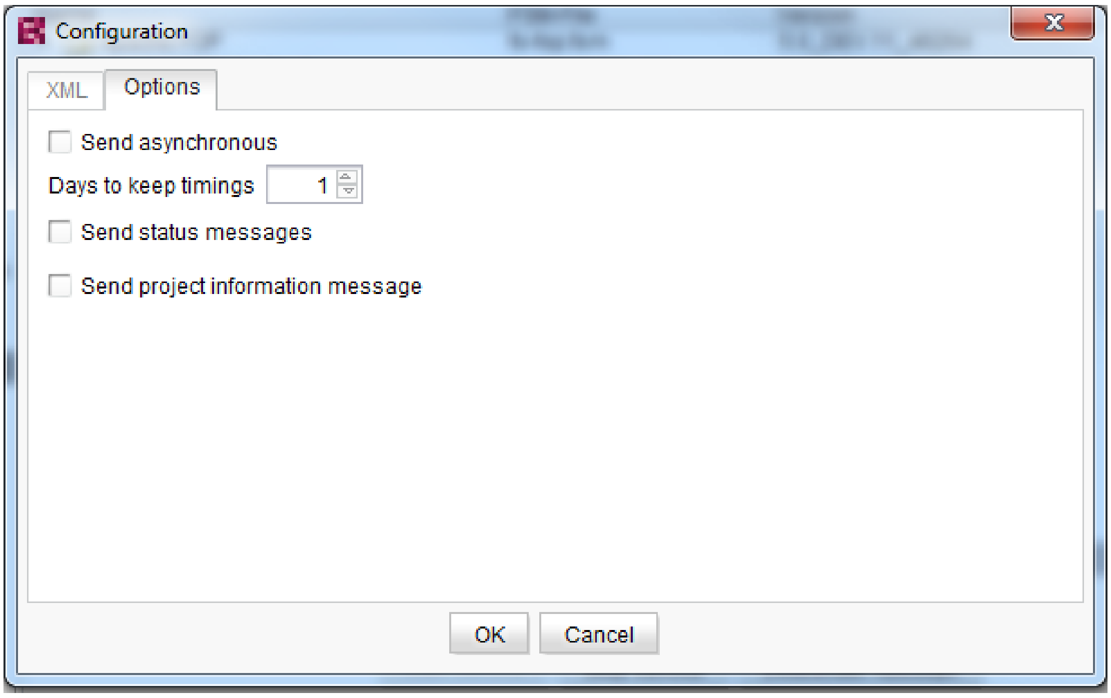
2.2.2.2 Options tab

In the second tab of UXBService, you can define whether the messages are to be sent asynchronously or synchronously. If the box is checked, the messages are sent asynchronously. A higher performance can be achieved by doing so, since, unlike with synchronous communication, there is no waiting for a response. This performance improvement can be very advantageous with large amounts of data.

Information on sent messages are saved internally in a cache file for timing purposes until a response is received from an adapter. The length of time that messages without a response are kept until they are removed from the cache can be defined using the "Days to keep timings" input window.

The "Send status messages" checkbox can be selected to configure whether the UX-Bridge sends another message to the bus at the start of and after generation so that it is possible to respond directly to these events in the adapter (see the Developer Handbook for more information).

The "Send project information message" checkbox controls whether a message is sent with project information at the start of generation (see the Developer Handbook for more information).



Send asynchronous: [ ]
Days to keep timings: 1
Send status messages: [ ]
Send project information message: [ ]

2.3 Installing adapters

Adapters are components that are developed within the project. To simplify administration and maintenance, it is advisable to develop the adapter as a web application that is installed in a web/servlet container or application server. In many cases, the necessary infrastructure already exists. However, this is not a mandatory prerequisite, so the adapter can be developed as a standalone Java application that is installed on a server.

For more information on how to develop adapters, refer to Section 3.3, "Adapters", in the Developer Documentation.

3 Configuration

The UX-Bus includes some default routes when first installed, but they can be adjusted and expanded as needed. Depending on the kind of installation (standalone or in FirstSpirit), changes to the configuration will need to be made at various places. These aspects are explained in the initial sections.

The following sections cover information on high availability as well as security, monitoring, and logging.

3.1 Routing

Routing is used to pre-filter and distribute messages. Routing is configured in a Spring XML file expanded using Camel DSL (http://camel.apache.org/spring.html).

By default, FirstSpirit requires two routes: FS_OUT for sending and FS_IN for receiving messages. Different routes can be defined under Routing, depending on the need and use case. The messages sent via the FS_* routes can be filtered and/or distributed.

The default routes:

```
<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>
```

The routes, just like the adapters, are very project-specific. For a more detailed introduction, refer to Section 3.5, "Routing" in the Developer Documentation .

3.1.1 Standalone UX-Bus

The routing is defined in the conf/camel.xml file in the installation directory of the UX-Bus. In the distribution provided, some default routes are already specified, so that no adjustments are required initially. For details on configuring the routing, refer to Section 3.6 Changing the routing on page 36.

3.1.2 FirstSpirit Server

If the UX-Bus was installed on the FirstSpirit server, please proceed with configuring the routes as follows.

1. Start the Server and Project Configuration
2. Open the server properties
3. In the left-hand menu, select the "Web application" item
4. Select the created global web application (see Section 2.1.2 Installing the UX-Bus on the FirstSpirit server, page 11)
5. Click the "Configure" button
6. In the dialog you can now add new routes or edit existing routes.

For a description of the routes to be configured in FirstSpirit, refer to Section, 3.5.1, "End points in FirstSpirit", in the Developer Documentation.

3.2 Logging

This section covers the configuration of the Apache ActiveMQ logging mechanisms. In the event of errors, the log files and log outputs on the default output are the first places to look for the causes.

Apache ActiveMQ uses the Apache Commons Logging API for internal logging.

It can therefore be used easily with a wide variety of logging frameworks implemented in the Java environment. Apache log4j is used for logging in the default configuration. For detailed information about configuring log4j, refer to the log4j online documentation: http://logging.apache.org/log4j/1.2/

The Apache ActiveMQ documentation also includes some information on the topic of logging: http://activemq.apache.org/how-can-i-enable-detailed-logging.html

3.2.1 Logging in the UX-Bus

The default output file for logging is data/activemq.log. If problems arise, this is the place to start troubleshooting. Logging can be configured in the conf/log4j.properties file in order to adapt the log level to the particular requirements, for instance. The "INFO" log level is used in the default configuration. The following are output on the console and in the log file:

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

Furthermore, you can define packages which are to use a different log level. In the default configuration these are:

```
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
```

For debugging purposes, the log level can then be adjusted, for example, by replacing the respective line. Thus for the debug level:

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

In this case, you should be aware that many more messages will appear in debug mode, which makes it more difficult to filter out the relevant messages. To prevent this, you could leave the rootLogger configuration set to INFO and just set the configuration for the package to be investigated to DEBUG or even TRACE. Thus:

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

In the delivered distribution it is sufficient to activate the following two lines in the <activemq>/conf/log4j.properties file in order to receive appropriate debug messages for the routing and processing of the messages:

```
# 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

LoggingBrokerPlugin provides even more detailed logging than log4j and, above all, enables you to obtain more information about what occurs in the UX-Bus. The plug-in can simply be added to <plugins> in the conf/activemq.xml configuration file:

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

More detailed explanations and other attributes are included in the Apache ActiveMQ documentation:
http://activemq.apache.org/logging-interceptor.html

3.2.3 Logging in clients

Logging of the UX-Bus helps in the event of errors and problems on the broker. However, if errors arise outside of the UX-Bus, logging should also be possible to quickly remedy the problems. Problems can arise at all clients linked to the bus. This includes UXBService and the implemented adapters.

The logging tool used by UXBService is log4j, which is made available by FirstSpirit API. The logging configuration used by FirstSpirit server is also used. To retain DEBUG messages for the UX-Bridge service, you only need to activate DEBUG logging for the FirstSpirit server. To do so select the item Configuration -> Logging under Server Monitoring on the FirstSpirit server. Edit the active configuration and change the first line as follows.

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

The log level is adjusted automatically after the configuration is saved.

Adapters are used in the project, making the developer responsible for providing corresponding logging and documenting the usage accordingly.

3.3 High availability

If the UX-Bridge is used only for filling content repositories, the criteria pertaining to availability are often already fulfilled by the default configuration. If the UX-Bus fails, there is no direct impact on the website (live system), since the integrated web applications communicate only with the content repository. Operation of the website is therefore ensured even during a temporary failure. During the failure, however, no data from FirstSpirit can be written to the content repository.

Brief failures, such as due to restarting a component, a power failure, etc., are absorbed in the default configuration by mechanisms such as automatic reconnects, redelivery of messages, and persistent message storage. In these cases, therefore, no loss of data is to be expected.

If components fail irreparably, the original state can easily be recovered by reinstalling the components and fully deploying the data.

If the UX-Bridge is used as an integration component with which other systems also communicate by exchanging messages, the demands for availability are usually higher. In these cases, use of master/slave configurations can make sense for ensuring high availability of the UX-Bus. If there is an extremely high volume of messages or the messages are very large, another option is to configure load balancing. Of course, both mechanisms can also be used in combination.

If data that FirstSpirit cannot generate is also saved in the content repositories, special attention should be paid to backing up this data. Of course, in a high availability scenario, the content repository should also be designed with equivalent redundancy to avoid becoming a single point of failure. This is particularly critical since the function of the entire website may be impaired.

The following sections cover the individual components and issues to consider when configuring a high availability scenario. The term "broker" is used repeatedly and refers to the message component of the UX-Bus. Apache ActiveMQ is used in particular.

3.3.1 Message store

KahaDB is used with activated persistence in the delivered distribution as the message store of the UX-Bus. If you also want to implement a disaster recovery plan for the message store, corresponding instructions are at http://activemq.apache.org/replicated-message-store.html.

For detailed instructions on configuring the message store, visit http://activemq.apache.org/persistence.html.

3.3.1.1 Backing up the message store

At delivery, KahaDB is configured for message persistence in the configuration (\conf\activemq.xml):

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

To create a backup of the KahaDB, the following steps are necessary:

1. Freeze the file system containing the database to ensure that you get a consistent snapshot of the journal

2. Back up the database using standard backup mechanisms

For more information, visit:

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

If a JDBC persistence of the JMS messages is used as an alternative to the Kaha persistence (http://activemq.apache.org/persistence.html), use the familiar database-specific backup mechanisms.

3.3.2 Failover

If the UX-Bus fails, it should be ensured that the performance of UXBService and the adapters is not impaired. This means that either attempts are made to resend the messages later or the sending occurs directly via a second instance.

All clients logged in at the respective UX-Bus should be able to reconnect to another UX-Bus or attempt to establish a new connection after a certain waiting period. The failover protocol is provided for this purpose.

The failover protocol is already configured in the default configuration of UXBService:

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

In this case, the failover protocol ensures that if a connection is lost, attempts to establish a new connection are made at certain intervals. Here you can determine the various parameters yourself, such as how long the first waiting time should be, by what factor the waiting time should be increased with each attempt, and how long the maximum waiting time should be (initialReconnectDelay, backOffMultiplier, maxReconnectDelay).

The following is an example of a failover configuration with multiple UX-Bus instances:

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

In this case, broker1 is used to send the messages. If this fails, the messages are automatically sent by broker2.

If you use your own adapters, make sure that these also use the failover protocol to protect against dropped connections or failure of the UX-Bus.

The failover protocol is most useful when multiple UX-Bus instances are configured in a cluster. (also refer to Section 3.3.3 Master/slave operation, page 29) Now multiple UX-Bus instances can be specified, with which the failover protocol can connect. If the connection with a UX-Bus is lost, a connection to another UX-Bus is automatically established. For time-critical applications, the backup and backupPoolSize parameters can also be used to establish a certain number of backup connections that can be used immediately if an active connection is lost. The advantage is that the connection already exists in a standby state and does not have to be established in an emergency.

An alternative to using failover with clustering is to specify only one UX-Bus and, when configuring the UX-Bus, to arrange for the logged-in clients to automatically receive a list of every available UX-Bus with which they can establish a connection if there is a problem. The "updateClusterClients" and "rebalanceClusterClients" are provided for this purpose in the "TransportConnector" configuration of the UX-Bus.

For more information about the failover parameters, refer to the Apache ActiveMQ documentation at: http://activemq.apache.org/failover-transport-reference.html

3.3.3 Master/slave operation

Master/slave operation is a good way to increase the fail-safe performance. Here, an Apache ActiveMQ instance acts as master and one or more instances act as slave. The clients always send their requests to the master; only if the master fails does one of the slave instances become the new master and assumes its tasks. The messages will now be processed by this instance.

In contrast to this is the use of a network of brokers (Section 3.3.4 Network of brokers page 30), for which all Apache ActiveMQ instances have equal rights.

In both cases, it is important to configure the failover in the client as described in Section 3.3.2 Failover on page 28.

3.3.3.1 Shared storage master/slave

In "Shared storage master/slave" clustering mode, the same KahaDB is used by multiple brokers for message persistence. Only one master can access the persistence, while the slaves are blocked. The slaves check at regular intervals whether they can get access to the persistence. If the master broker fails and shuts down, the slave broker receives access to the persistence. Only at this moment does the slave broker boot up completely and enable clients to establish a connection.

Two types of this shared persistence are possible:

One option is to use a relational database, in which the brokers store the data. One advantage of this method is that databases are backed up quickly and easily.

The second option is to access a shared file system on which the Apache ActiveMQ brokers then deposit the KahaDB. The only requirement of this file system is that there be a locking mechanism that ensures that there is always only one broker with access to the KahaDB.

To configure "shared storage master/slave" clustering, you only need to configure two brokers with access to the same persistence. To safeguard against failure, the brokers should run on different machines. The clients receive both broker addresses in the failover protocol, and if a master broker fails, a connection is immediately established with the new master broker. This broker accesses the data of the same persistence and thus is up to date. In each of the brokers, the persistence adapter has to be adjusted so that they all use the same directory.

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

For more information about configuration, refer to the official documentation at http://activemq.apache.org/shared-file-system-master-slave.html.

Be aware that this mode is prone to errors in combination with the file system. If the machine on which the file system is located fails, all brokers that are supposed to access it lose access to the persistence and, in the worst case, have to be shut down. If your own application is to be protected from errors in connection with the file system, choose a file system that is also distributed across multiple machines and remains functional if a machine fails.

3.3.4 Network of brokers

A "network of brokers" consists of at least two different brokers, each of which has its own persistence store. In this case, it is not absolutely necessary for each broker to have the same messages (or all messages). One use case for this architecture is for load distribution.

In most usage scenarios, it is not necessary to use a network of brokers to distribute the load. Generally a single UX-Bus can process the anticipated amount of messages without any problems.

If load balancing is still desired, a client configuration would look something like this:

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

master1 and master2 are a "network of brokers"; the messages are randomly distributed.

This architecture is best clarified by way of an example:

Broker A receives a message in Queue A and saves it in its own store. As long as no client at Broker B is listening for messages from Queue A, these messages are not needed at Broker B, which is why it does not receive them. As soon as a client of Broker B listens for messages from Queue A, Apache ActiveMQ ensures that Broker B also receives these messages in order to deliver them to the client.

Information about which clients are listening to which queues at which brokers is exchanged among the various brokers by means of what are called "advisory messages". This way every broker always knows which messages are or are not of interest to other brokers.

When configuring individual brokers, you can also define filters for various messages or exclude other brokers from messages.

From the perspective of the client, this concerns the high availability of the brokers, but not the messages that are sent. The reason for this is that each broker has its own database for messages. If a broker fails, the messages of this broker are not distributed again until the broker has been restarted.

For more information on this topic, visit:

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

3.3.5 Use with FirstSpirit GenerationServer

By using one or more generation servers, the generation load can be decoupled from the FirstSpirit master server. In addition to generating the static pages, the generation server can also take over message generation. Messages continue to be sent via the UX-Bus service, which runs on the FirstSpirit master server.

3.3.6 High availability of the content repositories

In addition to the correct configuration of the UX-Bridge infrastructure, the configuration of the content repositories is also critical to ensure high availability.

Selection of a suitable repository is made for the specific project. Please consult the documentation for the repositories being used to learn more about high availability.

3.3.7 High availability of the adapters

Adapters are implemented depending on the specific project. During development, the requirements regarding high availability have to be taken into account accordingly. In many scenarios, the requirements can be covered by multiple instances of the adapter and appropriate routing.

3.4 Security

In the default UX-Bus configuration, each application can be connected to the bus to send and receive messages. Depending on the area where the UX-Bridge and dependent applications are used, this may not be desired, especially if sensitive data is being transmitted.

It should not be possible to reach the UX-Bus from outside, rather it should send the messages via an internal network only in order to ensure the highest possible security. In this case it is important to ensure that connections to and from the outside are blocked. We recommend doing this by configuring the relevant firewall rules on the computers. Alternatively or additionally, this can be achieved by security settings in the UX-Bus. One option is to use the "messageAuthorizationPolicy" mentioned below. For more details, refer to the Apache ActiveMQ documentation (http://activemq.apache.org/security.html).

The UX-Bus can be protected from unauthorized or unauthenticated access. For simple use cases, using the "simple authentication plugin" from Apache ActiveMQ is sufficient; this permits the access data to be stored in a configuration file. The Apache ActiveMQ "JAAS plug-in" also enables the use of standardized, easy to configure Java login modules, which allow authentication via various sources, such as LDAP, property files, etc. In addition, you can write your own JAAS login modules, which use mechanisms such as Kerberos, NTLM, NIS, etc. for authentication or authorization. The ActiveMQ broker can also be operated with certificate-based encryption (such as SSL).

If even more fine-grained control is to be set up, ActiveMQ offers "operation-level" authentication and "message-level" authentication options. The "operation-level" authentication option can be used, for example, to specify which users can read/write to/from which destination. Unlike authentication and authorization at the broker level, "message-level" authentication makes it possible to permit only certain messages to reach the destination, for example, only messages for a certain recipient.

Using the Apache ActiveMQ plug-in API, it is also possible to write new security plug-ins that are adapted to your own requirements.

For more detailed documentation, refer to http://activemq.apache.org/security.html.
We also recommend the Fuse MQ Enterprise documentation: http://fusesource.com/docs/mqent/7.0/security/front.html
The following keywords may help with your search:
simpleAuthenticationPlugin, jaasAuthenticationPlugin, authorizationPlugin, messageAuthorizationPolicy, jaasCertificateAuthenticationPlugin

Another point related to security is access via JMX. This is protected by default with ActiveMQ and can be configured using the “jmx.access” and “jmx.password” files (http://activemq.apache.org/jmx.html).

3.5 Monitoring

This section covers various options for monitoring Apache ActiveMQ. Since high-end applications always depend on good performance and low susceptibility to error, it is important to be able to localize bottlenecks quickly and reliably, and to be able to quickly gain a reliable picture of the current state of the infrastructure should problems arise.

3.5.1 Apache ActiveMQ Webconsole

Apache ActiveMQ comes with options for managing and monitoring the application. For instance, there is the web console, which can be accessed at http://localhost:8161/admin as soon as it is activated. Where appropriate, "localhost" has to be replaced with the address of the machine where the UX-Bus is running.

For details on activating and configuring the web console, refer to the online documentation: http://activemq.apache.org/web-console.html

Above all, it is necessary that not everyone has unlimited access to the web console. In the conf/jetty.xml file, therefore, the line

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

should be replaced with the line

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

Authorized users and their access data can then be entered in the file conf/jetty-realm.properties (the default user is "admin" with the username/password: admin/admin).

The web console offers simple functions for viewing information via the broker and providing a view of simple statistics. The display includes information about queues, topics, subscribers, connections, and the network. Additionally, there is a function for sending messages. This enables easy testing of applications without first writing extra code. The web console is intuitively operated, self-explanatory, and therefore not explained in greater detail here.

3.5.2 Apache ActiveMQ command line tools

Some useful command line tools are also delivered with Apache ActiveMQ; these provide basic admin functions, including some functions for monitoring. Using the "activemq-admin" tool and the "query" parameter you can, for example, retrieve and read diverse broker, destination, connector and connection information. You can inspect messages using the "browse" parameter.

For a more detailed description of the scope of functions for all the options, refer to the official Apache ActiveMQ documentation: http://activemq.apache.org/activemq-command-line-tools-reference.html

3.5.3 Monitoring using JMX

Another option for monitoring is to use the JMX API. (Java Management Extensions). Some parameters of the above-mentioned command line tools (such as "query") also use the JMX API to get information about brokers, queues, etc. For details about configuring JMX, please refer to the Apache ActiveMQ documentation: http://activemq.apache.org/jmx.html

If the Apache ActiveMQ broker is configured to permit JMX connections, you can, for example, write your own help programs, which establish a JMX connection and read broker information.

To do so, refer to the official JMX API (http://docs.oracle.com/javase/7/docs/technotes/guides/jmx/spec.html) and the Apache ActiveMQ API, which provide additional functions (http://activemq.apache.org/maven/5.6.0/activemq-core/apidocs/). The "BrokerViewMBean" interface from the Apache ActiveMQ API is a good starting point for reading out the broker's data using JMX.

The MBeans from Apache ActiveMQ are in the "org.apache.activemq" package and can be read out using tools such as JConsole, JVisualVM, or the like. For information about Apache Camel and the routes, refer to MBean org.apache.camel.

Via JMX Monitoring you can get information such as

- How many messages have been sent
- How many messages have been routed or have expired
- How many connections (UXBService, adapter) there are to the UX-Bus

In the event of an error, this can be used to gain valuable information via monitoring.

3.5.4 Monitoring using advisory messages

Advisory messages are used by Apache ActiveMQ to enable brokers to communicate with each other and to exchange information about queues, consumers, etc. To use these messages for your own monitoring purposes, for example, you can write your own application for listening to advisory messages. For details on this, refer to the Apache ActiveMQ documentation: http://activemq.apache.org/advisory-message.html

3.5.5 Monitoring in the schedule

UXB service itself also provides an option for monitoring. Each message sent to the UX-Bus receives a time stamp when created. If an adapter processes the message, another time stamp is set as well as the status after the processing (ok/fail). In the schedule, a method can be invoked that queries the UXB service and returns a list of messages, their processing time, and their status. To do so, a script action has to be added to the end of the schedule:

```
#! executable-class
com.espirit.moddev.uxbridge.inline.UxbResultHandler
```

Since the messages are processed asynchronously and can take different amounts of time depending on the adapter and number of messages, it is possible to use a script parameter (messageTimeout) to specify the maximum amount of time (in seconds) that the service should wait for the response message before the processing times are evaluated.

```
…
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)
```

The results of the messages that were not processed until after the specified waiting period are kept in the memory. The length of time set until these are automatically deleted can be configured in the service settings (see Section 2.2.2.2 Options tab).

3.6 Changing the routing

If the routing is to be changed because new adapters or content repositories have been added, for example, the configuration has to be adjusted as described in Section 3.1 Routing page 22.

After the changes, the UX-Bus has to be restarted, so that the new or changed routes are activated. When doing so, note that restarting the UX-Bus can result in messages being lost.

If a deployment is running during the restart and the UX-Bus restart lasts longer than the time that the failover transport requires to achieve the maximum number of reconnect attempts, the deployment of the corresponding content element is canceled and the attempts continue. This element will then not be present in the content repository later. To minimize this risk, it is advisable to test the failover transport in advance and configure it so that the UX-Bus can be restarted normally without loss of data.

The default configuration uses the failover default parameters. These are described in the online documentation:
http://activemq.apache.org/failover-transport-reference.html

The parameters maxReconnectAttempts and startupMaxReconnectAttempts were set to the following values:

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

The first waiting period is 10 ms and is doubled with each of 10 attempts. This yields a total waiting period of:

10+20+40+80+160+320+640+1,280+2,560+5,120=10,230 ms = 10.23 s

Thus if the server is not available for longer than 10 s, a data record will be lost.

The UXB service does not have to be restarted while the UX-Bus is restarting, since the failover transport automatically reestablishes the connection after the UX-Bus is restarted. It is advisable to use the failover transport in custom adapters so that the connection is reestablished if lost and the adapter does not have to be restarted.

3.7 Backup

Since the UX-Bridge consists of a variety of components, the backup has to be designed depending on the components.

3.7.1 Backing up the FirstSpirit project

For instructions on how to back up and restore FirstSpirit projects, refer to the FirstSpirit admin documentation.

3.7.2 Backing up the message broker

Usually there are very few messages on the message broker. A message in the queue leaves the queue as soon as it is retrieved. The message broker passes responsibility for subsequent handling of the message to the adapter. Messages that still have not been "picked up" by the adapter are on the broker. If you want to back these up, this can be accomplished by backing up the KahaDB if the messages and the KahaDB are permanently saved. For details on this, refer to Section 3.3.1.1, "Backup".

3.7.3 Backing up content repositories

The procedure for backing up the data in a content repository depends on the solution employed. Depending on the system used, follow the instructions from the corresponding documentation to create a backup.

3.8 Disaster recovery

Data can be damaged or lost after a system failure or critical error. In an emergency, this data should be restored as quickly and reliably as possible.

In the case of failures in which messages have been lost, it is easiest to carry out a full deployment from FirstSpirit; the current state of the project is then written by UX-Bridge to the content repository. You can then be certain of having consistent, current and error-free data in the repository that other applications can work with once again.

If for any reason it is not possible to renew the data using a full deployment, the data will have to be restored manually.

The process for doing so depends on the component that has failed.

3.8.1 Errors in FirstSpirit

If data recovery of the FirstSpirit server is required due to a failure, please contact e-Spirit AG help desk to receive relevant instructions.

3.8.2 Errors in the UX-Bus

If the error occurs in the bus infrastructure and the infrastructure has been damaged, backed-up data does not have to be manually restored in most cases. In the worst case, the messages that were not yet processed on the bus and were not persistently stored are lost. In the default configuration, however, persistence is enabled.

With the Apache ActiveMQ, the KahaDB is used to save and process messages. Therefore, the KahaDB will have to be restored if data is lost. This in turn is only possible if the broker is configured so that the KahaDB is persistently stored. Otherwise any data on the UX-Bus at the time of the crash will be lost. Normally the Apache ActiveMQ itself takes care of restoring persistently stored data when restarting the application. For custom configurations, for example in conjunction with a relational database or the KahaDB on a shared file system, the backup mechanisms common for the respective technology can be used. In addition, high availability is explained in Section 3.2 of this documentation.

3.8.3 Errors in the adapter

No substantial problems should arise if the adapter has crashed, since data loss is to be anticipated only in extreme situations. Normally the messages simply remain on the broker until the adapter is restarted and retrieves these messages.

Most likely, only the message that was being processed at the time of the crash will be lost, depending on the implementation of the adapter. In this case, you can carry out a redeployment of the data in FirstSpirit.

3.8.4 Errors in the external content repository

If data has to be restored in the external repository, whatever steps are necessary have to be carried out there. These vary depending on the application used. You can look up the details in the respective documentation. A typical use case could be restoring a relational database backup.

3.9 Error analysis

This section is intended to support the administrator or developer during troubleshooting.

The UX-Bridge consists of multiple components, any of which could be the source of a fault.

The following checklist can be processed if contents do not appear on the website as expected or are transferred to the content repository.

1. Check the FirstSpirit schedule that carries out the deployment
2. Check whether the messages are being routed correctly
3. Check whether the adapter is functioning correctly
4. Check the logs of the content repository

3.9.1 Check the FirstSpirit schedule

3.9.1.1 Errors in the UX-Bridge task in the schedule

3.9.1.1.1 Symptom

The task ended with an error message and the contents were not transferred to the content repository.

Example of the error message:

```
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 Cause

The UX-Bridge service has not been started.

3.9.1.1.3 Solution

Start the UX-Bridge service in the service properties and ensure that autostart is enabled for this service.

3.9.1.2 Errors in the UX-Bridge generation schedule

3.9.1.2.1 Symptom

The generation schedule for the UX-Bridge presentation channel ended with errors.

3.9.1.2.2 Cause

There are two conceivable causes for this:

1. There are errors in the templates, which have to be resolved by the template developer.

2. Error messages of the following type appear:

```
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 this case, the connection to the UX-Bus could not be established.

3.9.1.2.3 Solution

- Make sure the UX-Bus is available. Next, restart the UX-Bridge service. Ideally, the UX-Bus should be started before FirstSpirit Server.

- If the error messages occur despite having an active UX-Bus instance, please check the configuration of the UXB service. Is the hostname/IP of the UX-Bus correct? Correct the configuration if necessary and restart the UX-Bridge service (UXBService).

- Check whether the communication is being blocked by a firewall.

3.9.1.3 Warnings occur in the UX-Bridge statistics report task

3.9.1.3.1 Symptom

The UX-Bridge statistics report task in the schedule ends with warnings.

Example:

```
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 Cause

There are two potential causes conceivable for this:

1. The adapter was unable to process the message correctly; as a result, it did not send a message about the content repository update that occurred.

2. The adapter was able to process the message, but the message about the update that occurred was not sent within the defined time frame. Also refer to Section 3.9.5.2 False positives in UX-Bridge statistics report task, p. 44 in this regard.

3.9.1.3.3 Solution

Check whether the adapter routed and processed the message correctly. Refer to Section 3.9.2 Checking whether the messages are being routed properly, p. 42 in this regard.

3.9.2 Checking whether the messages are being routed properly

The UX-Bus does not log any information regarding routing messages in the default configuration.

In Section 3.2 Logging, starting on page 23, you can find information on how to configure logging accordingly.

3.9.3 Checking whether the adapter is functioning correctly

It is possible that the adapter cannot process the messages correctly. The developer of the adapter should write corresponding log outputs during its implementation so that these types of problems can be detected during ongoing operation.

3.9.4 Checking the content repository

Depending on the content repository being used, its log files and administration tools that may be available can aid in troubleshooting. We cannot provide additional information here, since these components are selected for a specific project.

3.9.5 Other errors

3.9.5.1 FirstSpirit Server reports that a JMS connection to the UX-Bus could not be established

3.9.5.1.1 Symptom

Error messages stating that no JMS connection to the UX-Bus could be established can be found in fs-server.log.

Example:

```
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 Cause

The UX-Bus could not be reached when starting FirstSpirit Server or the connection was lost during operation.

3.9.5.1.3 Solution

- Make sure the UX-Bus is available. Next, restart the UX-Bridge service (UXBService). Ideally, the UX-Bus should be started before FirstSpirit Server.
- If the error messages occur despite having an active UX-Bus instance, please check the configuration of the UXB service. Is the hostname/IP of the UX-Bus correct? Correct the configuration if necessary and restart the UX-Bridge service.
- Check whether the firewall is blocking communication.

3.9.5.2 False positives in UX-Bridge statistics report task

3.9.5.2.1 Symptom

Changes are made in a content repository, but the UX-Bridge statistics report task is still returning warnings for the object

3.9.5.2.2 Cause

Messages are sent asynchronously via the UX-Bus, just like writing to a content repository. This can make it take longer than 5 seconds before the adapter writes the data and sends a response to FirstSpirit.

3.9.5.2.3 Solution

Edit the task and enter a higher value for the script parameter “messageTimeout”. More information about this topic can be found in chapter 3.5.5 Monitoring in the schedule.

3.9.6 Configuring the UXBService results in an exception

3.9.6.1 Symptom

The following exception is thrown if UXBService has been selected in the admin console and then "Configure" is clicked:

```
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 Cause

The UX-Bridge module does not have the necessary permissions to make changes to the configuration file.

3.9.6.3 Solution

Select the root node of the "UX-Bridge" module and click "Configure". Check the "All permissions" box in the dialog that appears.

Then confirm all of the windows with "OK" and restart FirstSpirit Server. Now the module has the required permissions. The server configuration has to be closed and reopened after making this change. Only then can you make changes to the configuration.

4 Glossary

Adapter Project-specific interface between UX-Bus and content repository

Broker Message component for the UX-Bus (Apache ActiveMQ in this case)

UX-Bus Central infrastructure component for distributing content

UXBService (also UX-Bridge service) Interface from FirstSpirit to the UX-Bus

5 Legal notices

The "UX-Bridge Installation Manual" module is a product of e-Spirit AG, Dortmund, Germany.

Only a license agreed upon with e-Spirit AG is valid with respect to the user for using the module.

Details regarding any third-party software products in use but not created by e-Spirit AG, as well as the third-party licenses and, if applicable, update information can be found in the file "third-party-dependencies.txt" included with the module.