FirstSpirit™
Unlock Your Content

UX-Bridge Developer Documentation

Version	**1.7**
Status	**RELEASED**
Date	**2014-07-09**
Department	Product Management
Author(s)	C. Feddersen
Copyright	2014 e-Spirit AG

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

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 for the requirement 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.

This documentation is intended to support development with the UX-Bridge infrastructure and to provide a look at the concept.

The first section describes the general steps to consider when using UX-Bridge. The chapter covers the topics of installation, thoughts on data exchange format, setup in FirstSpirit, routing, and adapter and web application integration.

Chapter 3 explains the use of UX-Bridge through two tutorials. Here, we will cover implementation in FirstSpirit, the creation of an adapter, and the web application step-by-step, with concrete examples.

1 Concept

Certain projects have requirements that are to be implemented using UX-Bridge. In these cases, before implementation, a few questions have to be considered and addressed when defining the concept.

When developing a concept, two distinct situations can arise. If a FirstSpirit project is developed from scratch, then the focus of development is on the optimum implementation of the specialized requirements. At the same time, the concept should be flexible enough to make it easy to implement and integrate future requirements. If, on the other hand, UX-Bridge is to be integrated into a pre-existing project, the main question is how best to integrate UX-Bridge into the existing concepts and mechanisms.

In the following, some points will be considered which become relevant in many projects.

1.1 Generation and deployment concept

In many projects, the website is updated through periodic generation and deployment schedules. These schedules carry out a complete or partial alignment process. Under certain circumstances, these schedules can also be run manually. If, in this scenario, UX-Bridge is to be used, it is often sufficient to expand the existing schedules by adding the UX-Bridge-specific tasks. Details on this can be found in Section 2.1.4, "Create and configure schedule".

During generation, a message is sent to the UX-Bus for every page reference generated within the UX-Bridge generation task. Within the project, it should thus be ensured that, within this task, only the necessary page references are generated. If in a project, for example, only the news (maintained via a content source) is to be delivered via UX-Bridge, then only the necessary content projection pages are to be generated in the UX-Bridge generation task. In order to carry out this limitation, you can, for example, use a partial generation. Alternatively, full generation can also be used. However, "uxbSkipMessage" should then be used in order to cancel the generation of page references in the UX-Bridge presentation channel that are not to generate any messages (see Online Documentation for FirstSpirit: Vorlagenentwicklung\Vorlagensyntax\Systemobjekte\#global\vorschaubezogen\Abbruch einer Vorschau/Generierung).

In other projects, a majority of changes are released via workflows, which subsequently carry out generation and deployment of the participating objects. In most cases a script identifies the changed objects, which are then defined as start nodes of a partial generation. Objects are deleted using delete workflows. UX-Bridge can be easily integrated into these scenarios as well (see Section 2.1.7, "Workflow coupling").

If the time in which the objects must be available on the website is of high importance, then the use of a workflow-oriented approach is often the better choice. The fewer objects that have to be created during a generation process, the faster the objects will be available on the website. DeltaGeneration in FirstSpirit 5 provides a simplified mechanism for such scenarios (Developer API\Delta Generation, Access-API\Generate Task). FirstSpirit 4 already offers this capability through the revision API.

All told, UX-Bridge can be incorporated into the existing generation and deployment concept or into one to be newly created, and for this purpose, does not include a standalone (separate) solution.

1.2 Data model and adapter

If UX-Bridge is to be integrated into an existing FirstSpirit project, then the data model is often preset in FirstSpirit. With heavily structured content through the database schema; with weakly structured content through forms of the page and section templates. Here, we recommend reviewing whether the web applications to be created (which later are to access the UX-Bridge data) can work with this data model, or whether a different data model makes more sense. This could be the case if the web application requires a much simpler or a much more complicated data model.

In the latter case, the FirstSpirit data is, in other words, just a part of the data model of the web application. In the first case, a subset of the data model stored in FirstSpirit is sufficient for many web applications. You also must consider whether the data model of the web application is to be denormalized for performance reasons.

If you have decided to use deviating data models, then you should clarify the step in which the transformation from one data model to another is to be carried out. The answer is certainly project-specific, and therefore only the possible variants will be described here. An evaluation has to be carried out in the context of the project:

a) The transformation is handled in the CMS syntax in the templates, which generate the XML necessary for UX-Bridge (see Section 2.1.2, "Data exchange format – FS templating vs. WebApp development"). The adapters themselves do not carry out any large transformations, but instead write the objects to the content repository in the manner defined by the data exchange format.

b) The data exchange format corresponds to the data model in FirstSpirit 1:1. The transformation in the data model for the content repository is carried out within the adapter.

c) It is a hybrid approach, which carries out transformations in both components. This depends on where implementation is easier.

The following considerations can help during the evaluation:

1) If the data is to be written to multiple content repositories, then it is often sensible to generally retain the data exchange format and carry out any necessary/logical transformations for writing to the content repository within the adapter.

2) If the data is to be written to multiple content repositories, then for every content repository, a unique adapter can be implemented which contains only the logic necessary for this repository. Alternatively, an adapter can write the data to both content repositories. This is useful, for example, if the data is supposed to be written within a transaction bracket.

3) Does an adapter handle only one particular type of object, or are multiple object types bundled into one adapter? Object types in this context refer to different types of content. Say, for example, you would like to make all products and all news from a FirstSpirit data source available via UX-Bridge. Here, for example, we need to determine whether the two types of objects are to be transferred to the same content repository and/or data model or not.

4) For decoupling and maintenance purposes, it is generally important to consider whether it would be better to use multiple (but lean) adapters or one adapter that contains all the logic.

5) The advantage of a general data exchange format is that only one message has to be sent via the UX-Bus, which, however, then has to be processed by multiple adapters and can be written to multiple content repositories. In addition, no adaptations to the data exchange format may be necessary if new content repositories and web applications are to be connected in the course of the project.

6) If UX-Bridge is to communicate with a system that can already receive JMS messages, then it may make sense to carry out a transformation directly on the UX-Bus. In this way, no adapter has to be implemented, which in turn would generate JMS messages only. In such cases, the routing can be expanded on the UX-Bus by adding corresponding transformation instructions.

1.3 News distribution/routing

As mentioned in the previous section, for every page reference within the generation, a message is generated and sent to the UX-Bus. Part of the UX-Bus is a routing component which receives the message and forwards or routes it to another so-called "end point". Details on the standard configuration can be found in Section 2.4, "Routing".

This configuration can be adapted for the project-specific requirements. In this case, a simple domain-specific language (DSL) is available with the Apache Camel Spring XML Syntax with which all current enterprise integration patterns (see http://camel.apache.org/enterprise-integration-patterns) can be implemented. With this powerful integration framework, the UX-Bus can be used as an information hub for the website and all participating systems.

Here are some examples which can be implemented on the UX-Bus through a routing:

1) A content router is configured which sends certain messages only to certain end points/adapters.

2) New end points can be configured which serve as the interface to web applications or back-end systems. Through these, an adapter can, for example, direct a web application to empty its cache because new data has been written to the content repository.

3) Existing third-party applications can send messages to the UX-Bus, which are then processed by the web application, adapters or FirstSpirit.

In the standard configuration of UX-Bridge, a routing method is already configured which is sufficient for standard scenarios. Project-specific adaptations are necessary only for more complex scenarios (see "Data model and adapter").

2 Quick Walkthrough

This chapter describes the steps for implementing UX-Bridge in a project. The Quick Walkthrough is intended for experienced FirstSpirit template developers and web application developers. Detailed, step-by-step instructions can be found in Chapter 3, "Tutorials".

2.1 FirstSpirit

2.1.1 Installation

The first step in development within the context of UX-Bridge is to install the components.

To do this, first install the provided module in the server settings of FirstSpirit Server. Doing this will start the UX-Bridge service, and the UX-Bridge components will be available in the projects of the server (see "Installation of the FirstSpirit Module" in the UX-Bridge installation manual).

In addition to the FirstSpirit module, installation of the UX-Bus is also required. For local development, the installation in standalone operation is recommended (refer "Standalone Operation" in the UX-Bridge installation manual).

2.1.2 Data exchange format – FS templating vs. WebApp development

The architecture of UX-Bridge permits the development of solutions based on UX-Bridge to be divided into two roles. A template developer creates the necessary templates, workflows and schedules. A (web) application developer develops the adapter for the content repository and the (web) application. If the roles are performed by different people, then a common data exchange format should be defined during the conceptual design process. In this case the intended data format is the one that is generated by the templates and is sent as a message to the adapter via the UX-Bus.

The data exchange format thus forms the interface between the components and thus also between the two roles. From the point of view of the template developer, this is the end product of their work. For the (web) application developer, this represents the input for the adapter. Here, only an outer container is prescribed by UX-Bridge. The remainder can be freely defined (see the next chapter).

A carefully selected data exchange format can significantly reduce the implementation effort. The following questions can help during the selection:

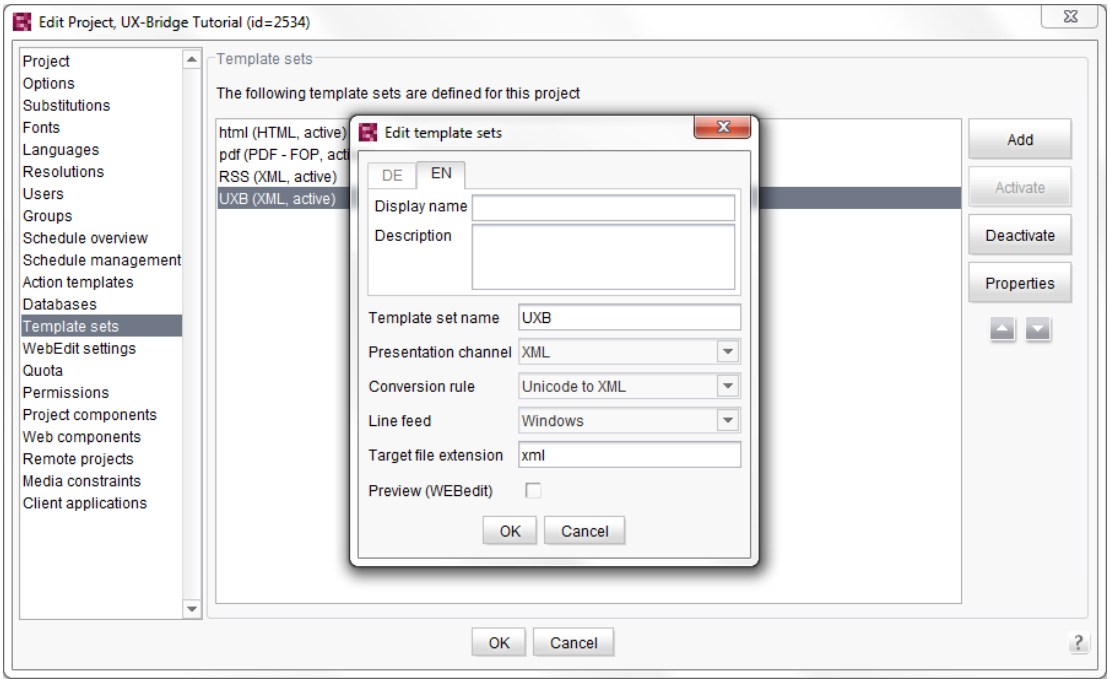
- Does the data, and thus the data model, already exist in FirstSpirit? If yes, then you are subject to certain limitations. If no, it is advisable to match the data exchange format as closely as possible to that of the data in the content repository or of the web application.

- For performance reasons, it may make sense to write the data to the content repository in denormalized form. Possible inconsistencies can be corrected through a full deployment, because the data usually continues to be available in normalized form in FirstSpirit.

- Is the data to be written to more than one content repository? If yes, it is to be determined whether a data exchange format is sufficient and/or the adapter(s) can take over the transformation and persistence in the content repository. In some cases, it may even be more efficient to generate two data exchange formats through FirstSpirit, which then can be adopted in the respective content repository without a transformation step.

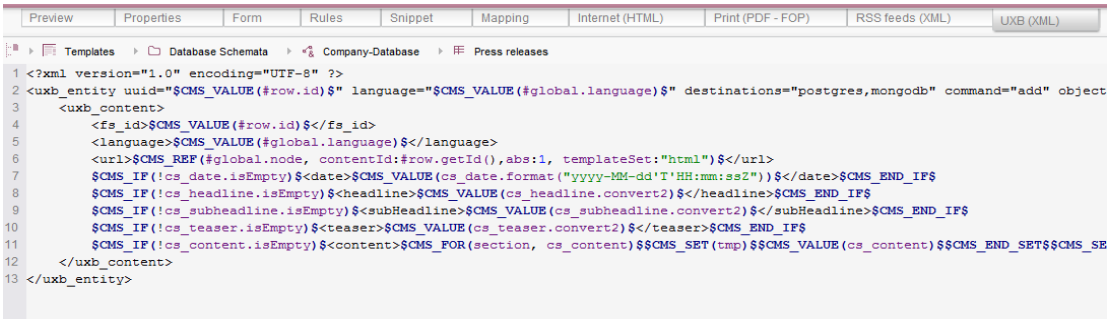
2.1.3 Creating and filling a presentation channel

To use UX-Bridge, a new template set (UXB) must first be created in the project settings under "Template sets". As a presentation channel, "XML" is to be configured as the "Unicode to XML" conversion rule and "xml" is to be configured as the target file extension.



The "Unicode to XML" conversion rule (see Conversion rules for Unicode to XML) serves to transform special characters and control characters into corresponding XML entities, which otherwise would be interpreted in XML as a part of the markup language or would appear as invalid characters.

In order to send messages to the UX-Bus, the corresponding template that is to generate the messages contains fields that have been defined in the data exchange format and which are to be output in XML format.



Only the following structure is predefined:

```
<uxb_entity
          uuid = String
          destinations = String
          language = String
          command = String
          objectType = String
>
          <uxb_content>
                    […] definiertes Datenaustauschformat […]
          </uxb_content>
</uxb_entity>
```

Property	Description	Example	Required field
Uuid	Unique identifier of the object, for example, fs_id	1234	Yes
destinations	Destination(s) of the message (live repository, comma-separated),	postgres,mongodb	Yes
language	Language of the message	DE (German)	No
command	Command to be executed by the adapter (e.g. create/ delete)	Add	No
objectType	Object type evaluated by the adapter (e.g. News, Products)	News	No

The language, command and objectType attributes are optional, but have proven helpful with the adapters implemented by e-Spirit.

2.1.4 Create and configure schedule

In order to convert the data from FirstSpirit into messages that can be further processed by UX-Bus, it is mandatory for a schedule to be created or an existing schedule extended so that it generates XML. This is then forwarded to the UXB service, which generates a message from it and sends this to the UX-Bus.

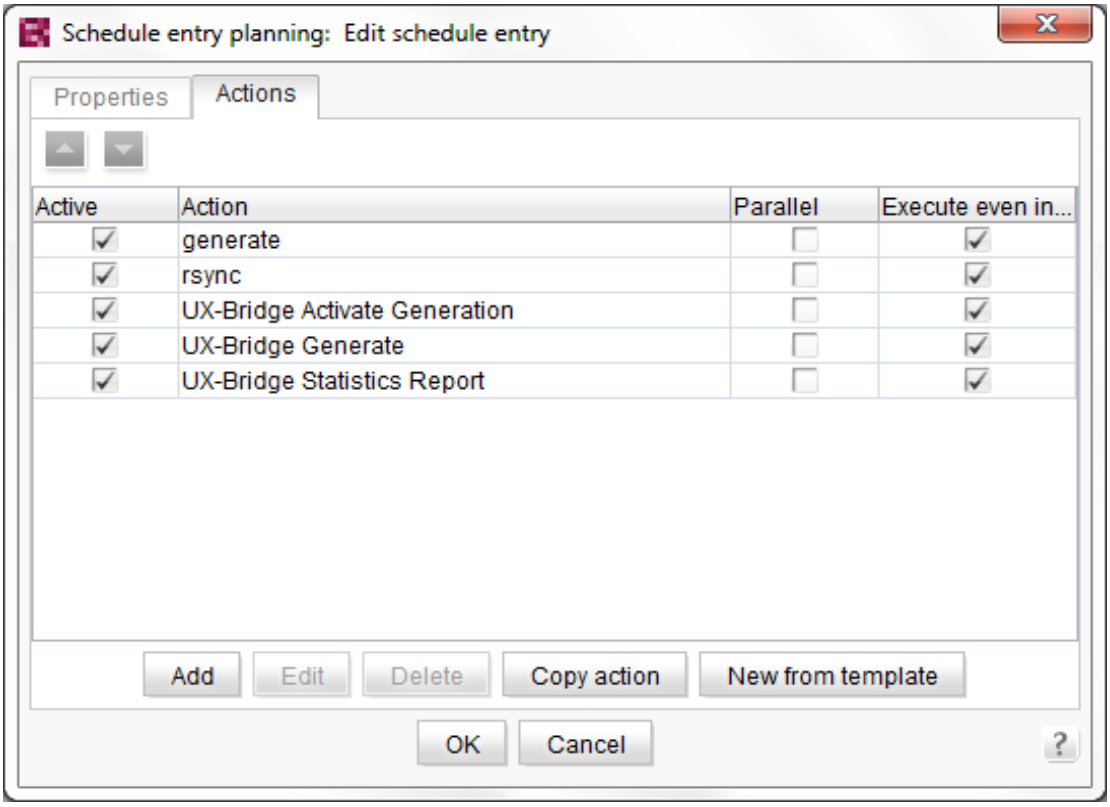
The schedule can be started later via a workflow (see Release workflow).

This section will now describe two schedules that will probably be needed in every project that uses UX-Bridge.

2.1.4.1 Partial generation

In order to publish new content quickly on the website, it is useful to create a schedule or to expand on one in such a manner that it carries out the steps described in the following while only generating and publishing the new content.

A typical generation schedule which uses UX-Bridge is divided into multiple actions as described in the following:



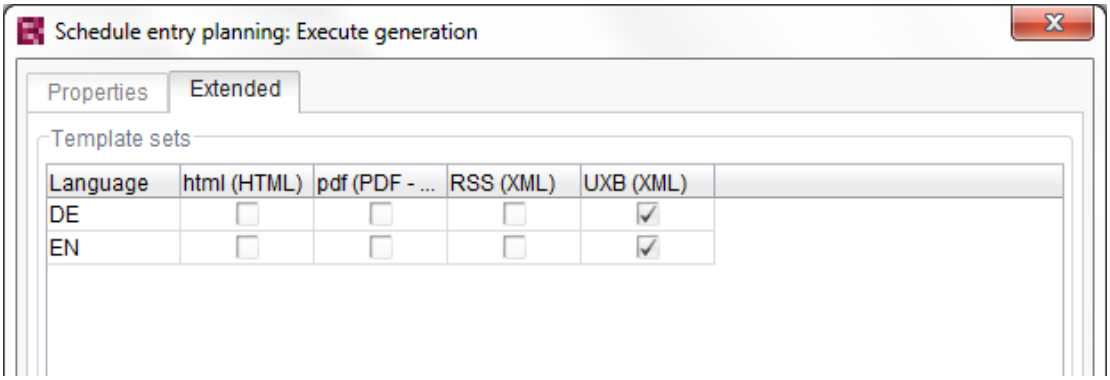
First, all of the complete static pages that are needed for display on the website (e.g. news overview pages) should be generated in a generation action. This generation action takes place in the deployment schedules commonly used to date and does not have to be adapted.

In the next step, the content generated in the previous step should then be transferred to the web server as usual (in the example, via rsync). In this step, too, no adaptations are usually necessary.

Next, in order to use UX-Bridge, a new, additional action needs to be added which activates the generation process for UX-Bridge by calling a script:

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

In the subsequent generation action, the exact page should then be generated (e.g., the news detail page) that generates the XML to be forwarded to the UXB service. It is important to ensure that the UXB template set has been enabled in the advanced properties.



Delta generation in FirstSpirit 5 now adapts this generation action automatically so that it no longer generates all pages, but rather just the page desired.

In addition, for example, a workflow script that initiates the generation process can transfer the generated page to the schedule.

The last action, "UX-Bridge Statistics Report", is optional and enables the cycle times to be measured for the messages in the bus until deployment on the website.

```
INFO  22.08.2012 09:59:54.631
(de.espirit.firstspirit.server.scheduler.ScheduleManagerImpl):
starting task 'UX-Bridge Statistics Report' - schedule entry 'UX-
Bridge-Test (News)' (id=5142)

INFO  22.08.2012 10:00:04.645
(com.espirit.moddev.uxbridge.service.UxbServiceImpl): Time for
#uxb/pressreleasesdetails/UXB/EN/256 (postgres): 242ms

INFO  22.08.2012 10:00:04.645
(com.espirit.moddev.uxbridge.service.UxbServiceImpl): Time for
#uxb/pressreleasesdetails/UXB/DE/256 (postgres): 224ms

INFO  22.08.2012 10:00:04.645
(com.espirit.moddev.uxbridge.service.UxbServiceImpl): 2/2 deployed
successfully (overall: 233ms, postgres: 233ms).

INFO  22.08.2012 10:00:04.646
(de.espirit.firstspirit.server.scheduler.ScheduleManagerImpl):
finished task 'UX-Bridge Statistics Report' - schedule entry
```

```
'UX-Bridge-Test (News)' (id=5142)
```

The following script call is necessary for this:

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

The service waits the maximum amount of time defined in the UX-Bridge service configuration until the adapters' responses are evaluated. If there is no response within this time frame, then the message is classified as having a delivery error. Since response times can vary depending on message and system, this value can be configured.

2.1.4.2 Complete alignment process

It makes sense to create an additional schedule which carries out a complete alignment process in order to maintain the most current state of the data inventory in the content repository. For this purpose, the data deleted in FirstSpirit must also be deleted in the external repository.

The following procedure is recommended:

1) Complete generation of the static pages in FirstSpirit.

2) Run the UX-Bridge schedule in order to write all data to the content repository (also refer to the previous section). This data is to be given an up-to-date time stamp, which is saved in content repository in the "lastmodified" field.

3) Run a script that calls up the cleanup method with a timestamp as a parameter describing the latest project revision of the schedule. The cleanup method then deletes all data saved in the lastmodified field that are older than the ones copied over. Those are the data that were already deleted in the FirstSpirit project and thus, in step 2, have no new timestamp saved in "lastmodified".

```
import com.espirit.moddev.uxbridge.api.v1.service.UxbService;
uxbService = context.getConnection().getService(UxbService.class);
uxbService.removeUxbEntriesByTime(context.getStartTime().getTime()
, "news", "postgres,mongodb");
```

2.1.4.3 Historic data

To generate historic data using UX-Bridge a new script action is to be added before the action "UX-Bridge - Activate Generation".

```
import java.util.Date;

Date d = new Date(114, 5, 24);
context.setStartTime(d);
```

The date that is set will be used in the following action ("UX-Bridge - Activate Generation").

2.1.4.4 Recognizing the schedule start/end in the adapter

To make sure the adapter is able to respond at the start and/or end of a schedule in certain circumstances, the UX-Bridge can send a separate message automatically at both the start and end of generation (see Installation Manual).

The format of the start message is as follows:

```
<uxb_entity projectName="PROJEKT_NAME" status="start"
schedulerId="AUFTRAGS_ID" createTime="ZEITPUNKT_DER_NACHRICHT"
projectId="PROJEKT_ID" startTime="STARTZEITPUNKT_DES_AUFTRAGS" />
```

During full generation, the command="startMaintenanceMode" attribute is also added.

The format of the end message is as follows:

```
<uxb_entity projectName="PROJEKT_NAME" status="end"
schedulerId="AUFTRAGS_ID" createTime="ZEITPUNKT_DER_NACHRICHT"
projectId="PROJEKT_ID" startTime="STARTZEITPUNKT_DES_AUFTRAGS" />
```

During full generation, the command="stopMaintenanceMode" attribute is also added.

2.1.4.5 Adding root attributes

The root node of every message sent through the UX-Bridge comes with a set of predefined root attributes like the project name or the id of the schedule entry (see also chapter 2.1.4.4 Recognizing the schedule start/end in the adapter). They can be extended by further custom attributes by adding the following script to the beginning of the schedule:

```
import java.util.HashMap;

attributeMap = new HashMap();
attributeMap.put("customAttribute1", "customAttributeValue1");
attributeMap.put("customAttribute2", "customAttributeValue2");

context.setProperty("uxbMessageRootAttributes", attributeMap);
```

The script creates a HashMap containing each custom attribute as a key/value-pair. The HashMap will then be added to the schedule context, so that the UXBService can include the additional attributes during generation of the messages.

2.1.5 Skipping pages during generation

Pages can be skipped when generating messages by setting the "uxbSkipMessage" page variable.

```
$CMS_SET(#global.pageContext["uxbSkipMessage"], true)$
```

The use of "stopGenerate" (see Online Documentation for FirstSpirit: Vorlagenentwicklung\Vorlagensyntax\Systemobjekte\#global\vorschaubezogen\Abbruch einer Vorschau/Generierung) is not supported and in this case will result in invalid XML, resulting in error messages in the log.

2.1.6 Reading out expanded project information

The UX-Bridge can send a message with expanded project information at the start of generation (see Installation Manual). This information currently includes the defined project languages and resolutions.

Example:

```
<uxb_entity projectName="UXB" objectType="projectInfo"
schedulerId="92460" createTime="1371037891875" projectId="88785"
startTime="1375346932923">
  <project key="UXB">
    <id>88785
    </id>
    <name>UXB
    </name>
    <languages>
      <language key="DE">
        <name>Deutsch</name>
        <abbreviation>DE</abbreviation>
        <isMasterLanguage>true</isMasterLanguage>
      </language>
    </languages>
    <resolutions>
      <resolution key="ORIGINAL">
        <name>Originalauflösung</name>
        <uid>ORIGINAL</uid>
        <height>0</height>
        <width>0</width>
        <isOriginal>true</isOriginal>
      </resolution>
    </resolutions>
  </project>
</uxb_entity>
```

2.1.7 Workflow coupling

The publication of content via UX-Bridge can be started directly via scripts and schedules or indirectly via workflows.

2.1.7.1 Release workflow

In order to publish content, an existing workflow simply has to be expanded by adding a workflow script that starts a schedule which, alongside generation and deployment, also generates XML messages and forwards them to the UXB service (see Partial generation).

2.1.7.2 Delete workflow

In order to delete content, an existing delete workflow has to be expanded by adding a workflow script that generates an XML message, which is forwarded to the UXB service.

Invoking the UXB service is written in the script as follows, where "msg" (string) corresponds to the XML message:

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

uxbService.removeUxbEntry(msg);
```

The XML message follows the example below:

```
<uxb_entity uuid=STRING language=STRING destinations=STRING
objectType=STRING command=STRING />
```

Property	Description	Example	Required field
Uuid	Unique identifier of the object, for example, fs_id	12345	Yes
Destinations	Target(s) of the message (comma-separated)	postgres	Yes
Command	Command to be executed by the adapter	delete	Yes
Language	Language of the message	DE (German)	No
objectType	Object type evaluated by the adapter (e.g. News, Products)	news	No

2.2 Adapters

Adapters are also used to read out the data from the JMS messages and to write them to the selected repositories. In the tutorial, two adapters are implemented as web applications as an example, but other implementations (e.g. standalone Java) are also possible.

2.2.1 Feedback

FirstSpirit expects a response from the adapter in the form of an XML document after a message has been written to a repository. The response is expected in the case of both successful and failed processing. The XML document is structured as follows:

```
<uxb_entity command=STRING createTime= STRING destinations=STRING
finishTime=STRING language=STRING path=STRING schedulerId=STRING
startTime=STRING status=STRING uuid=STRING ><uxb_error>STRING
</uxb_error></uxb_entity>
```

Property	Description	Example	Required field
destinations	The target repository to which the object has been or was to be written.	postgres	Yes
startTime	Timestamp for the start of the action (appended to the XML document by FirstSpirit)	1314567899516	Yes
finishTime	Timestamp for completion of the command	1314567899516	Yes
path	FirstSpirit internal path (appended to the XML document by FirstSpirit during the action)	the/Path/to/	Yes

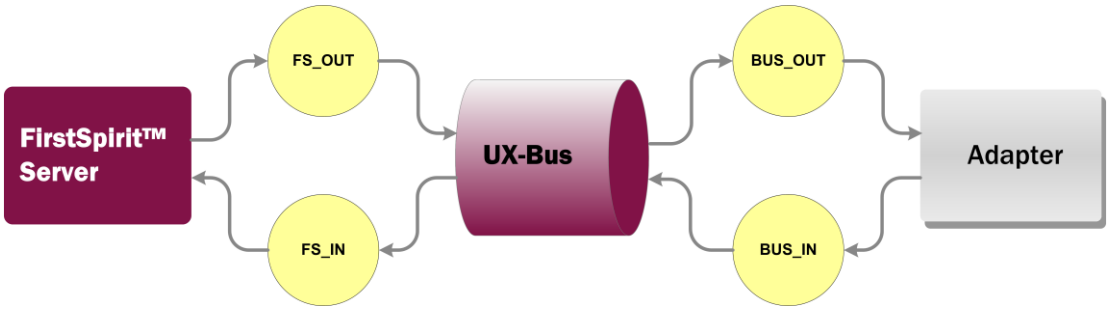
status	Status of the action. Possible values: "OK" if successful, "FAIL" if the action fails	OK	Yes
uuid	Unique identifier of the object, for example, fs_id	123456	Yes
schedulerId	Unique ID for the schedule (appended to the XML document by FirstSpirit during the action)	123456	Yes
command	Command executed by the adapter	delete	No
language	Language of the message	DE (German)	No
createTime	Timestamp for the creation of the action (appended to the XML document by FirstSpirit during the action)	1314567899516	No
uxb_error	The container element for the error message present in the event of an error	com.mongodb. MongoException	No

2.3 WebApplication

Through the open architecture of UX-Bridge and the fact that the type and number of repositories is not preassigned, the technology and the framework for developing the WebApplication can be freely selected. It is useful to base the selection of technology and the framework both on the application and the knowledge/company standards that are in place.

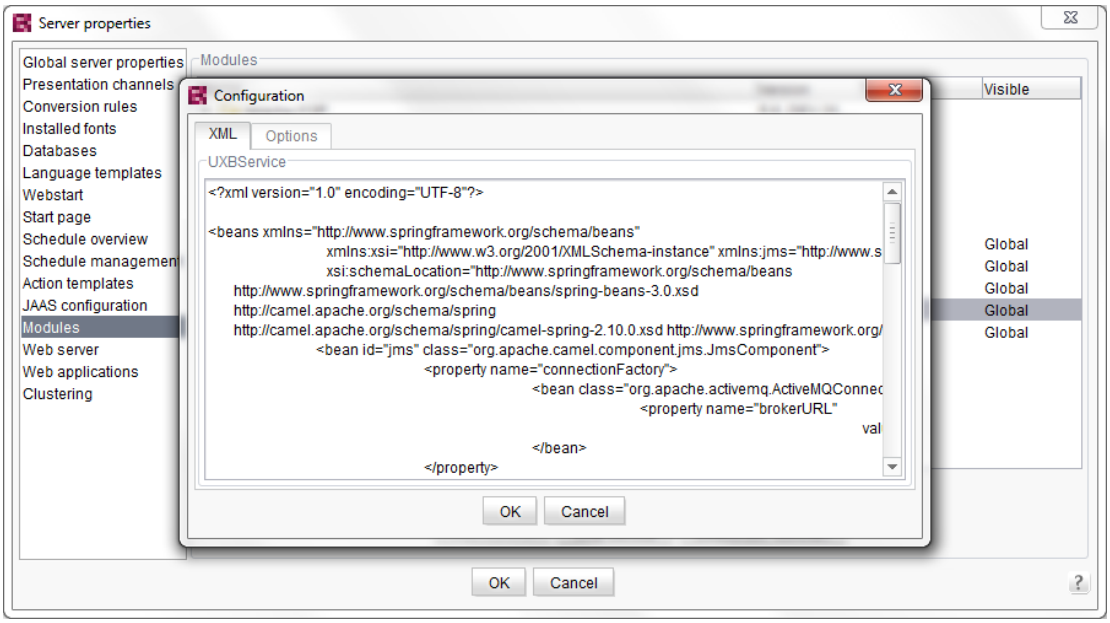
2.4 Routing

UX-Bridge uses Apache Camel to route messages. Java Message Service (JMS) is used for the transportation and message protocol. The two participating components of the FirstSpirit Server and adapters assume both the role of a producer, which generates messages and makes them available in an end point, and the role of a consumer, which retrieves the messages from an end point and processes them further. The UX-Bus, in this scenario, simply takes over the routing of messages between the participating end points.



2.4.1 End points in FirstSpirit

The configuration of the UXB service can be accessed via the FirstSpirit Server configuration and the module subitem. UXB service has to be selected in the expanded module tree; the "Configure" button is used to open the service configuration (Spring DSL), which also contains the end points and a route. The configuration does not usually have to be adjusted unless the names of the end points configured in the bus are changed. In this case, they will also have to be adjusted in the configuration of FirstSpirit. You must use the Adapter-Statistics-Response-Route with the UxbServiceStatisticsResponseHandler bean if UX-Bridge is to use its own monitoring (see "Monitoring in the Schedule" in the Installation Manual).



Within Spring DSL, a Camel context is described which contains the routes and end points:

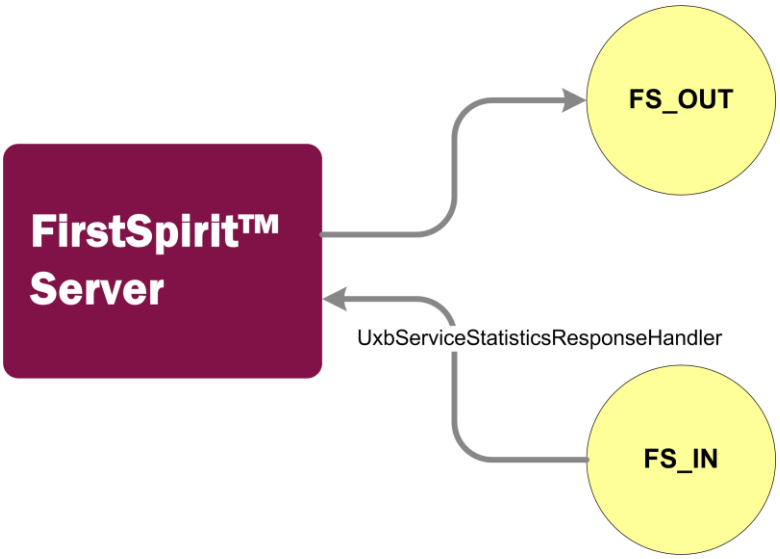
```
<camelContext xmlns="http://camel.apache.org/schema/spring"
id="camelContext" trace="false">
<package>com.espirit.moddev.uxbridge.service</package>
<template id="producerTemplate"/>
<endpoint id="FS-OUT" uri="activemq:topic:FS_OUT"></endpoint>
<onException>
          <exception>java.lang.Exception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

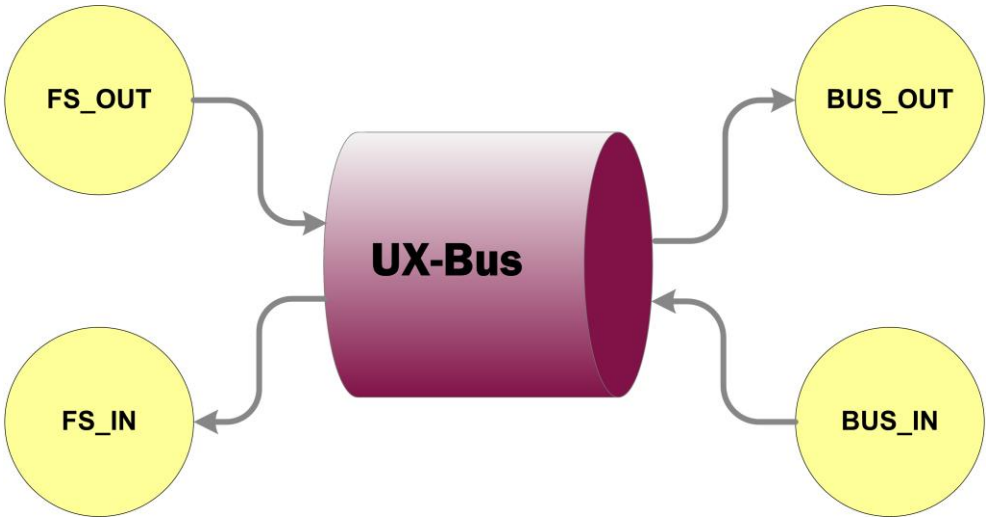
In the example, there is an end point with the ID "'FS_OUT", which serves as an end point for messages which are sent by the service.

Alongside that, the route "Adapter-Statistics-Response-Route" is defined, which consumes the messages from the end point "jms:topic:FS_IN". The messages are converted back into an object (UXBEntity) by the UXB service, and afterward the UxbServiceStatisticsResponseHandler is used on the objects so that these can again be evaluated for timing purposes, for instance.



2.4.2 Routing in the UX-Bus

The Spring DSL in the UX-Bus contains a Camel Context with four end points that form two routes. The first route goes from the end point of FirstSpirit to the end point of the adapter and the second from the end point of the adapter in reverse direction to the end point of the FirstSpirit service.




```
<camelContext trace="false"
xmlns="http://camel.apache.org/schema/spring">
 <route id="uxbridge-router">
<from uri="activemq:topic:FS_OUT"/>
<filter>
   <xpath>//uxb_entity[contains(@objecttype, 'products')]</xpath>
 <to uri="activemq:topic:VirtualTopic.BUS_OUT_mongo"/>
</filter>
<filter>
 <xpath>//uxb_entity[contains(@objecttype, 'news')]</xpath>
 <to uri="activemq:topic:VirtualTopic.BUS_OUT_postgres"/>
</filter>
 </route>
 <route id="uxbridge-router-response">
  <from uri="activemq:topic:BUS_IN"/>
<to uri="activemq:topic:FS_IN"/>
 </route>
</camelContext>
```

In the standard configuration, virtual end points are used (see http://activemq.apache.org/virtual-destinations.html). The advantage of virtual end points is that no modifications have to be carried out on the routing for additional adapters. The virtual end points follow the naming schema VirtualTopic.%destination-endpoint%“. Through the virtualization, messages are not read as in a queue by only one adapter; rather, all corresponding adapters receive the message.

The first route sends messages from the end point "activemq:topic:FS_OUT" in the

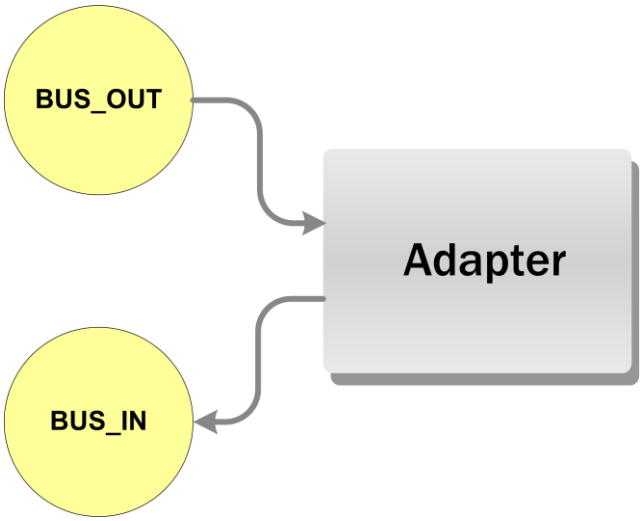
direction of the adapter to its end point "activemq:topic:VirtualTopic.BUS_OUT". Via XPath, for example, another differentiation is carried out for multiple adapters. "@objecttype" refers here to the JMS message header (see Creating and filling a presentation channel).

Messages that are sent by the adapters to the end point "activemq:topic:BUS_IN" in the FirstSpirit Service direction are redirected to the end point "activemq:topic:FS_IN".

In this configuration, usually adaptations to the route to the adapter are carried out only if the name of the end point is to be changed, or special routing mechanisms such as a case differentiation for multiple adapters is to be carried out.

2.4.3 End points in the adapter

The adapter can, in contrast to use in FirstSpirit and in UX-Bus, be freely implemented. The only requirement is that the adapter can receive JMS messages from an end point and can generate them in an additional end point. Two examples from adapters used with Camel can be found in the "Tutorials" below.



3 Tutorials

In the following tutorials, two examples are explained step-by-step. These examples can be adopted for your own projects or used as motivation for your own implementations.

A freshly set up "Mithras Energy" project was used as a basis for the examples. It is included as a sample project in every FirstSpirit installation.

Current versions of the source code for the examples are available at: https://github.com/e-Spirit/uxbridge-samples

For these examples, basic knowledge of the following technologies is useful.

- FirstSpirit
- Spring
- JAXB
- Apache Tomcat
- Apache Camel
- Hibernate
- MongoDB
- Apache ActiveMQ

Apart from that, UX-Bus must be operating and accessible. Information on operating UX-Bus can be found in the installation documentation.

In the case of the applications, it is required to adapt the database configuration to the local conditions. Please read the relevant section to obtain information on the location of the respective configurations.

3.1 News widget scenario

In this example, a simple widget is created which displays the latest articles. The display is automatically updated via JavaScript as soon as new articles are added to the live repository.

FirstSpirit is the leading system; in other words, the pages are generated statically and stored on the server. The dynamic widget is integrated into the page by JavaScript at runtime.

In the application example, the widget is integrated into the right column on the start page.

The source code for this example can be found in the Github repository under newsWidget.

https://github.com/e-Spirit/uxbridge-samples/tree/master/newsWidget

3.1.1 Web application

The web application provides only the JavaScript as a jQuery plugin and a service with JSONP support for updating the data. The basic framework of the widget is managed in FirstSpirit.

The web application was created using the Grails web framework, version 2.1.0.

3.1.1.1 Configuration

All configuration files are in the folder typically used for Grails applications <grailswidget>/grails-app/config.

At this point, the important files are DataSource.groovy and Config.groovy

3.1.1.1.1 DataSource.groovy

Here, the database connections are configured for the different environments (test, development and production).

3.1.1.1.2 Config.groovy

Here, the connection to MongoDB is also configured alongside the URLs for the different environments.

3.1.1.1.3 UrlMappings.groovy

Two mappings were added here:

"/rest/v1/articles" refers to the "list" action of the ArticleRestController.

"/rest/v1/article/$id" refers to the "show" action of the ArticleRestController.

3.1.1.2 Domain class

This application has an individual domain class: com.espirit.moddev.examples.uxbridge.widget.Article

Grails, like the adapter, uses the Hibernate persistence framework. Therefore, it is necessary to take care to use the same names for the attributes, tables and indices that were already used in the adapter.

3.1.1.3 Rest controller

The com.espirit.moddev.examples.uxbridge.widget package contains the ArticleRestController. Via this controller, the widget loads the list of articles.

The ArticleRestController provides the two methods "list" and "show".

3.1.1.3.1 Method: list

This method returns a certain number of articles in JSONP format.

3.1.1.3.2 Method: show

This method provides an article based on the FirstSpirit ID and the language in JSONP format. If no article is found for the parameters passed, the method delivers a 404 error code.

3.1.1.4 Service

The ArticleService is in the package com.espirit.moddev.examples.uxbridge.widget. For this example, the two methods "getLatestArticles" and "ellipsis" have been implemented.
The ArticleService is used in the ArticleRestController.

3.1.1.4.1 getLatestArticles

The method returns the latest articles from the live repository

3.1.1.4.2 ellipsis

This method is used to shorten the widget text to a specific number of characters.

3.1.1.5 SQL and NoSQL

In contrast to the adapters, an adaptation of the web application source code is not usually necessary. Thanks to the use of the Grails framework, the domain classes can be saved in a relational and a NoSQL database.

3.1.1.6 Starting the application example

The application is started via the command line:

```
grails run-app
```

To start the application with the MongoDB live repository, the corresponding environment must be specified

```
grails mongo run-app
```

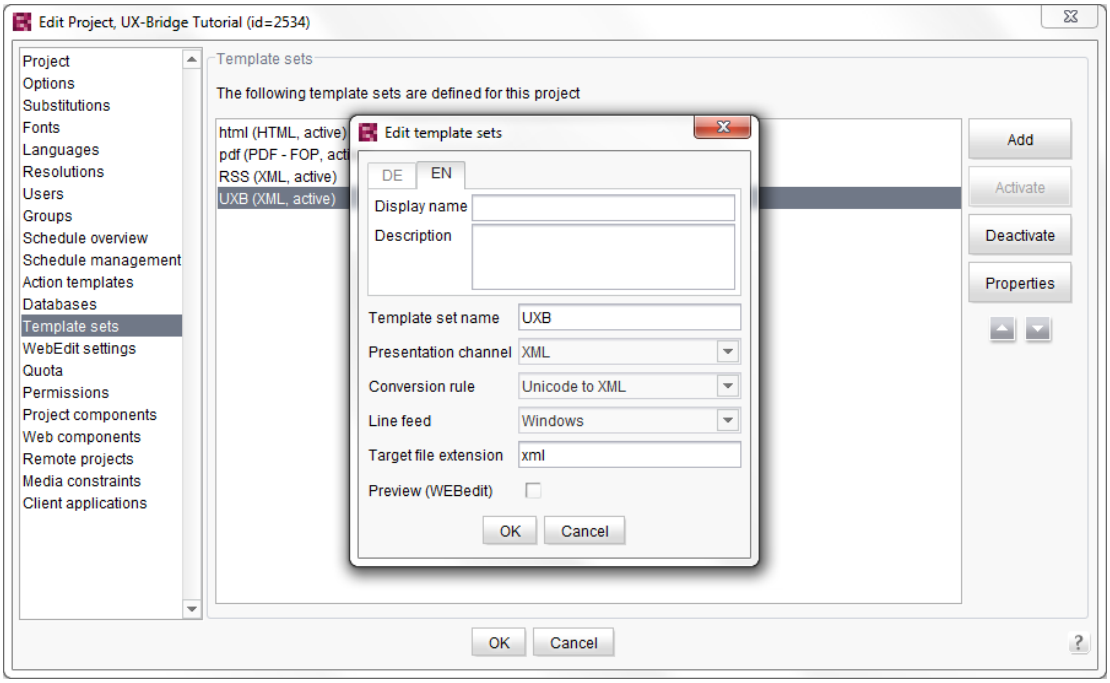
3.1.2 FirstSpirit development

The News widget is integrated in this tutorial in the standard "Mithras Energy" project. Therefore, import this first and carry out all the subsequent changes in this project.

The complete, finished sample project is also provided under the name "uxbridge_tutorial_newsWidget.tar.gz" and can be used to view the template code and the settings.

3.1.2.1 Project configuration

The first step is to create a new template set for UX-Bridge in the project configuration, which is to be configured as follows.



In order to send messages to the UX-Bus, the corresponding template, which is to generate the messages, contains the fields that were defined in the data model and are to be output in XML format (compare to Section 2.1.3, "Creating and filling a presentation channel").

3.1.2.2 Project settings

In the project settings, the URL for the web application is defined from which suitable articles are dynamically reloaded later and shown in the widget. For this purpose, the template for the project settings is expanded by one field.

```
<CMS_GROUP>
      <LANGINFOS>
        <LANGINFO lang="*" label="UX-Bridge"/>
      </LANGINFOS>

      <CMS_INPUT_TEXT name="ps_baseURL_UXB" hFill="yes"
singleLine="no" useLanguages="no">
        <LANGINFOS>
          <LANGINFO lang="*" label="Base URL UXB Widget"
description="Insert the base URL for the UX Bridge Widget"/>
          <LANGINFO lang="DE" label="Basis URL UXB Widget"
description="Geben sie hier die Basis URL für die UX Bridge Widget
an"/>
        </LANGINFOS>
      </CMS_INPUT_TEXT>
    </CMS_GROUP>
```

As soon as this is done, the URLs can be updated for UX-Bridge. This base URL is that of the Grails application widgetExample, in other words for example:

http://localhost:8080/widgetExample

In the global content area, a new global page based on the template "multilanguagelabel" has to be created with the unique name "latestarticles".

German: Neueste Artikel

English: Latest articles

3.1.2.3 Page templates

An input component has been added to the page templates to be used by UX-Bridge in the example. This input component enlarges the marginal column to give more space to the widget that is to be integrated. This is done because the default width of the marginal column is too small to show the News widget in an appropriate resolution.

```
<CMS_GROUP>
      <LANGINFOS>
        <LANGINFO lang="*" label="UX Bridge Features"
description="Enable/Disable UX Bridge features for this page"/>
        <LANGINFO lang="DE" label="UX Bridge Funktionen"
description="Aktivieren/Deaktivieren von UX Bridge-
Funktionalitäten für diese Seite"/>
      </LANGINFOS>

      <CMS_INPUT_TOGGLE
        name="pt_enableUxBridgeLayout"
        type="radio"
        hFill="yes"
        preset="copy"
        singleLine="no"
        useLanguages="no">
        <LANGINFOS>
          <LANGINFO lang="*" label="Enable UX Bridge layout for
this page" description="Enables UX Bridge for this page"/>
          <LANGINFO lang="DE" label="UX Bridge Layout für diese
Seite aktivieren" description="UX Bridge Layout für diese Seite
aktivieren"/>
        </LANGINFOS>
        <OFF>
          <LANGINFO lang="*" label="No"/>
          <LANGINFO lang="DE" label="Nein"/>
        </OFF>
        <ON>
          <LANGINFO lang="*" label="Yes"/>
          <LANGINFO lang="DE" label="Ja"/>
        </ON>
      </CMS_INPUT_TOGGLE>

    </CMS_GROUP>
```

These changes also serve to activate UX-Bridge on only the pages that integrate it.

In addition to expanding the input form, the code also has to be adapted in the template.

At the beginning of the page template, the body area needs to be stored in a variable so that the variables set in the body are already available at the beginning of the page template; for example:

```
$CMS_SET(set_pt_bodyright)$$CMS_VALUE(#global.page.body("Content
right"))$$CMS_END_SET$
$CMS_SET(set_pt_bodyright, set_pt_bodyright.toString)$
```

The output at the former location of the body is then, for example:

```
$CMS_VALUE(set_pt_bodyright)$
```

The header of the page template must still be expanded by adding the following call, which initializes the page variables of UX-Bridge:

```
$CMS_SET set_pt_insertIntoHead,"")$
```

The HTML header must then be expanded to include the following call in order to import the Java scripts:

```
$CMS_VALUE(set_pt_insertIntoHead)$
```

3.1.2.4 Section template

The News widget is integrated into the marginal column of the desired page via a section template. In addition, a new section template with the name "uxb_widget" is created first as follows.

```
<CMS_HEADER>
</CMS_HEADER>

$CMS_IF(pt_enableUxBridgeLayout)$
$CMS_SET(set_st_insertIntoHead)$
          $CMS RENDER(template:"uxbridge widget head",
set_news_count:st_entries)$
$CMS_END_SET$

<div class="clearfix teasermodule uxbWidgetContainer">
          <div class="uxbWidgetHeader">

<span>$CMS_VALUE(#global.gca("latestarticles"))$</span>
          </div>
    <div id="uxbWidgetContent"></div>
</div>
```

```
$CMS_SET(#global.pageContext["set_pt_insertIntoHead"],
set_st_insertIntoHead.toString)$
$CMS_END_IF$
```

The clearfix and teasermodule classes use CSS properties from Mithras and are designed to be used in the "Content right" area of the standard page template or on the homepage.

The Number input component placed in the form lets you define how many news entries are to be shown in the widget.

```
<CMS_MODULE>


  <CMS_INPUT_NUMBER
    name="st_entries"
    allowEmpty="no"
    hFill="yes"
    max="20.0"
    min="1.0"
    preset="copy"
    singleLine="no"
    useLanguages="yes">
    <LANGINFOS>
      <LANGINFO lang="*" label="Number of entries"
description="Choose the number of entries shown in the widget"/>
      <LANGINFO lang="DE" label="Anzahl der Einträge"
description="Anzahl der Einträge im Widget"/>
    </LANGINFOS>
  </CMS_INPUT_NUMBER>


</CMS_MODULE>
```

3.1.2.5 Format template

In the example, a new format template (uxbridge_widget_head) is used which contains the required JavaScript and CSS code. The parameters with which the jQuery plugin "uxb_widget" is initialized can be configured.

```
<script type="text/javascript"
src="$CMS_VALUE(ps_baseURL_UXB)$/static/bundle-
ui_head.js"></script>
```

```
<link rel="stylesheet"
href="$CMS_VALUE(ps_baseURL_UXB)$/static/bundle-ui_head.css"/>


<script>
$(document).ready(function () {

$("#uxbWidgetContent").uxb_widget({lang:'DE',url:"$CMS_VALUE(ps_ba
seURL_UXB)$/rest/v1/articles",speed:2000, fadeFrom: "#F7D358",
fadeTo: "white", count: $CMS_VALUE(set_news_count)$});
});
</script>
```

3.1.2.6 Creating a page

In order to use UX-Bridge, a new section of the "uxb_widget" type is integrated in any page, and, where appropriate, the section in the page template for the marginal column must also be allowed in advance.

3.1.2.7 Table and table template (XML)

Based on the Press_Releases table already defined in the schema, a table template should then be created which generates the XML that is forwarded to the UXB service:

```
<uxb_entity
          uuid = String
          destinations = String
          language = String
          command = String
          objectType = String
>
          <uxb_content>
                    <fs_id/>
                    <language/>
                    <url/>
                    <date/>
                    <headline/>
                    <subheadline/>
                    <teaser/>
                    <content/>
```

```
        </uxb_content>
</uxb_entity>
```

The child elements in the uxb_content tag simultaneously function as the content fields which are written by the adapter to the connected content repository, and therefore should also be taken into account during creation of the data structure.

The sample code here represents only the structure of this table template. The complete code can be found in the sample project in the table template with the reference name "Products.press_releases".

3.1.2.8 Deployment

In the project example, the generation of JMS messages, and with that entries in the connected content repositories, can be started automatically via a workflow directly from the data source. Likewise, it is possible to delete objects in FirstSpirit and in the connected content repositories directly from the data sources using an additional workflow. To do so, in addition to scripts, the workflows use table queries and schedules, which must first be configured.

3.1.2.8.1 Creating table queries

Table queries have to be created for the generation of a data record and all data records for the News table. The queries for a data record still have to have a limitation on the column "fs_id", with the "ID" parameter that will be created.

The complete code can be found in the sample project in the table query with the reference name "Products.pressdetailfilter".

3.1.2.8.2 Creating a schedule

A new schedule has to be created which, alongside the generation of JMS messages for the UXB service, also takes over the generation and deployment of overview pages. In addition, a generation action, which generates the overview pages, must first be added to the schedule. The Delta deployment expands on this action during runtime by adding the detail page of the data record currently to be generated. Afterward, a script action is required which activates UX-Bridge:

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

A partial generation should then take place in the subsequent generation to be created. Page and data record are later entered automatically through the Delta deployment so that only the desired JMS message is generated. The web pages can then be deployed as usual.

If the processing time is measured for the messages in the bus until deployment on the website, then in the final step the action "UX-Bridge Statistics Report" has to be added, which contains the following script:

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

The service waits the maximum amount of time defined in the UX-Bridge service configuration until the adapters' responses are evaluated. If there is no response within this time frame, then the message is classified as having a delivery error. Since response times can vary depending on message and system, this value can be configured.

3.1.2.8.3 Importing workflow scripts

In the next step, the required workflow scripts must be imported:

- uxb_content_release_init
- uxb_content_release_script
- uxb_content_delete_init
- uxb_content_delete_script

The Init scripts in this case initialize variables and write these to the session so that the methods of the UXB module, which are queried in the other scripts, can access them.

The following parameters have to be configured in uxb_content_release_init:

Parameter	Example value	Description
detail_page	pressreleasesdetails	Page reference of the page which contains the JMS messages

query_uid	Products.pressdetailsfilter	Table query which generates all the data records
single_query_uid	Products.pressdetailfilter	Table query which contains the ID of the data record that is to be generated as a parameter
query_param	Id	Parameter name of the table query
schedule_name	UX-Bridge	Name of the schedule that is to generate the JMS messages
scheduler_uxb_generate	UX-Bridge Generate	Name of the generation action for the JMS messages
scheduler_generate	Generate	Name of the generation action for the HTML pages

The script "uxb_content_release_script" then starts the previously configured schedule and carries out the defined transition.

The following parameters have to be configured in uxb_content_delete_init:

Parameter	**Example value**	**Description**
Destinations	postgres	Name of the content repositories from which the data record is to be deleted
transition_name	release	Name of the transition in the workflow (see "Workflow") which is to be switched to after the content_delete_script

object_type	news	Type of object that is to be deleted

Within the following script, "uxb_content_delete_script", the selected data record is deleted in FirstSpirit and a message is sent via the UXB service and the bus to the connected content repository, which triggers the delete action there.

3.1.2.8.4 Importing workflows

In order to provide the editor with a simple way to run the previously defined scripts on a data record, both workflows "uxb_content_release" and "uxb_content_delete" from the demo project are used. The workflows then run the desired operations (release, delete) in FirstSpirit as well as via UX-Bridge in the configured content repository.

3.1.2.8.5 Complete alignment process

In the FirstSpirit sample project, the complete alignment process is implemented in the "UX-Bridge Full Deployment" schedule.

Information on the procedure for the complete alignment process are available in Section 2.1.4.2, "Complete alignment process", page 14.

3.1.3 Adapters

This example contains two adapters: one for a relational database (PostgreSQL) and one for a NoSql database (MongoDB).

Under https://github.com/e-Spirit/uxbridge-samples/newsWidget/adapter, alongside the projects for the two adapters (Hibernate, mongodb), there is a third project which contains the Java classes that are used in both adapters.

3.1.3.1 JAXB

JAXB is used to process the exchange format. The corresponding classes are located in the project https://github.com/e-Spirit/uxbridge-samples/newsWidget/adapter/base in the package com.espirit.moddev.uxbridge.entity.

JAXB makes it easy to work with Java objects without having to think about parsing the XML. Similar to JPA, work here is done with annotations.

```
@XmlRootElement(name = "uxb_entity")
@XmlAccessorType(XmlAccessType.FIELD)
public class UXBEntity {
          @XmlAttribute
          private String uuid;
          @XmlAttribute
          private String language;
          @XmlAttribute
          private String destinations;
          @XmlElement(type = UXBContent.class)
          private UXBContent uxb_content;
          @XmlAttribute
          private String command;
          @XmlAttribute
          private long createTime;
          @XmlAttribute
          private long finishTime;
```

3.1.3.1.1 DateType: XmlAdapter for the date format

Dates are input in the FirstSpirit presentation channel according to the format "yyyy-MM-dd'T'HH:mm:ssZ". The DateAdapter class has been implemented so that this format can be read into the JAXB classes. This class is located in the com.espirit.moddev.examples.uxbridge.widget.entity.type package.

```
@XmlElement()
@XmlJavaTypeAdapter(value = DateAdapter.class, type = Date.class)
private Date date;
```

3.1.3.1.2 UXBEntity and UXBContent

Both classes implement the part prescribed by UX-Bridge (UXBEntity) and the project-specific part (UXBContent) of the exchange format.

3.1.3.2 Relational database

The adapter for relational databases was implemented with the aid of Hibernate. In this example, PostgreSQL is used; the adapter, however, should also function with other Hibernate-supported databases.

3.1.3.2.1 Domain class: Article

The Article domain class is located in the project widgetExample/adapter/base in the package com.espirit.moddev.examples.uxbridge.widget.

The class has a generated ID:

```
@Id
@GeneratedValue(strategy = GenerationType.AUTO)
private Long id;
```

The ID attribute is used so that the domain class is compatible with Grails implementation in the web application.

In order to prevent complex database structures, in this example, an object was generated for each language. This means that the FirstSpirit ID is no longer unique in this context. The data is therefore accessed via the FirstSpirit ID (aid) and the language.

The use of a compound primary key would make sense here. In this example, however, due to the complexity, this option is intentionally not used.

Note that the ID changes after deleting and reinserting an article into the live repository. Therefore, it is necessary to always use the FirstSpirit ID and the language for access.

3.1.3.2.2 ArticleHandler

Access to the database is made in the ArticleHandler (com.espirit.moddev. examples.uxbridge.widget.jpa package). It handles reception and editing of the data. In the example, for each of the supported commands, a unique method was implemented in the handler.

3.1.3.2.2.1 Command: add

For saving or updating a press release in the live repository.

3.1.3.2.2.2 Command: delete

For deleting a press release in the live repository.

3.1.3.2.2.3 Command: cleanup

For deleting all press releases which are older than the date indicated.

3.1.3.2.3 Configuration

The configuration for this adapter is located in the WEB-INF/applicationContext.xml file. In addition to the database, the JMS, the ArticleHandler and the Camel routes are configured in this Spring XML file.

3.1.3.2.3.1 CamelContext

In this context, you configure the messages that are of interest to this adapter and are processed by it.

```
<camelContext id="camelContext" trace="false"
xmlns="http://camel.apache.org/schema/spring">
<package>com.espirit.moddev.examples.uxbridge.newswidget.entity</p
ackage>
<onException>
<exception>java.lang.Exception</exception>
<handled>
<constant>true</constant>
</handled>
<to
uri="adapterReturn:jms:topic:BUS_IN?destination=mongodb&amp;bodyVa
lue=bodyTemp " />
</onException>
<route id="uxbridge-commands">
<from uri="jms:topic:BUS_OUT" />
<filter>
```

```
<xpath>//uxb_entity[contains(@destinations, 'mongodb')]</xpath>
<filter>
<xpath>//uxb_entity[@objectType = 'news']</xpath>
<camel:setHeader
headerName="bodyTemp"><simple>${body}</simple></camel:setHeader>
<filter>
<xpath>//uxb_entity[@command = 'add']</xpath>
<convertBodyTo
type="com.espirit.moddev.examples.uxbridge.newswidget.entity.UXBEn
tity" />
<bean ref="articleHandler" method="add" />
</filter>
<filter>
<xpath>//uxb_entity[@command = 'delete']</xpath>
<convertBodyTo
type="com.espirit.moddev.examples.uxbridge.newswidget.entity.UXBEn
tity" />
<bean ref="articleHandler" method="delete" />
</filter>
<filter>
<xpath>//uxb_entity[@command = 'cleanup']</xpath>
<convertBodyTo
type="com.espirit.moddev.examples.uxbridge.newswidget.entity.UXBEn
tity" />
<bean ref="articleHandler" method="cleanup" />
</filter>
<to uri="adapterReturn:jms:topic:BUS_IN?destination=mongodb" />
</filter>
</filter>
</route>
<route>
<from uri="jms:topic:BUS_IN" />
<to uri="stream:out" />
</route>
</camelContext>
```

A detailed explanation of how to create the response can be found in Section 3.5, "Using the Camel component to generate a response".

3.1.3.2.3.1.1 Receiving messages

You can use the From tag (<from uri=" activemq:Consumer.newsWidgetHibernate. VirtualTopic.BUS_OUT" />) to configure the URI used to read in messages. At this location, a virtual end point is used (see http://activemq.apache.org/virtual-destinations.html). The advantage of virtual end points is that no modifications have to be carried out on the routing for additional adapters. New, virtual end points only have to follow the "Consumer.%any adapter name%.VirtualTopic.%source termination point%" naming schema. Through the virtualization, messages are not read as in a queue by only one adapter; rather, all corresponding adapters receive the message.

If, for example, the new adapter "myAdapter" is also to consume messages that are delivered at the end point FS_OUT, then a possible end point might look as follows:

activemq:Consumer.myAdapter.VirtualTopic.BUS_OUT

3.1.3.2.3.1.2 Filters for the live repository

The XPath expression (//uxb_entity[contains(@destinations, 'postgres')]) is used to filter the messages which are to be written to this live repository.

3.1.3.2.3.1.3 Filtering the object type

The expression //uxb_entity[@objectType = 'news'] is used to limit messages to News type objects.

3.1.3.2.3.1.4 Adding and deleting articles

With these expressions, filtering is done according to the corresponding command. "//uxb_entity[@command = 'add']" adds messages to the repository, and "//uxb_entity[@command = 'delete']" deletes messages from the repository.

Before the actual method query, JAXB and the Camel instruction <convertBodyTo type="com.espirit.moddev.examples.uxbridge.widget.entity.UXBEntity"/> are used to convert the exchange format. The corresponding method in the ArticleHandler is then queried using a UXBEntity type object.

3.1.3.2.3.2 ArticleHandler

The ArticleHandler is the part of the adapter which processes the command and writes the article to or deletes it from the repository.

The ArticleHandler requires the EntityManagerFactory for access to the database as well as the CamelContext and the name of the routes on which the messages can be sent back to FirstSpirit.

3.1.3.3 MongoDB

For the NoSQL live repository, the MongoDB database driver was used exclusively. The use of a persistence framework was deliberately omitted, since the DB structure of the web application would have had to be recreated in the adapter.

3.1.3.3.1 Domain class: Article

The MongoDB adapter uses the same domain class that is used by the Hibernate adapter. The JPA annotations are not taken into account in this case.

3.1.3.3.1.1 ID generation

In the web application, Grails GORM is used for the database access. In order for the adapter to use the identical database structure, a helper method generateIdentifier had to be introduced. In this method, IDs are managed via an extra collection (http://www.mongodb.org/display/DOCS/Collections).

3.1.3.3.2 ArticleHandler

The procedure in the ArticleHandler is no different from the Hibernate ArticleHandler procedure (see also Section 3.1.3.2.2, "ArticleHandler").

3.1.3.3.3 Configuration

The configuration is only marginally different from the configuration of the Hibernate adapter.

The destination filter filters messages for the mongodb destination. The parameters for the connection to the database are transferred directly to the ArticleHandler.

3.1.3.4 Starting the sample adapters

The API can be loaded into the local Maven repository using the following call:

```
mvn install:install-file -Dfile=<path-to-file> -DgroupId=
com.espirit.moddev.uxbridge -DartifactId= uxbridge-camel-component
-Dversion=<version> -Dpackaging=jar
```

An implementation example might look like the following:

```
mvn install:install-file -Dfile=D:\ uxbridge-camel-component-
1.2.4.1133.jar -DgroupId=com.espirit.moddev.uxbridge -
DartifactId=uxbridge-camel-component -Dversion=1.2.4.1133 -
Dpackaging=jar
```

The sample adapters can be established via the command line:

```
mvn package
```

The War file resulting from this can be deployed on any servlet container (Tomcat, Jetty etc.).

Alternatively, the adapters can also be started via the command line:

```
mvn tomcat7:run
```

In order to adapt the port of the Tomcat which was started by this, the file pom.xml has to be adapted in the directory of the respective adapter.

3.1.3.5 Tests included

The sample project includes unit and integration tests. For the tests, an In-Memory database and jMockMongo (https://github.com/thiloplanz/jmockmongo) are used. The jMockMongo jar file has to be imported into the local repository or the following Maven repository has to be used so that the tests for the MongoDB adapter can be started:

```
<repositories>
   <repository>
        <id>thiloplanz-snapshot</id>
        <url>http://repository-
thiloplanz.forge.cloudbees.com/snapshot/</url>
   </repository>
</repositories>
```

The dependency must then appear as follows:

```
<dependency>
     <groupId>jmockmongo</groupId>
     <artifactId>jmockmongo</artifactId>
     <version>0.0.1-SNAPSHOT</version>
     <scope>test</scope>
 </dependency>
```

The integration tests can be started with the following call:
mvn verify -Pintegration-test

3.2 News widget scenario without programming

As in the previous example, a simple widget is created in this example that displays the latest articles. The difference stems from the way the adapter is implemented. It has been implemented without programming, using Camel alone.

Using the ArticleHandler is not necessary. The functions of the ArticleHandler are replaced by CamelContext configuration in this case.

Most items are identical to the previous example. Therefore, only the changes required to implement the example without programming are described in the following.

3.2.1 CamelContext

In some spots, the explanations for CamelContext are identical to those in the previous example. The entire Context is explained below regardless.

```
<camelContext id="camelContext" trace="false"
xmlns="http://camel.apache.org/schema/spring">
<onException>
<exception>java.io.IOException</exception>
<handled><constant>true</constant></handled>
<to
uri="adapterReturn:jms:topic:BUS_IN?destination=mongodb&amp;bodyVa
lue=bodyTemp" />
</onException>
<route id="uxbridge-commands">
```

```
<from uri="jms:topic:BUS_OUT" />
<filter>

<xpath>//uxb_entity[contains(@destinations, 'mongodb')]</xpath>
<filter>
<xpath>//uxb_entity[@objectType = 'news']</xpath>
<camel:setHeader headerName="bodyTemp">
<simple>${body}</simple>
</camel:setHeader>
<filter>
<xpath>//uxb_entity[@command = 'add']</xpath>
<camel:split stopOnException="true">
<camel:xpath>/uxb_entity/uxb_content/text()</camel:xpath>
<camel:convertBodyTo type="java.lang.String" />
<camel:setBody>
<language
language="groovy"><![CDATA[request.getBody().substring(request.get
Body().indexOf("<![CDATA[")+9,request.getBody().lastIndexOf("]]]]>
<![CDATA[>"))]]></language>
</camel:setBody>
<camel:convertBodyTo type="com.mongodb.DBObject" />
<to
uri="mongodb:myDb?database=newsWidget&amp;collection=article&amp;o
peration=save" />
</camel:split>
</filter>
<filter>
<xpath>//uxb_entity[@command = 'delete']</xpath>
<camel:split stopOnException="true">
<camel:xpath>/uxb_entity</camel:xpath>
<camel:convertBodyTo type="java.lang.String" />
<camel:setBody><camel:groovy>'{aid:'+request.getBody().substring(r
equest.getBody().indexOf('uuid=')+6,request.getBody().indexOf('"',
request.getBody().indexOf('uuid=')+6))+',"language":"'+request.get
Body().substring(request.getBody().indexOf('language=')+10,request
.getBody().indexOf('"',request.getBody().indexOf('language=')+10))
+'"}'</camel:groovy></camel:setBody>
<camel:convertBodyTo type="com.mongodb.DBObject" />
<to
uri="mongodb:myDb?database=newsWidget&amp;collection=article&amp;o
peration=remove" />
</camel:split>
```

```
</filter>
<filter>
<xpath>//uxb_entity[@command = 'cleanup']</xpath>
<camel:split stopOnException="true">
<camel:xpath>/uxb_entity</camel:xpath>
<camel:convertBodyTo type="java.lang.String" />
<camel:setBody><camel:groovy>'{"lastmodified":{$lt:'+request.getBo
dy().substring(request.getBody().indexOf('createTime=')+12,request
.getBody().indexOf('"',request.getBody().indexOf('createTime=')+12
))+'}}'</camel:groovy></camel:setBody>
<camel:convertBodyTo type="com.mongodb.DBObject" />
<to
uri="mongodb:myDb?database=newsWidget&amp;collection=article&amp;o
peration=remove" />
</camel:split>
</filter>
<to uri="adapterReturn:jms:topic:BUS_IN?destination=mongodb" />
</filter>
</filter>
</route>
</camelContext>
```

CamelContext begins with exception handling. The way exceptions are processed is defined in the associated onException tag. In this case, only a java.io.Exception is specified. However, multiple exceptions may occur at the same time.

The exception is handled if the handled tag is set to true. In this case, the exception is no longer thrown and the entire process is not interrupted. This corresponds to a try-catch block for all routes. An explicit try-catch block for specific areas is possible if exceptions are to be handled separately for them.

What happens in the event of an exception is then specified. In this case, a message is sent to BUS_IN. The exact structure is described in Section 3.5, "Using the Camel component to generate a response".

The passed XML is analyzed within the route using filter and xpath and the corresponding calls are made.

A DBObject has to be generated in order to be able to communicate with a Mongo database. It is generated using <camel:convertBodyTo type="com.mongodb. DBObject" />. A JSON object in the form of a string is expected as the transfer parameter. A JSON object is passed within the XML document for this purpose. The content of an XML tag, the JSON object in this case, is read out using text().

If the JSON object contains data that has already been interpreted by an XML parser, then the JSON has to be enclosed by a CDATA section to prevent unwanted interpretation. This section has to be removed before creating the DBObject. This can be done using <language language="groovy"><![CDATA[request.getBody().substring(request.getBody().indexOf("<![CDATA[")+9,request.getBody().lastIndexOf("]]]><![CDATA[>"))]]></language>.

To transmit the DBObject to the Mongo database, the following is called: <to uri="mongodb:myDb?database=newsWidget&collection=article&operation=save" />. The database, collection and operation are also passed as parameters.

The JSON objects are created in the delete and cleanup area using groovy. The required information (uuid,language,createTime) is parsed from the XML document's uxb_entity tag and placed in the corresponding spot in the JSON object. This makes it unnecessary to pass a JSON object within the XML document.

3.2.2 Adjustments in FirstSpirit

A slight adjustment in FirstSpirit is required in order to be able to use the News widget scenario without programming. As described earlier in the section, the information in JSON format has to be passed wrapped in an XML document.

3.2.2.1 Adding content

The Products.press_release UXB channel's database schema has to be adjusted in order to add content. The UXB channel must look like the following:

```
<?xml version="1.0" encoding="UTF-8" ?>
$CMS_SET(_id)$$CMS_VALUE(#row.id)$$CMS_VALUE(#global.language.hashCode())$$CMS_END_SET$
<uxb_entity uuid="$CMS_VALUE(#row.id)$"
language="$CMS_VALUE(#global.language)$"
destinations="postgres,mongodb" command="add" objectType="news">
<uxb_content><![CDATA[
{
"_id":$CMS_VALUE(_id)$,
"aid":$CMS_VALUE(#row.id)$,
"language":"$CMS_VALUE(#global.language)$",
"url":"$CMS_REF(#global.node, contentId:#row.getId(),abs:1,
templateSet:"html")$",
```

```
$CMS_IF(#global.preview)$"lastmodified":$CMS_VALUE(#global.now.get
TimeInMillis())$,

$CMS_ELSE$
"lastmodified":$CMS_VALUE(#global.getScheduleContext().getStartTim
e().getTimeInMillis())$,

$CMS_END_IF$

$CMS_IF(!cs_date.isEmpty)$"date":{"$date":"$CMS_VALUE(cs_date.form
at("yyyy-MM-dd'T'HH:mm:ss'Z'"))$"},$CMS_END_IF$

$CMS_IF(!cs_headline.isEmpty)$"title":"$CMS_VALUE(cs_headline.conv
ert2)$",$CMS_END_IF$

$CMS_IF(!cs_subheadline.isEmpty)$"subHeadline":"$CMS_VALUE(cs_subh
eadline.convert2)$",$CMS_END_IF$

$CMS_IF(!cs_teaser.isEmpty)$"teaser":"$CMS_VALUE(cs_teaser.convert
2)$",$CMS_END_IF$

$CMS_IF(!cs_content.isEmpty)$"content":"$CMS_FOR(section,
cs_content)$$CMS_SET(tmp)$$CMS_VALUE(section)$$CMS_END_SET$$CMS_SE
T(tmp,tmp.toString)$$CMS_VALUE(tmp.convert2)$$CMS_END_FOR$"$CMS_EN
D_IF$

}]]>

</uxb_content>

</uxb_entity>
```

As described previously, an XML document is generated with the JSON object embedded inside.

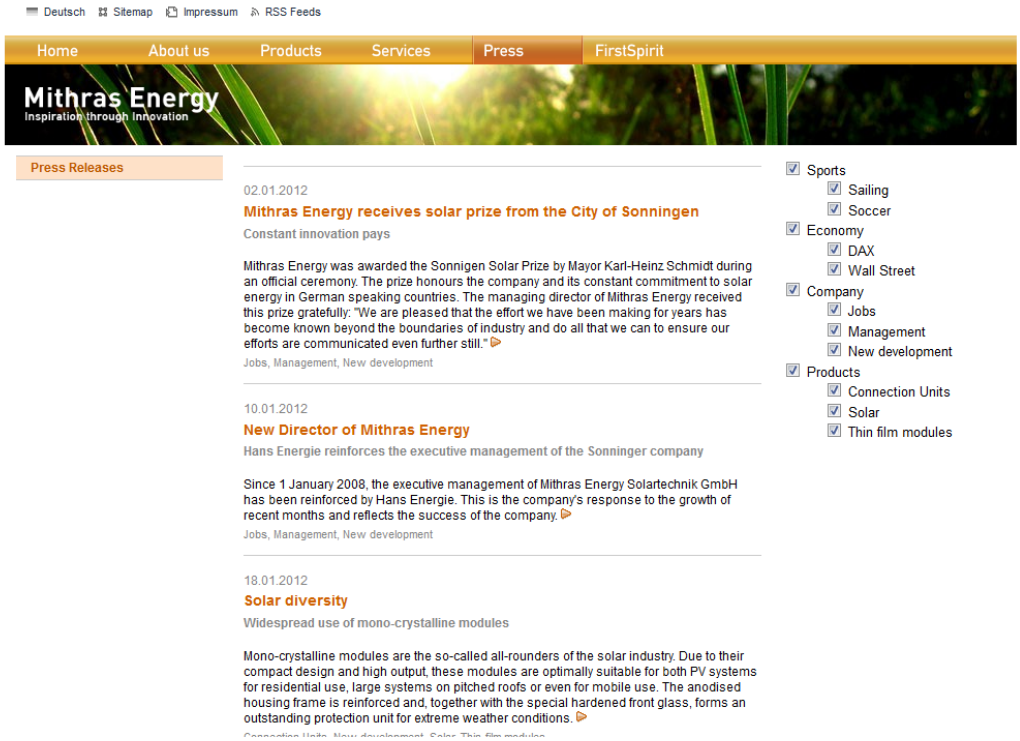
3.3 News Drill-Down scenario

In this example, an overview of press releases is generated which can be filtered by category using a drill-down function.

The web application in this case is the leading system. In other words, the drill-down function and the overview page are created dynamically; the detail pages and the remaining pages are generated statically. Header and footer are integrated as HTML fragments in the overview page. These fragments are likewise generated by FirstSpirit.

The news articles, categories and meta categories are written to a content repository with the aid of UX-Bridge, to which the web app has access. This implementation is kept simple for the example, and is not performance-optimized; as with every update of a news item, both the category and the meta category are accessed and updated if necessary. In a real adapter, of course, you would optimize them; categories and meta categories would be read out once only, and an update would be carried out only in the event of changes to the categories. All categories and meta categories are shown in the web app in a drill-down menu, where you can select checkboxes to

mark the categories for which the news is to be shown. With every selection and deselection of a checkbox, an AJAX query is sent. The returned HTML is integrated into the news list on the page. Pagination guarantees clarity. This likewise uses AJAX, because the number of the news items to be listed varies with the selected categories.



The sample application newsExample consists of the adapter (Hibernate), the web application (Grails) and the FirstSpirit sample project.

3.3.1 Web app development

The web application was created with the web framework Grails, version 2.1.0.

3.3.1.1 Configuration

All configuration files are in the folder typical for Grails applications: <newsExample>/grails-app/conf.

In this area, the important files are DataSource.groovy and Config.groovy

3.3.1.1.1 DataSource.groovy

Here, the database connections are configured for the different environments (test, development and production).

3.3.1.1.2 Config.groovy

Here, the URLs for the navigation generated from FirstSpirit are defined.

3.3.1.2 Domain classes

Create three domain classes with the names News, Category and MetaCategory. Grails, like the adapter, uses the Hibernate persistence framework. Therefore, it is necessary to make sure that the same names are used for the attributes, tables and indices that were already used in the adapter.

3.3.1.3 Rest controller

Create the appropriate controller for the domain class news and implement the "list" method. Via this method, the web -application loads the list of articles.

3.3.1.3.1 Method: list

This method is used to render the gsp of the same name.

3.3.1.3.2 Method: listNews

This method renders the gsp template "newsListing" for a certain list of news, which is fetched from the FilterService.

3.3.1.3.3 Method: drilldown

This method renders the template of the same name, providing the drill-down menu and the JavaScript necessary for it.

The drill-down menu is rendered directly in the page when the page is viewed.

The JavaScript contained within it uses jQuery and manages checking and unchecking of the checkboxes for the individual categories and meta categories.

With every click on a checkbox, an AJAX query with the currently selected categories is sent to the controller's "listNews" method. The HTML returned is then inserted into the news overview page of the div intended for it. The list of news remains clearly arranged and is edited using pagination, which likewise dynamically reloads the correct pages with the correct articles via AJAX queries.

3.3.1.4 Service

3.3.1.4.1 FilterService

This service provides methods to retrieve news according to their categories.

3.3.1.4.1.1 Method: filter

This method returns a map with the following keys:

newsInstanceList: A list of news in the queried categories

newsInstanceTotal: The total number of news items in the queried categories (required for pagination)

msg: If categories are not found based on an ID, the string "noCategory" is returned, which is used by the controller in order to show a message about this.

With the aid of the parameter "categories", all categories to be shown can be specified. A string is passed to the parameter in the "cat_1cat_2_cat_4" format,

for example, in order to display the categories with the IDs 1, 2 and 4. If the string contains "all", all categories are returned.

The parameters "max" and "offset" are required to be able to use pagination.

3.3.1.4.1.2 filterForCategory

This method returns all news of a given category. It is called by the filter method for each individual category.

3.3.1.4.2 RenderService

This service provides a method for rendering HTML.

3.3.1.4.2.1 renderHtml

You pass the URL to the method. An HTTP request is carried out, which fetches the HTML snippet. The correctly formatted HTML snippet is then returned.

3.3.1.5 RenderTagLib

This TagLib provides 3 tags for rendering the header, the footer and the left navigation column. These tags are used in the main.gsp.

3.3.1.6 Starting the sample application

The application is started via the command line:

```
grails run-app
```

3.3.1.7 Overview page as a Grails app

As soon as the application has been successfully started, it is possible to query the news overview page using the following URL:

http://localhost:8080/newsDrilldown/

The links to the news articles on the dynamic overview page refer to the statically generated news detail pages. In this way, a high level of dynamics can be achieved on the website without compromising performance.

3.3.2 FirstSpirit development

The news scenario is integrated in this tutorial in the standard Mithras Energy project. Therefore, import this first and carry out all the subsequent changes in this project.

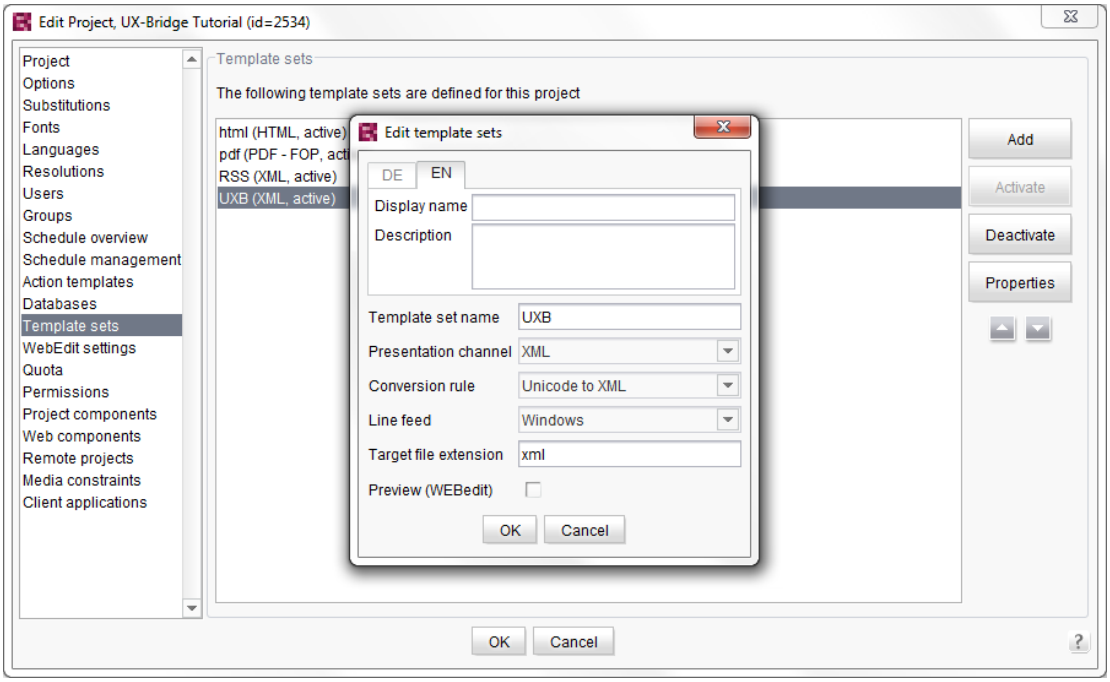
The completely finished FirstSpirit sample project is delivered under the name "uxbridge_tutorial_newsDrilldown.tar.gz" and can be used to view the template code and the settings.

3.3.2.1 Server configuration

In the first step, a new conversion- rule is stored in the server properties. The corresponding rule is to be stored beforehand as a text file.

3.3.2.2 Project configuration

In the project configuration, a new template set for UX-Bridge is to be created, which is to be configured as follows.



In order to send messages to the UX-Bus, the corresponding template, which is to generate the messages, contains the fields that were defined in the data model and are to be output in XML format (compare to Section 2.1.3, "Creating and filling a presentation channel").

3.3.2.3 Section templates

First, create four new section templates with the names "navigation_header", "navigation_footer", "navigation_left" and "navigation_css" and then fill the HTML output channel with the necessary HTML and CSS fragments of Mithras Energy Navigation. These are then output separately and installed in the web app.

In the sample project, you will find this in the "Header/Footer" folder in the section templates.

3.3.2.4 Page templates

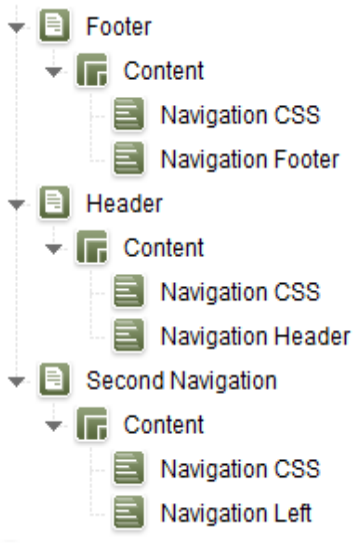
Create a new page template and insert your previously created section templates for the allowed content areas in the Properties tab. Finally, edit your HTML presentation channel as follows:

```
$CMS_VALUE(#global.page.body("content"))$
```

Make sure that you do not use a basic HTML framework in your page template!

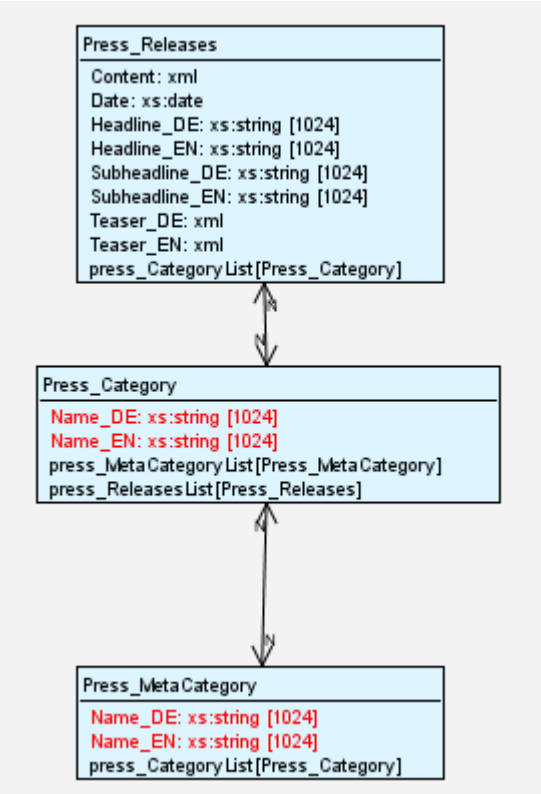
3.3.2.5 Creating pages

Now, based on the previously created page template, create three new pages in your content store and insert the following section templates.



3.3.2.6 Table and table template (XML)

In the schema, you now have to define the data structure for the news, the categories and the meta categories, if they are not already available.



In the "Press_Releases" table, the general content of the press release is defined. Among other things, a header, the text, and the date are included. Using an n:m relationship, the table "Press_Category" is referenced in which the name of a category can be saved. Using an additional m:n relationship, multiple meta categories can be added to a category.

Based on the news table, a table template is to be created according to the following schema which generates the XML that is forwarded to the UXB service.

```
<uxb_entity
          uuid = String
          destinations = String
          language = String
          command = String
          objectType = String
>
```

```
    <uxb_content>
          <fs_id/>
          <language/>
          <url/>
          <date/>
          <headline/>
          <subheadline/>
          <teaser/>
          <content/>
          <metaCategories>
                <metaCategory>
                      <fs_id/>
                      <name/>
                      <categories>
                            <category>
                                  <fs_id/>
                                  <name/>
                            </category>
                      </categories>
                </metaCategory>
          </metaCategories>
    </uxb_content>
</uxb_entity>
```

The child elements in the uxb_content tag function simultaneously as the content fields, which are written by the adapter to the connected content repository, and therefore should also be taken into account during creation of the data structure.

Also create two table templates in addition to the news table for the category and the meta category, and then fill these in in your content sources. In the sample project, you will find them in the schema "Products" with the reference names "Products.press_category" and "Products.press_metacategory".

3.3.2.7 Deployment

In the sample project, it is possible to start generation of the JMS messages, and therefore also of the entries in the connected content repositories automatically via a workflow, directly from the content source. It is also possible to delete objects in FirstSpirit and in the connected content repositories directly from the content sources using an additional workflow. To do so, in addition to scripts, the workflows use table queries and schedules, which must first be configured.

3.3.2.7.1 Create table queries

Table queries have to be created for the generation of a data record and all data records for the News table. The query for a data record therefore has to receive a limitation on the "fs_id" column with the "ID" parameter to be created.

3.3.2.7.2 Creating a schedule

A new schedule has to be created which, alongside the generation of JMS messages for the UXB service, also takes over the generation and deployment of overview pages. In addition, a generation action, which generates the overview pages, must first be added to the schedule. The Delta deployment expands on this action during runtime by adding the detail page of the data record currently to be generated. A script action must then occur which activates UX-Bridge:

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

A partial generation should then take place in the subsequent generation to be created. Page and data record are later entered automatically through the Delta deployment so that only the desired JMS message is generated. The web pages can then be deployed as usual.

If the processing time is measured for the messages in the bus until deployment on the website, then in the final step the action "UX-Bridge Statistics Report" has to be added, which contains the following script:

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

The service waits the maximum amount of time defined in the UX-Bridge service configuration until the adapters' responses are evaluated. If there is no response within this time frame, then the message is classified as having a delivery error. Since response times can vary depending on message and system, this value can be configured.

3.3.2.7.3 Importing workflow scripts

In the next step, the required workflow scripts must be imported:

- uxb_news_example_release_init
- uxb_news_example_release_script
- uxb_news_example_delete_init
- uxb_news_example_delete_script

In this case, the Init scripts initialize variables and write them to the session to ensure that they can be accessed by the UXB module methods, which are queried in the other scripts.

The following parameters have to be configured in uxb_news_example_release_init:

Parameter	Example value	Description
detail_page	pressreleasesdetails	Page reference of the page which generates the JMS messages
query_uid	Products.pressdetailsfilter	Table query which generates all the data records
single_query_uid	Products.pressdetailfilter	Table query which contains the ID of the data record that is to be generated as a parameter
query_param	Id	Parameter name of the table query
schedule_name	UX-Bridge	Name of the schedule that is to generate the JMS messages

scheduler_uxb-generate	UX-Bridge Generate	Name of the generation action for the JMS messages
scheduler_generate	Generate	Name of the generation action for the HTML pages
transition_name	Release	Name of the transition in the workflow (see "Workflow") which is to be switched to after the content_release_script

The script "uxb_news_example_release_script" then starts the previously configured schedule and executes the defined transition.

The following parameters are configured in uxb_news_example_delete_init:

Parameter	**Example value**	**Description**
destinations	postgres	Name of the content repositories from which the data record is to be deleted
transition_name	release	Name of the transition in the workflow (see "Workflow") which is to be switched to after the content_delete_script
object_type	news	Type of object that is to be deleted

Within the following script "uxb_news_example_delete_script", the selected data record is deleted in FirstSpirit and a message is sent via the UXB service and the bus to the attached content repository, which triggers the delete action there.

3.3.2.7.4 Importing workflows

The workflows "uxb_news_example_release" and "uxb_news_example_delete" query the previously configured scripts.

3.3.2.7.5 Complete alignment process

In the FirstSpirit sample project, the complete alignment process is implemented in the "UX-Bridge Full Deployment" schedule.

Information on the procedure for the complete alignment process are found in Section 2.1.4.2, "Complete alignment process", page 14.

3.3.3 Adapters

The adapter is the component which reads in the data from the UX-Bus and writes them to the live repository.

3.3.3.1 JAXB – XML processing

In this example, JAXB is used for the processing of the XML defined in the presentation channel. The corresponding classes are located in the **com.espirit.moddev.examples.uxbridge.newsdrilldown.entity** package.

Like JPA, in JAXB, work is done to bind the XML tags to a Java object with annotations.

```
@XmlRootElement(name = "uxb_entity")
@XmlAccessorType(XmlAccessType.FIELD)
Public class UXBEntity {
     @XmlAttribute
     private String uuid;
     @XmlAttribute
     private String language;
     @XmlAttribute
     private String destinations;
     @XmlElement(type = UXBContent.class)
     private UXBContent uxb_content;
```

```
    @XmlAttribute
    private String command;
    @XmlAttribute
    private String createTime;
    @XmlAttribute
    private String finishTime;
```

3.3.3.1.1 DateType: XmlAdapter for the date format

Dates are input in the FirstSpirit presentation channel according to the format "yyyy-MM-dd'T'HH:mm:ssZ". The DateAdapter class has been implemented so that this format can be read into the JAXB classes. This class is in the **com.espirit.moddev.examples.uxbridge.newsdrilldown.entity.type** package.

```
@XmlElement()
@XmlJavaTypeAdapter(value = DateAdapter.class, type = Date.class)
Private Date date
```

3.3.3.1.2 UXBEntity, UXBContent, UXBMetaCategory and UXBCategory

These classes represent the exchange format defined in the presentation channel.

3.3.3.1.3 UXBEntity

This class corresponds to the basic framework of the exchange format prescribed by UX-Bridge.

3.3.3.1.4 UXBContent, UXBMetaCategory and UXBCategory

The project-specific JAXB classes for processing the exchange format. This contains the actual information of the objects that are distributed via UX-Bridge.

This example therefore shows the press releases with the corresponding meta categories and categories.

3.3.3.2 Hibernate domain classes

The domain classes are located in the **com.espirit.moddev.uxbridge** package.

In order to map multiple languages, every object contains a language, and every

language is saved as an independent object in the repository.

As a result, the FirstSpirit ID (UUID) is no longer unique. Standard Hibernate/JPA mechanisms are therefore used to generate a unique ID.

```
@Id
@GeneratedValue(strategy = GenerationType.AUTO)
private Long id;
```

This ID in the live repository changes after deleting it from the repository and adding it again. If your application situation requires a different behavior, then you can use a compiled primary key comprising the FirstSpirit ID and the language.

3.3.3.2.1 News, NewsCategory, NewsMetaCategory

The structure of the classes corresponds to the structure in the database schema defined in FirstSpirit. This procedure is not absolutely necessary, but is being described here to provide a better understanding of the concept.

3.3.3.3 NewsHandler

The NewsHandler in the **com.espirit.moddev.exsamples.uxbridge.news.jpa** package is the class which takes the data and processes it. In the example, a unique method was implemented in the handler for each of the supported commands.

3.3.3.3.1 Command: add

Saving or updating a press release in the live repository.

In this situation, it must be ensured that the meta categories and categories are transferred in the exchange format within the press release. In the repository, however, categories and meta categories are saved separately.

For this example, this means that, in addition to the press release, the included categories and meta categories have to be newly created or updated with this command.

3.3.3.3.2 Command: delete

Deletes a press release in the live repository and the associated detail page defined in the schedule script (see Creating a schedule) on the web server. In order for the methods to be able to find the correct page on the web server, the "webpath"

parameter on the path to the web server directory (for example, ''(/home/tomcat/webapps") must be set in the applicationContext.xml in the news handler bean.

```
<constructor-arg name="webpath" value="/home/tomcat/webapps"/>
```

Note that the implementation in this example does not provide for the deletion of meta categories or categories if there is no press release in one of these categories.

3.3.3.3.3 Command: cleanup

Deletes all press releases which are older than the date indicated.

3.3.3.4 Routing

Components are configured in the Spring XML file: **WEB-INF/applicationContext.xml**. This means that the database connection, ConnectionPooling, JMS and the routing are defined.

The routing is defined in the XML area <camelContext id="camelContext" ...>.

```
<camelContext  id="camelContext" trace="false"
xmlns="http://camel.apache.org/schema/spring">
<package>com.espirit.moddev.examples.uxbridge.newsdrilldown.entity
</package>
<onException>
<exception>java.io.IOException</exception>
<handled>
<constant>true</constant>
</handled>
<to
uri="adapterReturn:jms:topic:BUS_IN?destination=postgres&amp;bodyV
alue=bodyTemp" />
</onException>
<route id="uxbridge-commands" >
<from uri="activemq:Consumer.newsDrillDown-
Hibernate.VirtualTopic.BUS_OUT" />
<filter>
<xpath>//uxb_entity[contains(@destinations, 'postgres')]</xpath>
<filter>
<xpath>//uxb_entity[@objectType = 'news_article']</xpath>
```

```
<camel:setHeader headerName="bodyTemp">
<simple>${body}</simple>
</camel:setHeader>
<filter>
<xpath>//uxb_entity[@command = 'add']</xpath>
<convertBodyTo
type="com.espirit.moddev.examples.uxbridge.newsdrilldown.entity.UX
BEntity" />
<bean ref="newsHandler" method="add" />
</filter>
<filter>
<xpath>//uxb_entity[@command = 'delete']</xpath>
<convertBodyTo
type="com.espirit.moddev.examples.uxbridge.newsdrilldown.entity.UX
BEntity" />
<bean ref="newsHandler" method="delete" />
</filter>
<filter>
<xpath>//uxb_entity[@command = 'cleanup']</xpath>
<convertBodyTo
type="com.espirit.moddev.examples.uxbridge.newsdrilldown.entity.UX
BEntity" />
<bean ref="newsHandler" method="cleanup" />
</filter>
<to uri="adapterReturn:jms:topic:BUS_IN?destination=postgres" />
</filter>
</filter>
</route>
</camelContext>
```

Additional information and options can be found under http://camel.apache.org/spring.html.

A detailed explanation of how to create the response can be found in Section 3.5, "Using the Camel component to generate a response".

3.3.3.4.1 The route uxbridge-commands

Any number of routes can be defined; the routes defined in the adapter are not to be confused with the routes of the UX-Bus and do not take over their tasks. The routes in the adapter should only contain the routes important for this adapter.

In this application example, a route was defined: <route id="uxbridge-commands">

3.3.3.4.2 Message source

The From tag specifies the integration framework (Apache Camel) from which the data is read. In this example, the From tag appears as follows:

```
<from uri="activemq:Consumer.newsDrillDown-
Hibernate.VirtualTopic.BUS_OUT" />
```

The data or messages are read out via the JMS Topic "BUS_OUT". At this location, a virtual end point is used (see http://activemq.apache.org/virtual-destinations.html). The advantage of virtual end points is that no modifications have to be carried out on the routing for additional adapters. New, virtual end points only have to follow the "Consumer.%any adapter name%.VirtualTopic.%source termination point%" naming schema. Through the virtualization, messages are not read as in a queue by only one adapter; rather, all corresponding adapters receive the message.

If, for example, the new adapter "myAdapter" is also to consume messages that are delivered at the end point FS_OUT, then a possible end point might look like the following:

activemq:Consumer.myAdapter.VirtualTopic.BUS_OUT

3.3.3.4.3 Filters

By filtering messages, the NewsHandler filters out messages which are unimportant to it, i.e. messages for a different repository or from a different type of object.

To filter the messages, in this example, we just use XPath expressions.

In this functionally limited example, not all filter options are necessary, but have been included to help inspire new ideas.

3.3.3.4.3.1 Destination filter

```
//uxb_entity[contains(@destinations, 'postgres')]
```

Here, messages are filtered which are to land in the PostgreSQL database. All other messages that do not fit this expression are then ignored. The messages can be simultaneously written to different live repositories, and are processed in this location with 'contains'.

3.3.3.4.3.2 Object type filter

```
//uxb_entity[@objectType = 'news_article']
```

The NewsHandler can only process objects of the type "news_article". Messages that do not apply to this expression are then ignored in this case as well. Usually, messages always contain only one object of a type; therefore, this is processed here with "=".

3.3.3.4.3.3 Command filter

```
<xpath>//uxb_entity[@command = 'add']</xpath>
```

In the last step, the messages are filtered according to commands.

3.3.3.4.3.4 JAXB conversion

```
<convertBodyTo
type="com.espirit.moddev.examples.uxbridge.news.entity.UXBEntity"
/>
```

The XML of the message is converted to a Java class via a JAXB.

3.3.3.4.3.5 Method query

```
<bean ref="newsHandler" method="add" />
```

At the very end of the filter chain, the corresponding method is queried in the NewsHandler.

3.3.3.5 Starting the sample adapters

The API can be loaded into the local Maven repository using the following call:

```
mvn install:install-file -Dfile=<path-to-file> -DgroupId=<group-
id> -DartifactId=<artifact-id> -Dversion=<version> -
Dpackaging=<packaging>
```

An implementation example might look like the following:

```
mvn install:install-file -Dfile= D:\uxbridge-module-api-
1.2.4.1133.jar -DgroupId=com.espirit.moddev.uxbridge -
DartifactId=uxbridge-module-api -Dversion=1.2.4.1133 -
Dpackaging=jar
```

The sample adapters can be established via the command line:

```
mvn package
```

The War file resulting from this can be deployed on any servlet container (Tomcat, Jetty etc.).

Alternatively, the adapters can also be started via the command line:

```
mvn tomcat7:run
```

In order to adapt the port of the Tomcat which was started by this, the file pom.xml has to be adapted in the directory of the respective adapter.

3.3.3.6 Tests included

The sample project includes unit and integration tests. For the tests, an In-Memory database and jMockMongo (https://github.com/thiloplanz/jmockmongo) are used. The jMockMongo Jar has to be imported into the local repository so that the tests for the MongoDB adapter can be started.

The integration tests can be started with the following call: **mvn verify -Pintegration-test**

3.4 Using the UXB service API

To use UXBService in a module or script, the API jar file of an equivalent version has to be added to the class path.

You then receive access to UXBService using the following call:

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

You can find an implementation example for delete and release executables in the Github repository under uxbridge-api-example.

3.4.1 Creating a demo project

Apache Maven is required to create a demo project. In addition, fs-access.jar has to be installed as an artifact in the local Maven repository.

The project can be built using **"mvn clean package"** once these prerequisites have been met. In this step, an FSM is created in the project's target directory; it can be installed using the FirstSpirit Server admin console. It will then be possible to use the included sample executables.

3.4.2 Use

Both sample implementations can be used in both of the previous tutorials. This requires that you proceed as follows:

3.4.2.1 "Delete data record" script (uxb_content_delete_script)

The script has been implemented as an executable; therefore, just the executable is called at this point.

```
#! executable-class
com.espirit.moddev.uxbridge.samples.workflow.DeleteEntityExecutable
```

3.4.2.2 "Release data record" script (uxb_content_release_script)

This script has also been replaced by the corresponding executable.

```
#! executable-class
com.espirit.moddev.uxbridge.samples.workflow.ReleaseAndDeployEntityExecutable
```

3.4.2.3 CamelContext return

FirstSpirit expects feedback in the form of an XML document after interacting with an adapter using UX-Bridge. There is a Camel component available that creates this XML document.

To access the component in CamelContext, the component has to be integrated as follows:

```
<bean id="adapterReturn"
class="org.uxbridge.camel.component.AdapterReturnComponent">
</bean>
```

In order to use the function, it also has to be called when forwarding data to BUS_IN.

```
<to uri="adapterReturn:jms:topic:BUS_IN" />
```

This happens regardless of whether the data is transmitted to the database successfully or there is an exception. The function creates the appropriate response in either case.

3.5 Using the Camel component to generate a response

This component can be used to generate the response that FirstSpirit expects from an adapter. This is true for both a regular response and for a response in the event of an error. Using this component requires that the adapter is implemented using Apache Camel. You can find more information on Apache Camel on the Apache Camel website (http://camel.apache.org/).

3.5.1 Integrating the component

A Camel component has to be integrated in order for you to receive access to it. This is done using the provided uxbridge-camel-component-<version>.jar file. This has to be integrated into the project's Java class path. (For Eclipse: right-click on the project->Java Build Path->Libraries->Add external JARs)

3.5.2 Integrating the component as a bean

The component has to be integrated as a bean in order for the component to be used within an adapter. The call for this appears as follows: <bean id="adapterReturn"
class="org.uxbridge.camel.component.AdapterReturnComponent"></bean>

You can choose any ID. However, changing the ID will require the URL structure shown in the following subsection to be adapted accordingly.

3.5.3 Structure of the URL

The structure of the URL starts with the call for the component. This is done using the ID specified when integrating the component. This is followed by the call for the destination. The destination parameter is also appended to the end of the URL.

The complete structure might then look like the following:

<to uri="adapterReturn:jms:topic:BUS_IN?destination=mongodb" />

A second parameter is required when calling within exception handling; this has to be supplemented by attaching the URL structure:

<to
uri="adapterReturn:jms:topic:BUS_IN?destination=mongodb&bodyValue=body
Temp" />

3.5.4 Parameters

Two parameters are passed to the component.

The first parameter is the destination the response is generated from. This parameter is appended directly to the URL using a question mark. Since multiple destinations can be passed within the call for the adapter, this parameter is used to differentiate the destination that led to this response.

The second parameter is only required in the event of an exception. Since the current status of the message is passed to the exception handler in the event of an exception, there may be instances where the message content is no longer complete and part of it, such as the XML document root element, is missing. However, since processing requires the entire XML document, the document has to be stored temporarily before being processed in the message's header. The content can be stored temporarily using <setHeader headerName="bodyTemp"><simple>${body}
</simple></setHeader>. You can choose any headerName in the process, but it has to be shared with the component. This is done using the second bodyValue parameter.

4 Expansion Options

4.1 Creating your own messages from FirstSpirit

Expansion makes it possible to send your own messages in any format you define to the bus via the UXB service. Implementation of the UxbMessageGenerators interface is required. The interface already includes the schedule context as well as some methods that provide information to the generated element. The context is used to access the entire project and its elements.

4.1.1 UxbMessageGenerator interface

To implement your own UxbMessageGenerator, the UX-Bridge API jar file of the equivalent version has to be added to the class path (see Section **Fehler! Verweisquelle konnte nicht gefunden werden.**, **Fehler! Verweisquelle konnte nicht gefunden werden.**). In addition, it has a dependency on the fs-access.jar. This is included in any FirstSpirit Server installation and therefore also needs to be made available in the class path.

The new class first implements the UxbMessageGenerator interface:

```
public class DemoMessage implements UxbMessageGenerator
```

In addition, there are some required methods in this class that are used by the UXB service in order to pass information. These methods are briefly described in the following:

setData(byte[ ] data)

The data object contains the rendered content of the presentation channel.

setCreateTime(long createTime)

createTime is the time at which the workflow was started.

setStartTime(long startTime)

startTime is the time at which the message was generated.

setSchedulerId(long schedulerId)

schedulerId is the ID of the schedule that was started.

setProjectId(long projectId)

The ID of the generated project.

setGenerationContext(GenerationContext generationContext)

The entire generation context for the schedule is contained in generationContext. It is used to access the entire project and its elements as well as the currently generated element.

generate()

The generate method is called by the UXB service to generate a message and must return a 'Document' type generated message.

4.1.2 Calling your own UxbMessageGenerator

Another script action within the schedule that generates the UXB messages must be carried out before the "Activate Generation" script action (see 2.1.4.1, Partial generation)

The script must contain the 'MessageGenerator' context variable with the fully qualified name of the class that will handle message generation.

```
context.setProperty("MessageGenerator","com.package.DemoMessage");
```

For this purpose, the class must be made available within FirstSpirit Server, e.g. via a module. To ensure that classes are loaded properly, the class can be configured as a public component with local module resources.

4.1.2.1 Calling within a cluster operation

Since no scripts are started on the slave systems in cluster mode, the name of the class that will handle message generation is not passed using a script action, but rather in the template of the project settings page. Additional information needs to be added to following lines:

```
$CMS_SET(#global.pageContext["MessageGenerator"],
"com.espirit.moddev.portal.UxbPortalMessage")$
```

In addition, it is important to make sure that "Generate Media in the generation directory" is not selected in the generation action of the schedule.

5 Appendix

5.1 Conversion rules for Unicode to XML

```
[convert]
0x00=""
0x01=""
0x02=""
0x03=""
0x04=""
0x05=""
0x06=""
0x07=""
0x08=""
0x09=""
0x0A=""
0x0B=""
0x0C=""
0x0D=""
0x0E=""
0x0F=""
0x10=""
0x11=""
0x12=""
0x13=""
0x14=""
0x15=""
0x17=""
0x18=""
0x19=""
0x1A=""
0x1B=""
0x1C=""
0x1D=""
0x1E=""
0x1F=""
0x3c="&lt;"
```

```
0x3e="&gt;"
0x22="&quot;"
0x27="&#039;"
0x26="&amp;"
[quote]
```