FirstSpirit™
Unlock Your Content

UX-Bridge Developer Documentation

Version	**1.1**
State	**RELEASED**
Date	**2013-03-07**
Department	Produktmanagement
Author/ Authors	C. Feddersen
Copyright	2012 e-Spirit AG
File name	UX-Bridge_DeveloperDocumentation_EN

Inhaltsverzeichnis

Introduction

The "UX-Bridge" module is a response to the trend of dynamic websites. Whenever content cannot be pre-generated, 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 the hybrid architecture approach by adding a standard component for dynamic content delivery. Additional information can be found in the white paper, Chapter 1.3.

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

In the first chapter, it describes in general the steps which are to be taken into account while 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 2 explains the use of UX-Bridge based on two tutorials. Here, we will go through 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 answered when defining the concept.

During definition of the 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 an already existing project, then 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 implement 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 Chapter "Create and configure schedule".

During generation, a message is sent to the UX-Bus for each 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. For instance, if only the news (maintained using a content source) in a project is to be delivered via UX-Bridge, then only the necessary content projection pages should 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 be used for this. However, "stopGenerate" should then be used in order to cancel the generation of page references in the UX-Bridge presentation channel, which are not to generate any messages (see Online Documentation for FirstSpirit: Vorlagenentwicklung\Vorlagensyntax\Systemobjekte\#global\vorschaubezogen\Abbruch einer Vorschau/Generierung).

In other projects, a large proportion of the changes are released via workflows, which subsequently carry out a generation and deployment of the participating objects. In this, mostly the changed objects are identified by script, which then are defined as start nodes of a partial generation. Objects are deleted using delete workflows. UX-Bridge can be integrated into these scenarios also without problems (see Chapter "Workflow coupling").

If the time in which the objects have to be available on the website plays a large role, 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 more quickly the objects are available on the website. With FirstSpirit5, a simplified mechanism is available for such scenarios with DeltaGeneration (Developer API\Delta Generation, Access-API\Generate Task). In FirstSpirit4, this can already be achieved using the revision API.

Altogether, UX-Bridge can be incorporated into the existing generation and deployment concept or into one to be newly created, and for this purpose, brings no independent (separate) solution with it.

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 contents through the database schema; with weakly structured contents 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 in which step the transformation from one data model to another is to be carried out. The answer is certainly project- specific, so that here, only the possible variants are described. An evaluation has to be done in the context of the project:

a) The transformation is done via the CMS syntax in the templates, which generate the XML necessary for UX-Bridge (see Chapter "Data exchange format – FS Templating vs. WebApp development"). The adapter itself does not carry out any large transformations, but rather writes the objects in such a manner in the content repository as was 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 it is easier to be implemented.

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 keep the data exchange format generalized and carry out any necessary/logical transformations for writing to the content repository within the adapter.

2) If the data are 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 refer to different types of contents. 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) In general, it is to be considered whether, for decoupling and maintenance reasons, it would be better to use multiple (but lean) adapters or one adapter that contains all the logic.

5) A general data exchange format has the advantage that only one message has to be sent via 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 which 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 chapter, for every page reference within the generation, a message is generated and sent to the UX-Bus. A 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 the chapter on "Routing".

This configuration can be adapted for the project-specific requirements. In this, with the Apache Camel Spring XML Syntax, there is a simple domain-specific language (DSL) available, 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 this, an adapter can, for example, direct a web application to empty its cache because new data was written to the content repository.

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

In the standard configuration of UX-Bridge, a routing 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

In this chapter, the steps for implementing UX-Bridge in a project are described. The Quick Walkthrough is intended for experienced FirstSpirit template developer and web application developers. A more in-depth, step-by-step introduction can be found in the "Tutorials" chapter.

2.1 FirstSpirit

2.1.1 Installation

The starting point for development in the context of UX-Bridge is the installation of the components.

To do so, first, the provided module should be installed in the server settings of the FirstSpirit server. Through this, the UX-Bridge service is started, and the UX-Bridge components are available in the projects of the server (compare to UX-Bridge installation handbook: "Installation of the FirstSpirit Module").

Alongside the FirstSpirit module, the installation of the UX-Bus is also necessary. For local development, the installation in standalone operation is recommended (refer to UX-Bridge installation handbook: "Standalone Operation").

2.1.2 Data exchange format – FS Templating vs. WebApp development

The architecture of UX-Bridge allows the development of solutions on the basis of 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 done by different people, then during conceptual design, a common data exchange format should be defined. With this, the data format is intended that is generated by the templates and sent to the adapter via the UX-Bus as a message.

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 involves 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 rest 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:

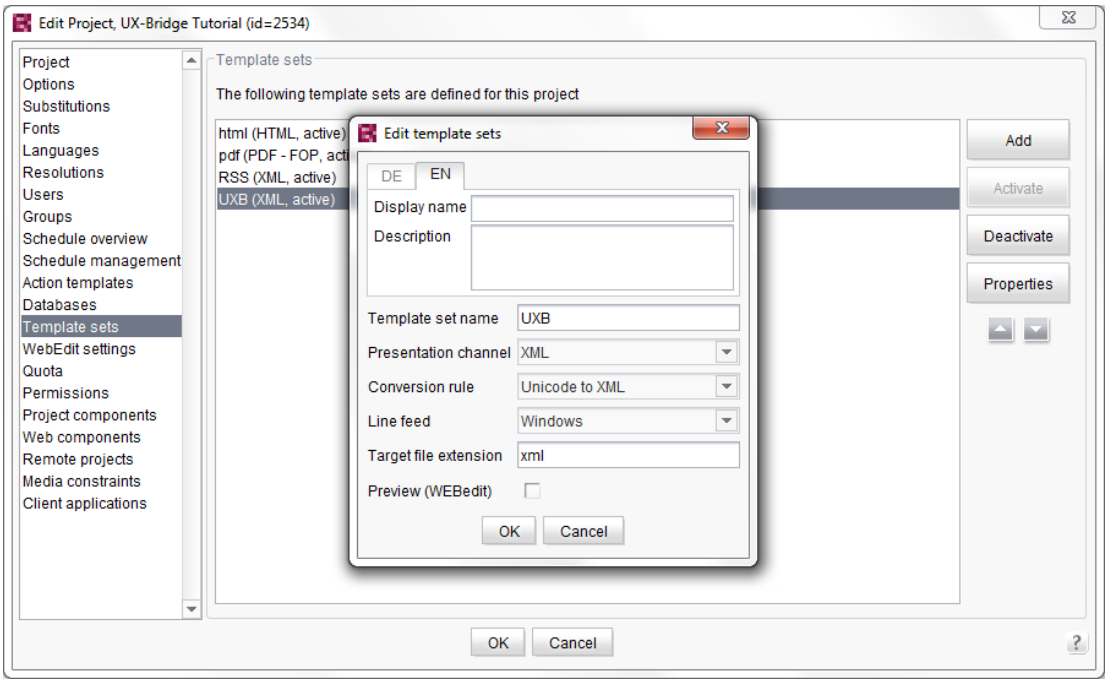
- Do the data, and thus the data model, already exist in FirstSpirit? If yes, then you are subject to certain limitations. If no, it is advisable to match the data exchange format as closely as possible to the content repository or the web application.

- For performance reasons, it can 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.

- Are 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 can also be more efficient to generate two data exchange formats through FirstSpirit, which then can be taken over without a transformation step in the respective content repository.

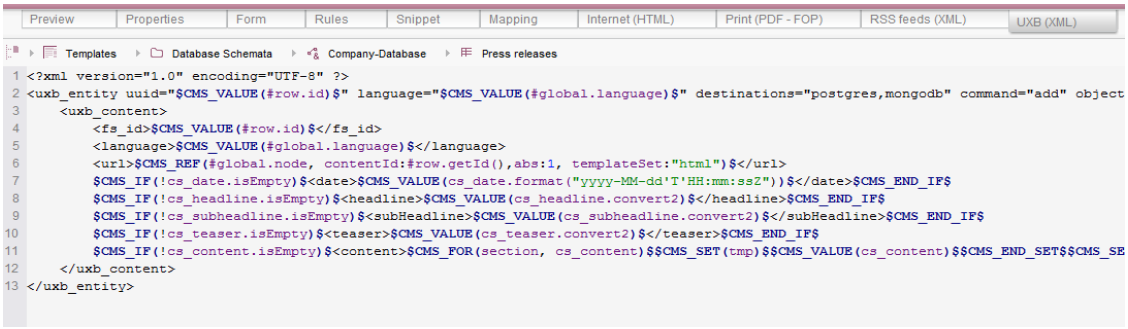
2.1.3 Creating and filling a presentation channel

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



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

In order to send messages to the UX-Bus, which is to generate the messages, the fields which were defined in the data exchange format are to be output in XML form in the corresponding template.



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

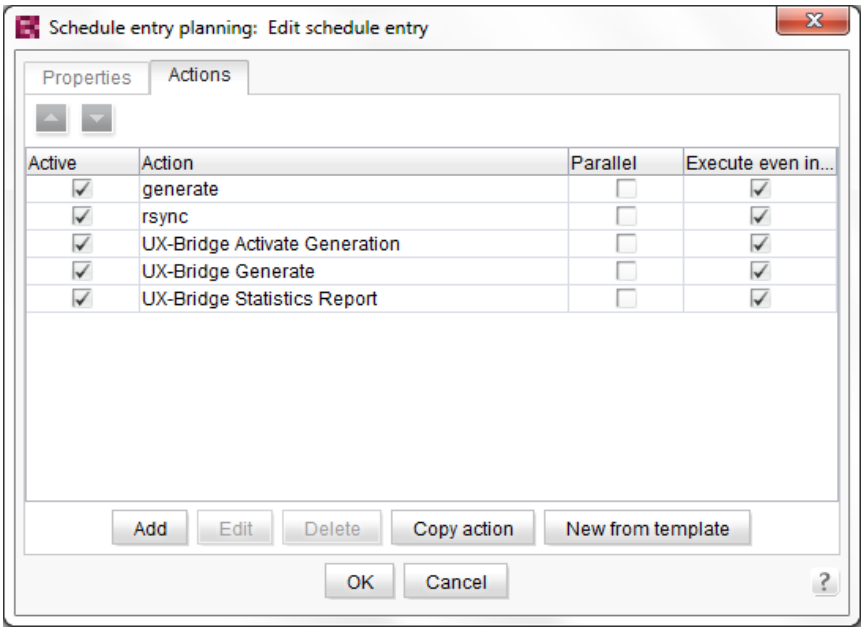
In order to convert the data from FirstSpirit into messages that can be further processed by UX-Bus, it is mandatory for a schedule to be created or an existing schedule extended so that it generates XML. This is then forwarded to the UXB service, which generates a message from it and sends this to the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

In this chapter, two schedules are now to be explained, which are probably 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 one in such a manner that it carries out the steps described in the following; and, in doing so, only the new content is generated and published.

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



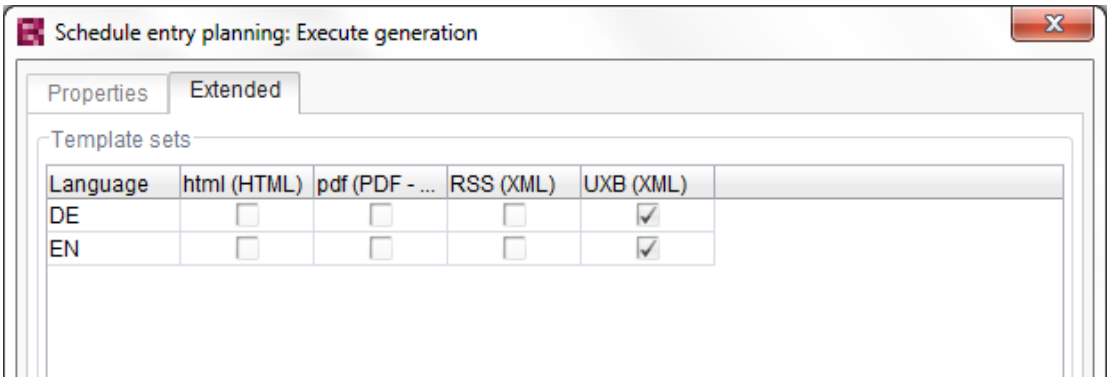
First, all of the complete static pages that are needed for display on the website are to be generated (e.g. news overview pages) in a generation action. This generation action takes place in the deployment schedules commonly used to date, and does not have to be adapted.

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.

Afterward, in order to use UX-Bridge, a new, additional action is to be added which, by means of querying the script, activates the generation for UX-Bridge:

```
#! executable-class

com.espirit.moddev.uxbridge.inline.UxbInlineUtil
```

In the following generation action, the exact page should be generated (e.g., the news detail page) that the XML creates, which is to be forwarded to the UXB Service. It is important to ensure that the UXB template set was enabled in the advanced properties.



The 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 can transfer the generated page to the schedule.

The last action, "UX-Bridge Statistics Report", is optional, and enables the cycle times for the messages in the bus to be measured until publication to 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)
```

To do so, the following script call is necessary:

```
import com.espirit.moddev.uxbridge.service.UxbService;

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

uxbService.waitSeconds(10);

uxbService.getTimings(context.getStartTime().getTime());
```

In the example, the service waits 10 seconds, until the adapters' answers are evaluated. If there is no answer in this time frame, then the message is classified as having a delivery error. Because the response times can vary depending on

message and system variant, this value can be configured (for example, uxbService.waitSeconds(10)).

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 data inventory in the content repository in the most current state. For this, the data deleted in FirstSpirit must likewise be deleted in the external repository.

The following procedure is recommended:

1) Complete generation of the static pages in FirstSpirit.

2) Run UX-Bridge schedule in order to write all data to the ContentRepository (see also the previous section). These data are given an up-to-date time stamp, which is saved in ContentRepository in the "lastmodified" field.

3) Run script that calls up the cleanup method with a timestamp as a parameter that describes the latest project revision of the schedule. The cleanup method then deletes all data that have saved an older time than the one given to them in the "lastmodified" field. 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.service.UxbService;

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

uxbService.removeUxbEntriesByTime(context.getStartTime().getTime()
, "news", "postgres,mongodb");
```

2.1.5 Workflow coupling

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

2.1.5.1 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).

2.1.5.2 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.

The call-up of the UXB service in the script is 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 following example:

```
<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 also serve 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 for 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 was written or should 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

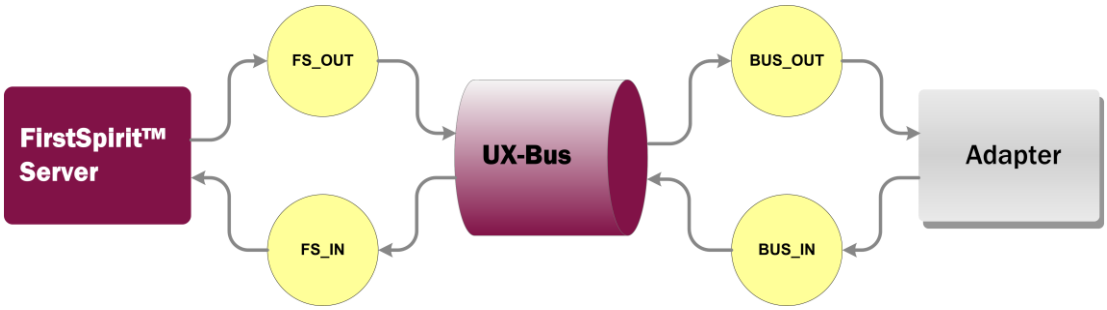
schedulerId	Unique ID for the schedule (appended to the XML document by FirstSpirit during the action)	123456	Yes
command	Command executed by the adapter	delete	No
language	Language of the message	DE (German)	No
createTime	Timestamp for the creation of the action (appended to the XML document by FirstSpirit 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 circumstance 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 is in place.

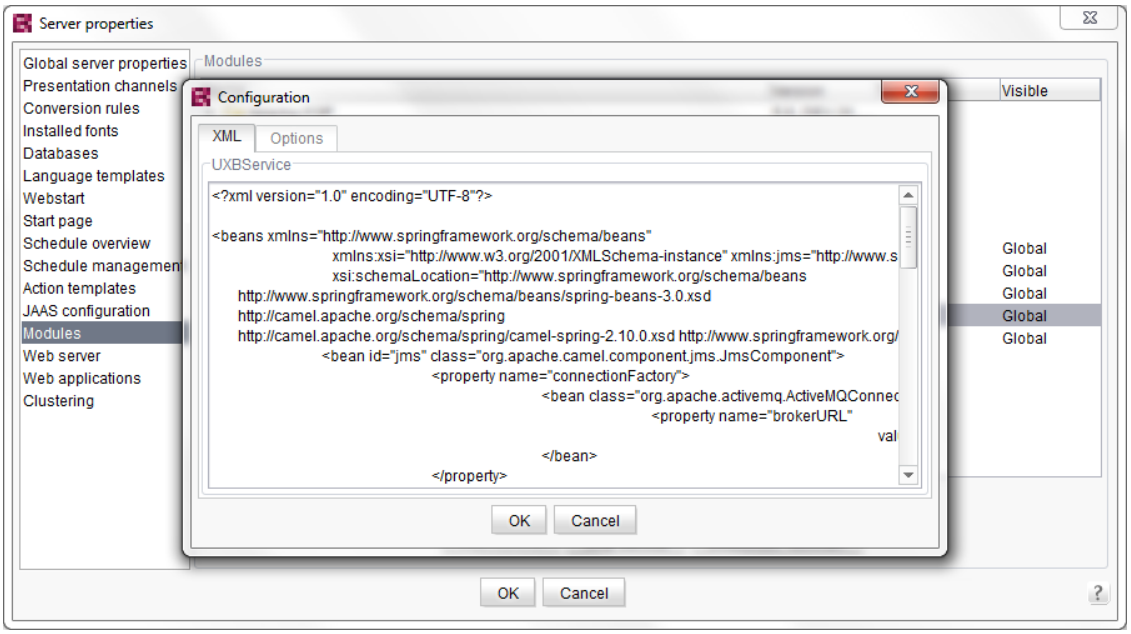
2.4 Routing

UX-Bridge uses Apache Camel to route messages. As a transportation and message protocol, Java Message Service (JMS) is used. Both participating components of the FirstSpirit Server and adapter function in the role of a producer, which generates messages and makes them available in an end point, and in 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 the messages between the participating end points.



2.4.1 End points in FirstSpirit

The configuration of the UXB service can be reached via the FirstSpirit Server configuration and the Module subitem. On the expanded module tree, UXB service has to be selected and opened via the "Configure" button, the configuration of services (Spring DSL) opens, which also contains the end points and a route. The configuration does not usually have to be adapted; however, if the names of the end points configured in the bus are changed, then these also have to be adapted in the configuration in FirstSpirit. You must use the Adapter-Statistics-Response-Route with the UxbServiceStatisticsResponseHandler bean if UX-Bridge is to use its own monitoring (see Installation Handbook: Monitoring in the Schedule).

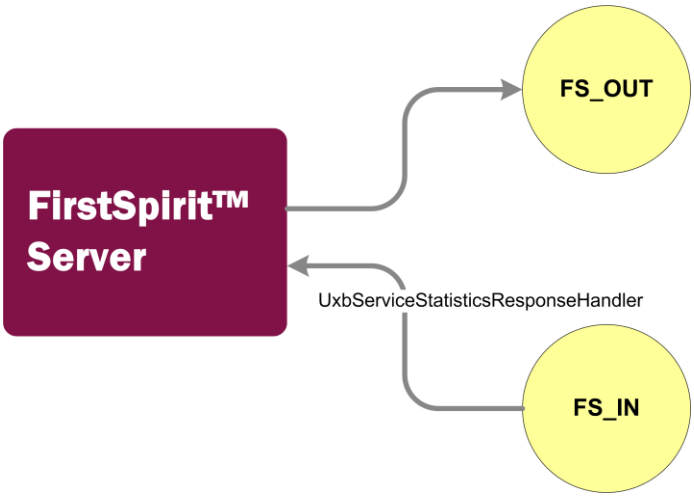


Within Spring DSL, a Camel context is described which contains the routes and end points:

```
<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>

<route id="Adapter-Statistics-Response-Route">

          <from uri="jms:topic:FS_IN"/>

          <convertBodyTo
type="com.espirit.moddev.uxbridge.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>
```

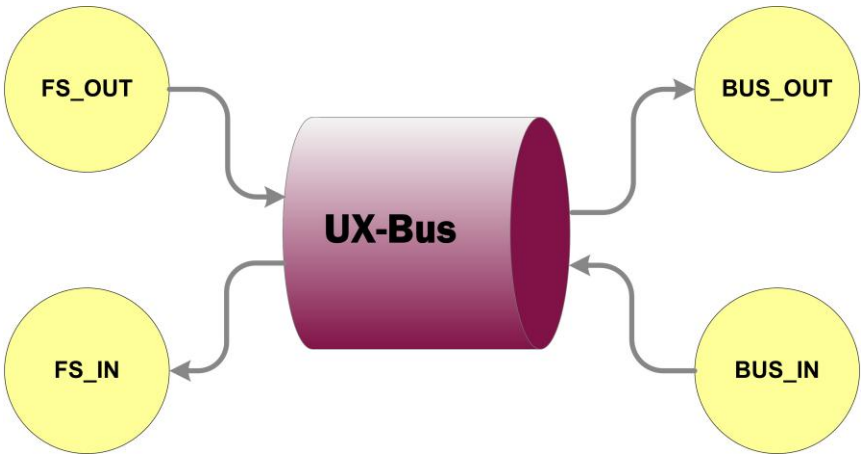
In the example, there is an end point with the ID "'FS_OUT", which serves as an end point for messages which are sent by the service.

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, say, timing purposes.



2.4.2 Routing in the UX-Bus

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

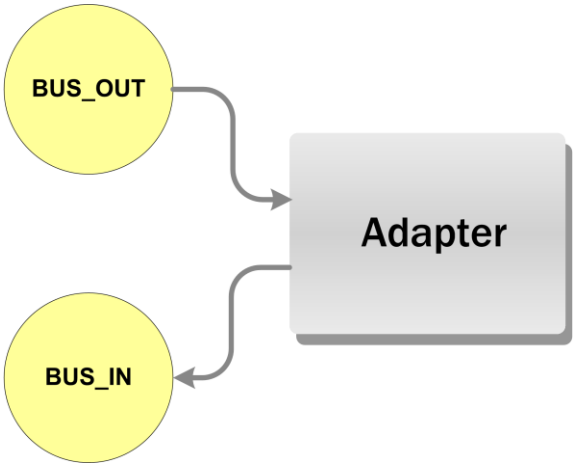
The first route sends messages from the end point "activemq:topic:FS_OUT" in the direction of the adapter to its end point "activemq:topic:VirtualTopic.BUS_OUT". Via XPath, for example, another differentiation for multiple adapters is carried out. "@objecttype" refers here to the JMS message header (see 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 in 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 transferred into your own projects or used as a suggestion 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 is found under: 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

Aside from that, it is required that the UX-Bus is in operation and is accessible. Information for the operation of UX-Bus can be found in the Installation Documentation.

With the applications, it is required to adapt the database configuration to the local conditions. Information on where the respective configurations are can be taken from the respective chapter.

3.1 News widget scenario

In this example, a simple widget is created which shows 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 installed in the page by JavaScript at runtime.

In the example application, 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 administered in FirstSpirit.

The web application was created with the Grails web framework in Version 2.1.0.

3.1.1.1 Configuration

All configuration files are in the typical file for Grails applications <grailswidget>/grails-app/config.

In this area, 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, in other words, 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

Here, 2 mappings were added:

"/rest/v1/articles" indicates the action "list" of the ArticleRestController.

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

3.1.1.2 Domain class

In this application, there is an individual domain class: com.espirit.moddev.examples.uxbridge.widget.Article

Grails, like the adapter, uses the persistence framework Hibernate. 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, for the parameters transferred, no article is found, 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 most up-to-date articles from the live repository

3.1.1.4.2 ellipsis

The method shortens the text after a certain number of characters. This method is used to shorten the text for the widget.

3.1.1.5 SQL and NoSQL

In contrast to the adapters, an adaptation of the source code of the web application 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 example application

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 indicated

```
grails mongo run-app
```

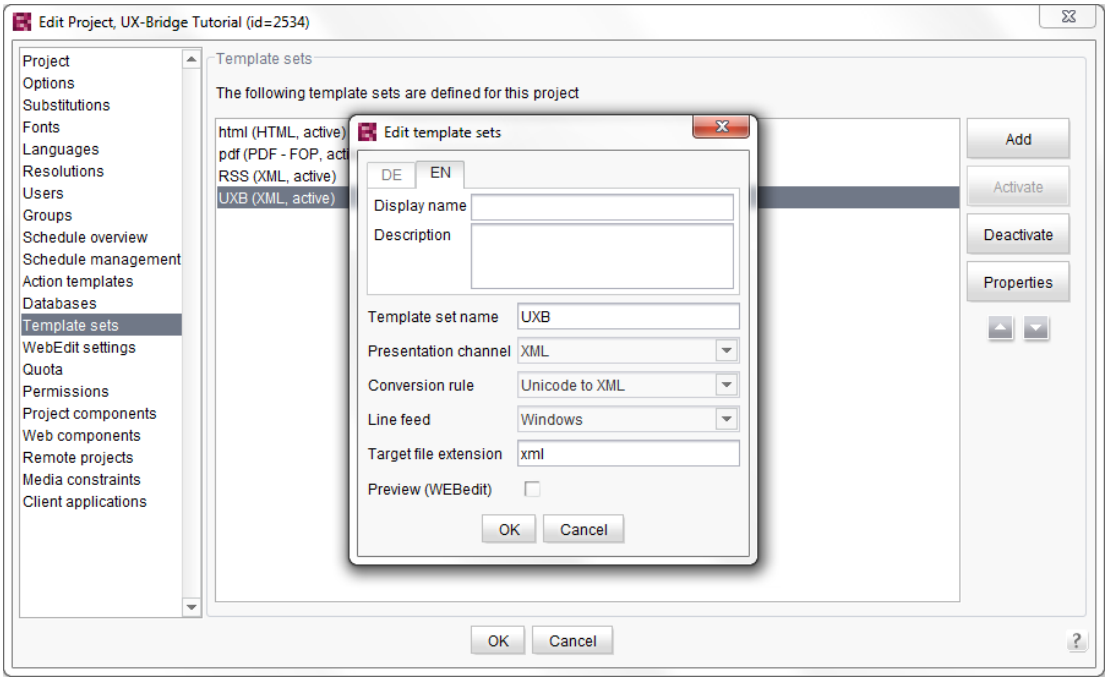
3.1.2 FirstSpirit development

The News widget is installed in this tutorial in the standard project Mithras Energy. That is why you import this first and carry out all the following changes in this project.

The complete, finished example project is delivered at the same time 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

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



In order to send messages to the UX-Bus, in the corresponding template, which is to generate the messages, the fields which were defined in the data model are to be output in the form of XML (compare to 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 later the fitting articles are dynamically reloaded and shown in the widget. For this, 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

The page templates that UX-Bridge is to use are expanded in the example by an input component. This input component has the effect that the marginal column is enlarged so that the widget to be installed has more space available. The reason for that is that the marginal column is too small in its standard width 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 in addition serve to activate UX-Bridge on only the pages that integrate them.

Alongside the expansion of the input form, the code also has to be adapted in the template.

At the beginning of the page template, a variable must be stored in the body area in a variable, so that the variables that are set in the body are available already 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 extended by adding the following call, which initializes the page variables of UX-Bridge:

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

The HTML header must then be extended by adding 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, first a new section template with the name "uxb_widget" is created 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 classes clearfix and teasermodule 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.

In the form suitable for this, you can 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 installed 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_baseURL_UXB)$/rest/v1/articles",speed:2000, fadeFrom: "#F7D358", fadeTo: "white", count: $CMS_VALUE(set_news_count)$});
});
</script>
```

3.1.2.6 Create page

In order to use UX-Bridge, a new template of the type "uxb_widget" is installed 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 table press releases 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 are simultaneously 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.

Here, the code example 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 example project, the generation of JMS messages, and with it, entry into 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 the connected content repositories directly from the data sources using an additional workflow. To do so, the workflows use, alongside the scripts, table queries and schedules, which are first to be configured.

3.1.2.8.1 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 queries for a data record still have to have a limitation on the column "fs_id", with the parameter to be newly created, "Id".

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

3.1.2.8.2 Create 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 must first be added to the schedule, which generates the overview pages. The Delta deployment extends this action during runtime by adding the detail page of the data record currently to be generated. Afterward, a script action has to happen which activates UX-Bridge:

```
#! executable-class

com.espirit.moddev.uxbridge.inline.UxbInlineUtil
```

A partial generation is to take place in the generation to be created thereafter. Page and data record are later entered automatically through the Delta deployment,

so that only the desired JMS message is generated. Afterward, the web pages can be deployed as usual.

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

```
import com.espirit.moddev.uxbridge.service.UxbService;
uxbService = context.getConnection().getService(UxbService.class);
uxbService.waitSeconds(10);
uxbService.getTimings(context.getStartTime().getTime());
```

In the example, the service waits 10 seconds, until the adapters' answers are evaluated. If there is no answer in this time frame, then the message is classified as having a delivery error. Because the response times can vary depending on message and system, the value can be configured.

3.1.2.8.3 Import 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, initiate variables and write them to the session, so that the methods of the UXB module which are queried in the other scripts can access these.

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 of 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 Import workflows

In order to provide the editor a simple option 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 (approval, release) in FirstSpirit and also 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.

Notes on the procedure for the complete alignment process are found in Chapter 2.1.4.2, page 14 Complete alignment process .

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

To process the exchange format, JAXB is used. 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. Similarly to JPA, here, work 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

Date input is formatted in the FirstSpirit output channel according to the format "yyyy-MM-dd'T'HH:mm:ssZ". So that this format can be read into the JAXB classes, the DateAdapter class has been implemented. 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 on the one hand implement the part prescribed by UX-Bridge (UXBEntity), and the project-specific part (UXBContent) of the exchange format on the other.

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 Domain class Article 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. Access to the data is therefore done 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.

It is to be noted that, after deleting and reinserting an article into the live repository, the ID changes. 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 (package com.espirit.moddev.examples.uxbridge.widget.jpa). It takes care of answering and editing 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

Saving or updating a press release in the live repository.

3.1.3.2.2.2 Command: delete

Deleting a press release in the live repository.

3.1.3.2.2.3 Command: cleanup

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 this Spring Xml file, alongside the database, the JMS, the ArticleHandler and the Camel routes are configured.

3.1.3.2.3.1 CamelContext

In this context, you configure which messages are of interest to this adapter and 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 creating feedback can be found in Chapter 3.5.

3.1.3.2.3.1.1 Reception of messages

Using the From-Tag (<from uri=" activemq:Consumer.newsWidgetHibernate.VirtualTopic.BUS_OUT" />), you can 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 simply 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 get 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 could appear as follows:

activemq:Consumer.myAdapter.VirtualTopic.BUS_OUT

3.1.3.2.3.1.2 Filters for the live repository

By means of XPath expression (//uxb_entity[contains(@destinations, 'postgres')]), the messages which are to be written to this live repository are filtered.

3.1.3.2.3.1.3 Filtering the object type

With the expression //uxb_entity[@objectType = 'news'], the messages are limited 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']" involves an addition to the repository, and "//uxb_entity[@command = 'delete']" involves deletion from the repository.

Before the actual method query, the exchange format is changed by means of JAXB and the Camel instruction <convertBodyTo type="com.espirit.moddev.examples.uxbridge.widget.entity.UXBEntity"/>. The query of the corresponding method in the ArticleHandler is then done with an object of type UXBEntity.

3.1.3.2.3.2 ArticleHandler

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

The ArticleHandler requires both the EntityManagerFactory for access to the database and 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, because the DB structure of the web application 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.

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 procedure of the Hibernate ArticleHandler (see also Chapter 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 example implementation could appear as follows:

```
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 built via the command line:

```
mvn package
```

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

Alternatively, you can start the adapters using the command line:

```
mvn tomcat7:run
```

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

3.1.3.5 Tests contained

In the sample project, there are 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 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 a configuration of the CamelContext here.

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 manner for processing exceptions is defined in the onException tag for this. Which exceptions are processed is defined first. In this case, it is a java.io.Exception. Multiple exceptions can also be specified simultaneously.

Setting the handled tag to true specifies that the exception is handled. 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.

Finally it is specified what happens in the event of an exception. In this case, a message is sent to BUS_IN. The exact structure is described in Chapter "Using the Camel component for generating a response".

Initially, 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, 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 uri="mongodb:myDb?database=newsWidget&collection=article&operation=save" /> is called to transmit the DBObject to the Mongo database. 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 chapter, 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 has to appear as follows:

```
<?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.

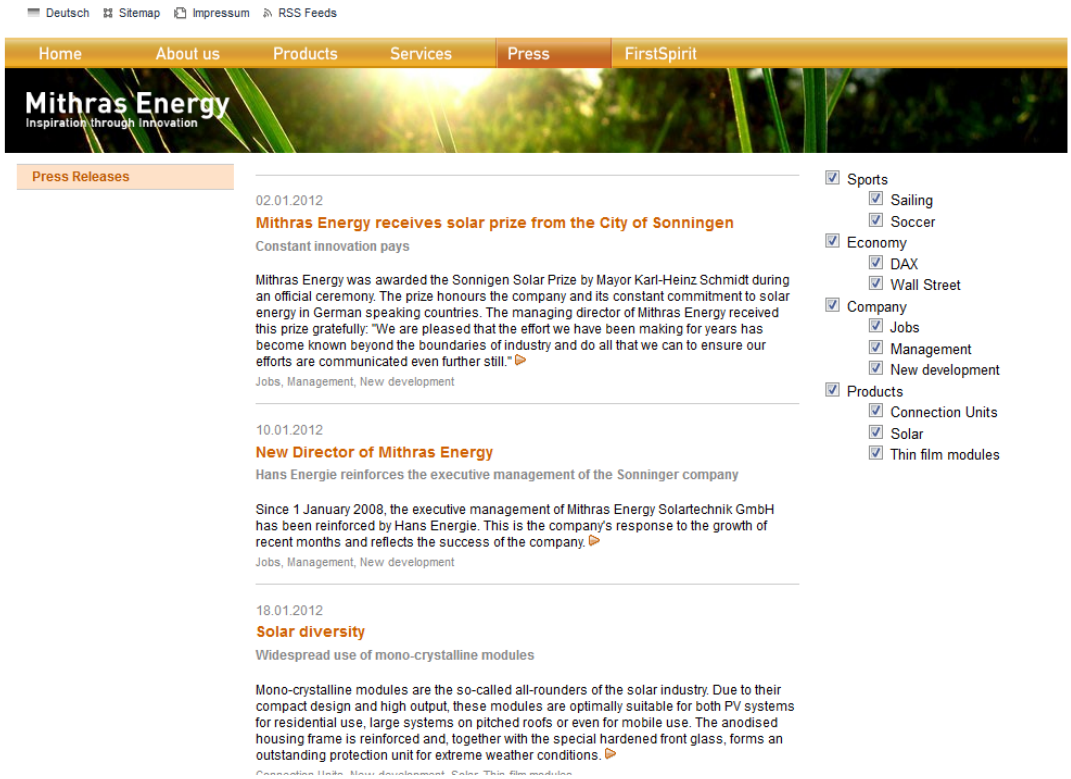
3.3 News Drill-Down scenario

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

The web application is the leading system in this, 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 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, and 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 mark the

categories for which the news is to be shown with checkboxes. 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. A 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 in Version 2.1.0.

3.3.1.1 Configuration

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

The important files here are DataSource.groovy and Config.groovy.

3.3.1.1.1 DataSource.groovy

Here, the database connections are configured for the different environments, in other words, 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 persistence framework Hibernate. Therefore, it is necessary to make sure that the same names are used for the attribute, tables and indices that were already used in the adapter.

3.3.1.3 Rest controller

Create the fitting controller for the domain class news and implement the method "list". 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 which makes 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 administers checking and unchecking 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 with a pagination, which likewise dynamically loads the correct pages via AJAX queries with the correct articles.

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 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 (is 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 can be indicated which should be shown. You pass a string to this with the "cat_1cat_2_cat_4" format in order to, for example, 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 an indicated category. It is queried by the filter method for each individual category.

3.3.1.4.2 RenderService

This service provides a method to render HTML.

3.3.1.4.2.1 renderHtml

You transfer the URL to the method. An http request is run which fetches the HTML snippet. The correctly formatted HTML snippet is then returned.

3.3.1.5 RenderTagLib

This TagLib provides 3 tags to render 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 was successfully started, you can query the news overview page by the following URL:

http://localhost:8080/newsDrilldown/

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

3.3.2 FirstSpirit development

The news scenario is integrated in this tutorial in the standard Mithras Energy project. That is why you import this first and carry out all the following changes in this project.

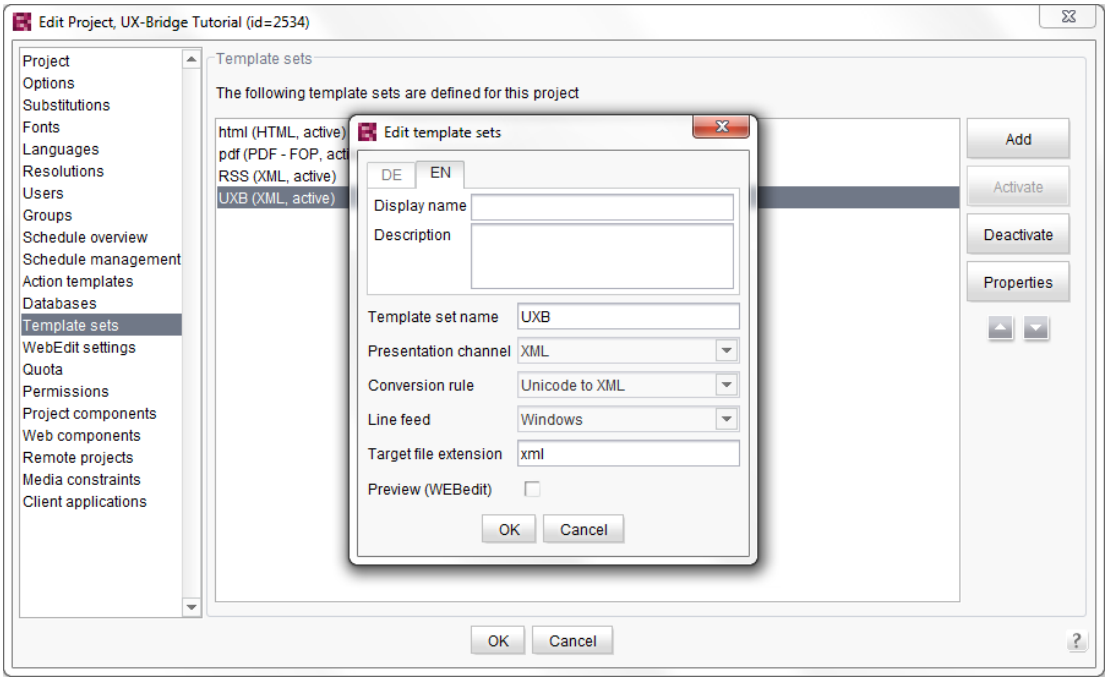
The completely finished FirstSpirit sample project is delivered under the name "uxbridge_tutorial_newsDrilldown.tar.gz", and can be used to view the template code and the settings.

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 UX-Bus, the fields which were defined in the data model are to be output in the form of XML in the corresponding template that is to generate the messages (refer also to 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 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 folder "Header / Footer" in the section templates.

3.3.2.4 Page templates

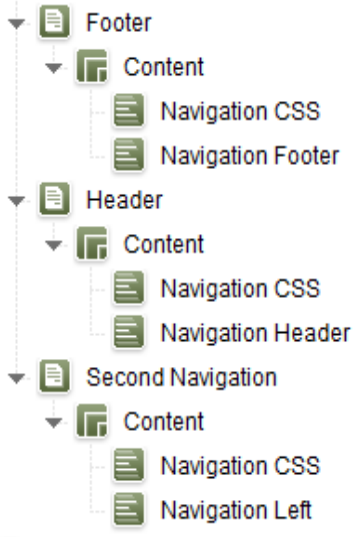
Create a new page template and insert your previously created section templates for the allowed content areas in the Properties tab. Finally, edit your HTML output channel as follows:

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

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

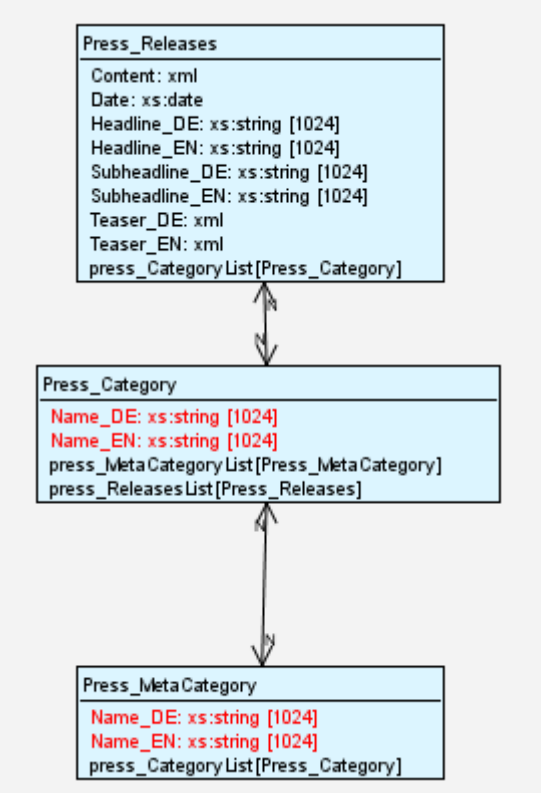
3.3.2.5 Create page

Now, based on the previously created page templates, 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 belong to this. Using an n:m relationship, the table "Press_Category" is referenced, in which you can save the name of a category. 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 based on the following schema which generates the XML, which 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 are simultaneously the content fields which are to be written to the attached content repository by the adapter and therefore are also to be taken into account when creating the data structure.

Also create two table templates alongside 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 generate the JMS messages, and therefore, also the entries in the connected content repositories automatically via a workflow, directly from the content source. Likewise, it is possible to delete objects in FirstSpirit and the connected content repositories directly from the data sources using an additional workflow. To do so, the workflows use, alongside the scripts, table queries and schedules, which are first to 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 parameter "ID", to be newly created.

3.3.2.7.2 Create 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 must first be added to the schedule, which generates the overview pages. The Delta deployment extends this action during runtime by adding the detail page of the data record currently to be generated. Afterward, a script action has to happen which activates UX-Bridge:

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

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

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

```
import com.espirit.moddev.uxbridge.service.UxbService;
uxbService = context.getConnection().getService(UxbService.class);
uxbService.waitSeconds(10);
uxbService.getTimings(context.getStartTime().getTime());
```

In the example, the service waits 10 seconds, until the adapters' answers are evaluated. If there is no answer in this time frame, then the message is classified as having a delivery error. Because the response times can vary depending on message and system, the value can be configured.

3.3.2.7.3 Import 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

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

In uxb_news_example_release_init, the following parameters have to be configured:

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 of 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.

In uxb_news_example_delete_init, the following parameters are configured:

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 Import 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.

Notes on the procedure for the complete alignment process are found in Chapter 2.1.4.2, page 14 Complete alignment process .

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

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

Like with 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

Date input is formatted in the FirstSpirit output channel according to the format "yyyy-MM-dd'T'HH:mm:ssZ". So that this format can be read into the JAXB classes, the DateAdapter class has been implemented. 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 output 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. Here, the actual information of the objects that are distributed via UX Bridge is contained.

In this example, these are in other words 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.

This procedure has, as a consequence, the result that the FirstSpirit-ID (UUID) is no longer unique. Therefore, standard Hibernate/JPA mechanisms are 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, made up of the FirstSpirit-ID and the language.

3.3.3.2.1 News, NewsCategory, NewsMetaCategory

The structure of the classes corresponds to those in the database schema defined in FirstSpirit. This procedure is not absolutely necessary, but is being described here as an aid to understanding.

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, for each of the supported commands, a unique method was implemented in the handler.

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, categories and meta categories are, however, saved separately.

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

3.3.3.3.2 Command: delete

Deleting a press release in the live repository and the detail page belonging to it as defined in the schedule script (see Create schedule) on the web server. In order for the methods to be able to find the correct page on the web server, in the applicationContext.xml in the news handler bean, the "webpath" parameter on the path to the web server directory (for example, "(/home/tomcat/ webapps") must be set.

```
<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

Deleting 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 creating feedback can be found in Chapter 3.5.

3.3.3.4.1 The route uxbridge-commands

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

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

3.3.3.4.2 Message source

With the From tag, the integration framework from which the data is read (Apache Camel) is indicated. 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 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 simply 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 get 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 could appear as follows:

activemq:Consumer.myAdapter.VirtualTopic.BUS_OUT

3.3.3.4.3 Filters

Through filtering messages, messages which are unimportant to the NewsHandler - in other words, messages for a different repository or from a different type of object - are filtered out.

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 to this expression are not handled further. The messages can be simultaneously written into 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". Here, too, the messages which do not affect this expression cannot be further handled. Usually, messages always only contain 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 via a JAXB in a Java class.

3.3.3.4.3.5 Methods query

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

Finally, at the 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 example implementation could appear as follows:

```
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 ServletContainer (Tomcat, Jetty etc.).

Alternatively, you can start the adapters using the command line:

```
mvn tomcat7:run
```

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

3.3.3.6 Tests contained

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

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

3.4 Using the UXB service API

If you would like to use the UXBService in a module or script, the API can be used. To do so, the API jar file in the corresponding version has to be added to the class path.

Then you receive access to the UXBService using the following call:

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

You can find an example implementation 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. Afterwards, the included sample executables can be used.

3.4.2 Use

Both example 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.

In order to have access to 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 for generating 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

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, the structure of the URL changes accordingly in relation to the example below.

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 added to the URL at the end.

The complete structure could then appear as follows:

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

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

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

3.5.4 Parameter

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 ?. 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 content is no longer complete and a part, such as the root element for the XML document, is missing. Since processing requires the entire XML document, however, the document has to be stored temporarily before processing 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 Appendix

4.1 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]
```