FirstSpirit™
Unlock Your Content

UX-Bridge Entwicklerhandbuch

Version	**1.7**
Status	**RELEASED**
Datum	**2014-07-09**
Abteilung	Produktmanagement
Autor/ Autoren	C. Feddersen
Copyright	2014 e-Spirit AG

e-Spirit AG

Barcelonaweg 14
44269 Dortmund | Germany

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

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

e-Spirit

Inhaltsverzeichnis

Einleitung

Das Modul „UX-Bridge“ kommt dem Trend von dynamischen Websites nach. Immer dort, wo eine Vorgenerierung von Inhalten nicht möglich ist, muss dynamisch auf die CMS-Inhalte zugegriffen werden. Dynamisch bedeutet in diesem Fall, dass sich der Inhalt für jeden Website-User und zu jedem Zeitpunkt ändern kann. Die UX-Bridge bietet eine Infrastruktur für die Anforderung einer dynamischen Content-Delivery-Plattform. Somit ergänzt das Modul den hybriden Architekturansatz um eine Standardkomponente für dynamische Content-Auslieferung. Weitere Informationen finden Sie im Whitepaper, Kapitel 1.3.

Diese Dokumentation soll die Entwicklung mit der UX-Bridge Infrastruktur unterstützen und einen Einblick in die Konzeption geben.

Es beschreibt im ersten Kapitel allgemein die Schritte, die beim Einsatz der UX-Bridge zu berücksichtigen sind. Das Kapitel deckt dabei die Themen Installation, Überlegungen zum Datenaustauschformat, Einrichtung in FirstSpirit, Routing, sowie Adapter- und Webapplikationseinbindung ab.

Kapitel 3 erläutert die Verwendung der UX-Bridge anhand von zwei Tutorials. Hier wird an konkreten Beispielen die Implementierung in FirstSpirit, die Erstellung eines Adapters sowie der Webanwendung Schritt für Schritt exemplarisch durchgeführt.

1 Konzeption

Bestehen in einem Projekt Anforderungen, die mit Hilfe der UX-Bridge umgesetzt werden sollen, so stellen sich vor der Implementierung einige Fragen, die im Rahmen der Konzeption bedacht und ggf. beantwortet werden sollten.

Bei der Konzeption können zwei Fälle unterschieden werden: Wird ein FirstSpirit Projekt vollständig neu entwickelt, so liegt der Fokus der Entwicklung auf der optimalen Umsetzung der fachlichen Anforderungen. Gleichzeitig soll das Konzept hinreichend flexibel sein, um auch zukünftige Anforderungen leicht umsetzen und integrieren zu können. Soll die UX-Bridge andererseits in ein bereits bestehendes Projekt integriert werden, so stellt sich primär die Frage, wie die UX-Bridge am besten in die bestehenden Konzepte und Mechanismen eingebunden werden kann.

Im Folgenden werden einige Punkte betrachtet, die in vielen Projekten relevant sein werden.

1.1 Generierungs- und Deploymentkonzept

In vielen Projekten erfolgt die Aktualisierung der Webseite durch zyklische Generierungs- und Deploymentaufträge. Diese Aufträge führen einen Komplett- bzw. einen Teilabgleich durch. Unter Umständen können diese Aufträge auch manuell ausgeführt werden. Soll in diesem Szenario die UX-Bridge verwendet werden, so reicht es oft aus, die bestehenden Aufträge um die UX-Bridge spezifischen Aufgaben zu erweitern. Details dazu finden Sie im Kapitel 2.1.4 „Auftrag anlegen und konfigurieren“.

Innerhalb der Generierung wird für jede Seitenreferenz, die innerhalb der UX-Bridge Generierungsaufgabe erzeugt wird, eine Nachricht an den UX-Bus gesendet. Innerhalb des Projektes sollte also darauf geachtet werden, dass innerhalb dieser Aufgabe nur die notwendigen Seitenreferenzen erzeugt werden. Sollen in einem Projekt z.B. nur die News (gepflegt über eine Datenquelle) über die UX-Bridge ausgeliefert werden, so sollten in der UX-Bridge Generierungsaufgabe auch nur die notwendigen Contentprojektionsseiten generiert werden. Um diese Einschränkung vorzunehmen, kann zum Beispiel eine Teilgenerierung verwendet werden. Alternativ dazu kann auch eine Vollgenerierung genutzt werden. Allerdings sollte dann „uxbSkipMessage“ eingesetzt werden, um die Generierung von Seitenreferenzen im UX-Bridge Ausgabekanal abzubrechen, die keine Nachrichten erzeugen sollen (s. Online Dokumentation für FirstSpirit: Vorlagenentwicklung\Vorlagensyntax\Systemobjekte\#global\vorschaubezogen\Abbr

uch einer Vorschau/Generierung).

In anderen Projekten wird ein Großteil der Änderungen über Arbeitsabläufe freigegeben, die anschließend eine Generierung und ein Deployment der beteiligten Objekte vornehmen. Hierbei werden zumeist die geänderten Objekte per Skript ermittelt, die dann als Startknoten einer Teilgenerierung definiert werden. Das Löschen von Objekten erfolgt durch den Einsatz von Löscharbeitsabläufen. Die UX-Bridge lässt sich in solche Szenarien ebenfalls problemlos integrieren (s. Kapitel 2.1.7 Workflow-Kopplung).

Spielt die Zeit, in der Objekte auf der Webseite verfügbar sein müssen, eine große Rolle, so ist die Verwendung des arbeitsablauforientierten Ansatzes oft die bessere Wahl. Je weniger Objekte im Zuge einer Generierung erzeugt werden müssen, desto schneller sind die Objekte auf der Webseite verfügbar. Mit FirstSpirit5 steht mit der DeltaGenerierung (Developer API\Delta Generation, Access-API\Generate Task) ein vereinfachter Mechanismus für solche Szenarien zur Verfügung. Unter FirstSpirit4 kann dies bereits durch Nutzung der Revisions-API erreicht werden.

Insgesamt gliedert sich die UX-Bridge somit in das bestehende oder in ein neu aufzusetzendes Generierungs- und Deployment-Konzept ein und bringt dafür keine eigenständige (getrennte) Lösung mit.

1.2 Datenmodell und Adapter

Soll die UX-Bridge in ein bestehendes FirstSpirit-Projekt integriert werden, so ist das Datenmodell in FirstSpirit oft bereits vorgegeben. Bei stark strukturierten Inhalten durch das Datenbankschema, bei schwach strukturierten Inhalten durch Formulare der Seiten- und Absatzvorlagen. Hier sollte geprüft werden, ob die zu erstellenden Webapplikationen, die später auf die Daten der UX-Bridge zugreifen soll, mit diesem Datenmodell arbeiten können oder ob ein anderes Datenmodell sinnvoller ist. Dies könnte der Fall sein, wenn die Webapplikation ein deutlich einfacheres bzw. deutliches komplexeres Datenmodell benötigt.

Im letzteren Fall sind die FirstSpirit Daten also nur einen Teil des Datenmodells der Webapplikation. Im erstgenannten Fall ist für viele Webapplikationen auch eine Teilmenge des in FirstSpirit hinterlegten Datenmodells ausreichend. Zusätzlich ist zu überlegen, ob aus Performancegründen das Datenmodell der Webapplikation denormalisiert wird.

Wurde die Entscheidung getroffen, abweichende Datenmodelle zu verwenden, so ist die Frage zu klären, in welchem Schritt die Transformation von einem Datenmodell in das andere Datenmodell vorgenommen werden soll. Die Antwort ist sicherlich

projektspezifisch, so dass hier nur die möglichen Varianten beschrieben werden. Eine Bewertung muss im Projektkontext erfolgen:

a) Die Transformation erfolgt über die CMS-Syntax in den Vorlagen, die das für die UX-Bridge notwendige XML erzeugen (s. Kapitel 2.1.2 „Datenaustauschformat – FS Templating vs. WebApp Entwicklung“). Die Adapter selbst nehmen keine größeren Transformationen vor, sondern schreiben die Objekte so in das Content-Repository, wie es das Datenaustauschformat definiert hat.

b) Das Datenaustauschformat entspricht 1:1 dem Datenmodell in FirstSpirit. Die Transformation in das Datenmodell für das Content-Repository wird innerhalb des Adapters vorgenommen.

c) Ein hybrider Ansatz, der in beiden Komponenten Transformationen vornimmt. Je nachdem wo es sich einfacher zu realisieren lässt.

Folgende Überlegungen können bei der Bewertung helfen:

1) Sollen die Daten in mehrere Content-Repositories geschrieben werden, so ist es oft sinnvoll, das Datenaustauschformat allgemein zu halten und etwaige notwendige bzw. sinnvolle Transformationen für das Schreiben in das Content-Repository innerhalb des Adapters vorzunehmen.

2) Sollen die Daten in mehrere Content-Repositories geschrieben werden, so kann für jedes Content-Repository ein eigener Adapter implementiert werden, der nur die notwendige Logik für dieses Repository enthält. Alternativ kann ein Adapter die Daten in beide Content-Repositories schreiben. Dies bietet sich z.B. an, wenn das Schreiben innerhalb einer Transaktionsklammer erfolgen soll.

3) Behandelt ein Adapter nur einen bestimmten Objekttyp oder bündelt man die Behandlung von mehreren Objekttypen in einem Adapter? Mit Objekttypen sind hier unterschiedliche Arten von Inhalten gemeint. So möchte man zum Beispiel alle Produkte und alle News aus einer FirstSpirit Datenquelle über die UX-Bridge zur Verfügung stellen. Hier spielt zum Beispiel eine Rolle ob die beiden Objekttypen in das gleiche Content-Repository und/oder Datenmodell überführt werden sollen oder nicht.

4) Generell ist zu überlegen ob man aus Gründen der Entkopplung und Wartung lieber auf mehrere, dafür aber schlanke, Adapter setzt oder lieber einen Adapter verwendet, der sämtliche Logik enthält.

5) Ein allgemein gehaltenes Datenaustauschformat hat den Vorteil, dass nur

eine Nachricht über den UX-Bus geschickt werden muss, die dann aber von mehreren Adaptern verarbeitet und in mehrere Content-Repositories geschrieben werden kann. Zudem sind ggf. keine Anpassungen am Datenaustauschformat notwendig, wenn im Projektverlauf neue Content-Repositories und Webapplikationen angebunden werden sollen.

6) Soll die UX-Bridge mit einem System kommunizieren, das bereits JMS-Nachrichten entgegen nehmen kann, so kann es sinnvoll sein, eine Transformation direkt auf dem UX-Bus vorzunehmen. So muss kein Adapter implementiert werden, der wiederum nur JMS-Nachrichten erzeugen würde. In solchen Fällen kann das Routing auf dem UX-Bus durch entsprechende Transformationsanweisungen erweitert werden.

1.3 Nachrichtenverteilungen / Routing

Wie im vorherigen Kapitel erwähnt wird für jede Seitenreferenz innerhalb der Generierung eine Nachricht erzeugt und an den UX-Bus gesendet. Teil des UX-Buses ist eine Routing-Komponente, die die Nachricht entgegen nimmt und an einen anderen sogenannten „Endpunkt“ weiterleitet bzw. routet. Details zur Standardkonfiguration finden Sie im Kapitel 2.4 „Routing“.

Diese Konfiguration kann für die projektspezifischen Anforderungen angepasst werden. Hierbei steht mit der Apache Camel Spring XML Syntax eine einfache Domain-Specific Language (DSL) zur Verfügung, mit der alle gängigen Enterprise Integration Patterns (siehe http://camel.apache.org/enterprise-integration-patterns) umgesetzt werden können. Mit diesem mächtigen Integrationsframework kann der UX-Bus als Informationsdrehscheibe für den Webauftritt und alle beteiligten Systeme verwendet werden.

Hier einige Beispiele, die durch ein Routing auf dem UX-Bus umgesetzt werden können:

1) Es wird ein Content-Router konfiguriert, der bestimmte Nachrichten nur an bestimmte Endpunkte/Adapter versendet

2) Es können neue Endpunkte konfiguriert werden, die als Schnittstelle zu Webapplikationen oder Backendsystemen dienen. Über diese kann zum Beispiel ein Adapter eine Webapplikation anweisen, ihren Cache zu leeren, da neue Daten in das Content-Repository geschrieben wurden.

3) Bestehende Drittapplikationen können Nachrichten an den UX-Bus schicken, die dann von der Webapplikation, den Adaptern oder FirstSpirit verarbeitet

werden.

In der Standard-Konfiguration der UX-Bridge ist bereits ein Routing konfiguriert, welches für Standardszenarien ausreicht. Nur für komplexere Szenarien (s. „Datenmodell und Adapter“) sind projektspezifische Anpassungen notwendig.

2 Quick Walkthrough

In diesem Kapitel werden die Schritte beschrieben, die notwendig sind, um die UX-Bridge in einem Projekt einzusetzen. Der Quick Walkthrough ist für den erfahrenen FirstSpirit-Vorlagenentwickler und Webapplikationsentwickler gedacht. Eine genauere Schritt-für-Schritt-Anleitung finden Sie im Kapitel 3 „Tutorials“.

2.1 FirstSpirit

2.1.1 Installation

Ausgangspunkt für die Entwicklung im UX-Bridge Kontext ist die Installation der Komponenten.

Dazu sollte zunächst das mitgelieferte Modul in den Server-Einstellungen des FirstSpirit-Servers installiert werden. Dadurch wird der UX-Bridge Service gestartet und die UX-Bridge Komponenten sind in den Projekten des Servers verfügbar (vgl. UX-Bridge Installationshandbuch: „Installation FirstSpirit Modul“).

Neben dem FirstSpirit-Modul ist auch die Installation des UX-Buses notwendig. Zur lokalen Entwicklung ist die Installation im Standalone-Betrieb empfohlen (vgl. UX-Bridge Installationshandbuch: „Standalone-Betrieb“).

2.1.2 Datenaustauschformat – FS Templating vs. WebApp Entwicklung

Die Architektur der UX-Bridge erlaubt es, die Entwicklung von Lösungen auf Basis der UX-Bridge auf zwei Rollen zu verteilen. Ein Vorlagenentwickler erstellt die notwendigen Vorlagen, Arbeitsabläufe und Aufträge. Ein (Web-) Applikationsentwickler entwickelt den Adapter für das Content-Repository und die (Web-) Anwendung. Werden die Rollen durch unterschiedliche Personen wahrgenommen, so sollte während der Konzeption ein gemeinsames Datenaustauschformat definiert werden. Hiermit ist das Datenformat gemeint, was durch die Vorlagen erzeugt und über den UX-Bus als Nachricht an den Adapter

verschickt wird.

Das Datenaustauschformat bildet also die Schnittstelle zwischen den Komponenten und damit auch zwischen den beiden Rollen. Aus Sicht des Vorlagenentwicklers handelt es sich hierbei um das Endprodukt seiner Arbeit. Für den (Web-) Applikationsentwickler stellt es den Input für den Adapter dar. Dabei wird seitens der UX-Bridge nur ein äußerer Container vorgegeben. Der Rest kann frei definiert werden (s. nächstes Kapitel).

Ein geschickt gewähltes Datenaustauschformat kann den Implementierungsaufwand deutlich reduzieren. Folgende Fragestellungen können bei der Wahl helfen:

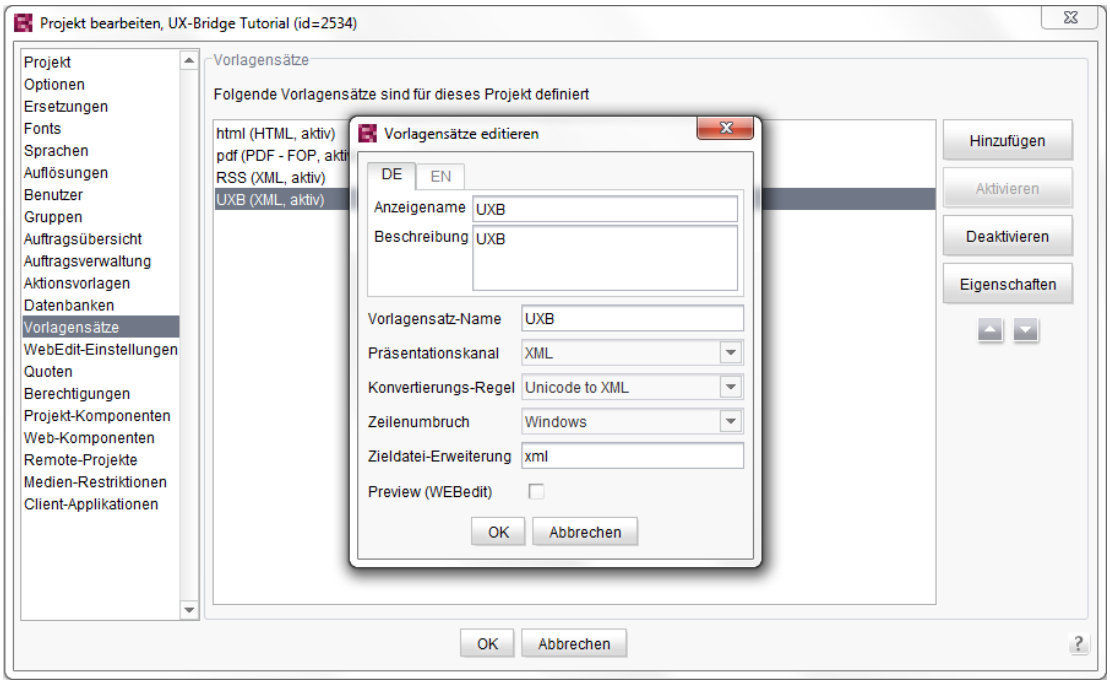
- Existieren die Daten und damit das Datenmodell bereits in FirstSpirit? Falls ja, ist man gewissen Beschränkungen unterworfen. Falls nein, bietet es sich an das Datenaustauschformat möglichst nah an das Datenformat im Content-Repository bzw. der Webapplikation anzulehnen.

- Aus Performancegründen kann es sinnvoll sein, die Daten denormalisiert in das Content-Repository zu schreiben. Mögliche Inkonsistenzen können durch ein Volldeployment korrigiert werden, da die Daten in FirstSpirit in der Regel weiterhin normalisiert vorliegen.

- Sollen die Daten in mehr als ein Content-Repository geschrieben werden? Falls ja, ist zu ermitteln ob ein Datenaustauschformat ausreichend ist und der/die Adapter die Transformation und Persistierung in das Content-Repository übernehmen. In manchen Fällen kann es auch effizienter sein, zwei Datenaustauschformate durch FirstSpirit erzeugen zu lassen, die dann ohne einen Transformationsschritt in das jeweilige Content-Repository übernommen werden können.

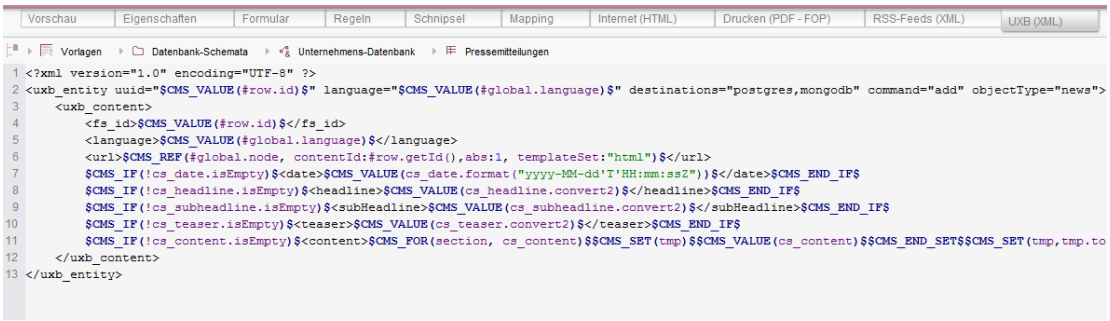
2.1.3 Ausgabekanal anlegen und befüllen

Um die UX-Bridge zu nutzen, sollte zunächst in den Projekteinstellungen unter „Vorlagensätze“ ein neuer Vorlagensatz (UXB) angelegt werden. Als Präsentationskanal sollte „XML“ als Konvertierungsregel „Unicode to XML“ und als Zieldateierweiterung „xml“ konfiguriert werden.



Die Konvertierungsregel „Unicode to XML“ (s. Konvertierungsregel Unicode to XML) dient dazu Sonder- und Steuerzeichen in korrespondierende XML-Entitäten umzuwandeln, die sonst in XML als Teil der Markupsprache interpretiert werden würden bzw. ungültige Zeichen darstellen würden.

Um Nachrichten an den UX-Bus zu schicken, sind in der entsprechenden Vorlage, die die Nachrichten erzeugen soll, die Felder, die im Datenaustauschformat definiert wurden, in Form von XML auszugeben.



Vorgegeben ist lediglich folgende Struktur:

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

```
            objectType = String
>
            <uxb_content>
                        […] definiertes Datenaustauschformat […]
            </uxb_content>
</uxb_entity>
```

Property	Beschreibung	Beispiel	Pflichtfeld
Uuid	Eindeutiger Bezeichner der Objektes z.B. fs_id	1234	Ja
destinations	Ziel(e) der Nachricht (Live-Repository, kommasepariert),	postgres,mongodb	Ja
language	Sprache der Nachricht	DE	Nein
command	Auszuführendes Kommando vom Adapter (z.B. anlegen/löschen)	Add	Nein
objectType	Objektyp der vom Adapter ausgewertet wird (z.B. News, Products)	News	Nein

Die Attribute language, command und objectType sind optional, haben sich aber bei den von e-Spirit implementierten Adaptern als hilfreich erwiesen.

2.1.4 Auftrag anlegen und konfigurieren

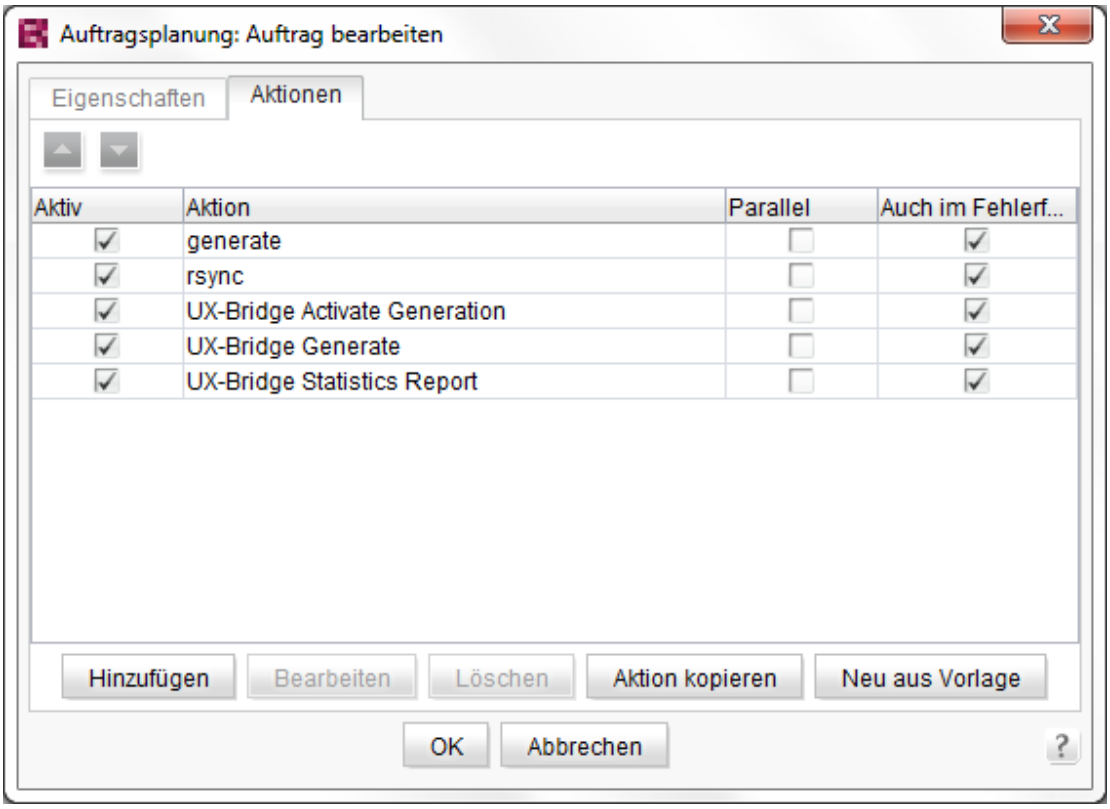
Um die Daten von FirstSpirit in Nachrichten umzuwandeln, die vom UX-Bus weiterverarbeitet werden können, muss zwingend ein Auftrag angelegt oder ein bestehender Auftrag so erweitert werden, dass er XML generiert und dieses an den UXB-Service weiterreicht, der daraus eine Nachricht erzeugt und diese an den UX-Bus versendet. Der Auftrag kann später über einen Workflow (s. Release Workflow) gestartet werden.

In diesem Kapitel sollen nun zwei Aufträge erklärt werden, die wahrscheinlich in jedem Projekt, welches die UX-Bridge nutzt, benötigt werden.

2.1.4.1 Teilgenerierung

Um neue Inhalte schnell auf der Webseite zu publizieren, bietet es sich an, einen Auftrag anzulegen bzw. so zu erweitern, dass er die im Folgenden beschriebenen Schritte durchführt und hierbei nur die neuen Inhalte generiert und veröffentlicht.

Ein typischer Generierungsauftrag, der die UX-Bridge nutzt, gliedert sich in mehrere Aktionen wie nachfolgend beschrieben:



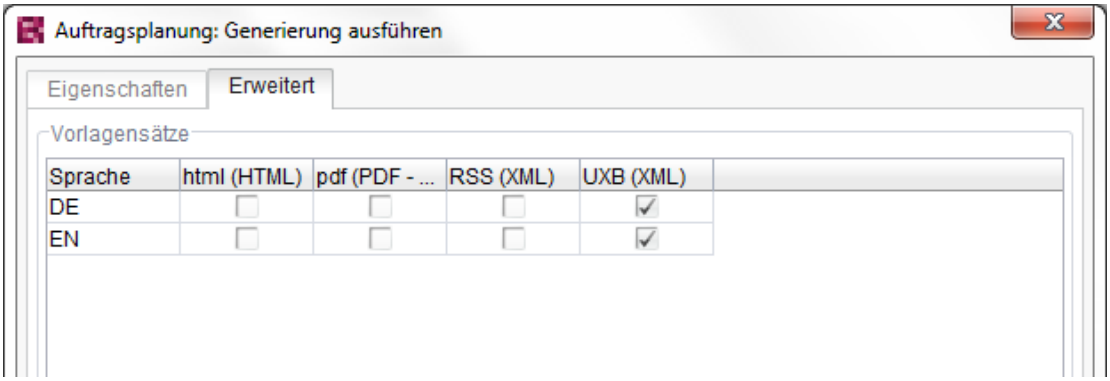
Zunächst sollten in einer Generierungsaktion die komplett statischen Seiten, die zur Anzeige auf der Webseite notwendig sind (z.B. News-Übersichtsseiten), generiert werden. Diese Generierungsaktion findet sich auch in den bisher üblichen Deployment-Aufträgen und muss dann nicht angepasst werden.

Im nächsten Schritt sollten dann die im vorherigen Schritt generierten Inhalte wie gewohnt auf den Webserver übertragen werden (im Beispiel über rsync). Auch in diesem Schritt sind in der Regel keine Anpassungen notwendig.

Um die UX-Bridge einzusetzen, ist danach eine neue, weitere Aktion hinzuzufügen, die mittels Skriptaufruf die Generierung für die UX-Bridge aktiviert:

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

In der darauf folgenden Generierungsaktion sollten dann genau die Seite generiert werden (z.B. News-Detailseite), die das XML erzeugt, das an den UXB-Service weitergereicht werden soll. Wichtig zu beachten ist, dass in den erweiterten Eigenschaften auch der UXB-Vorlagensatz aktiviert wurde.



Die Delta-Generierung in FirstSpirit 5, passt die Generierungsaktion automatisch so an, dass nicht mehr alle Seiten, sondern nur noch die gewünschte Seite, generiert werden.

Dazu kann z.B. in einem Workflowskript, das die Generierung anstößt, die zu generierende Seite an den Auftrag übergeben werden.

Die letzte Aktion „UX-Bridge Statistics Report“ ist optional und ermöglicht es, die Durchlaufzeiten für die Nachrichten im Bus bis zur Veröffentlichung auf der Webseite zu messen.

```
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)
```

Dazu ist folgender Skriptaufruf notwendig:

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

Der Service wartet maximal den in der Servicekonfiguration der UX-Bridge definierten Zeitraum, bis die Antworten der Adapter ausgewertet werden. Kommt in diesem Zeitfenster keine Antwort, so gilt die Nachricht als fehlerhaft zugestellt. Da die Antwortzeiten je nach Nachricht und System variieren können, ist dieser Wert konfigurierbar.

2.1.4.2 Komplettabgleich

Es ist sinnvoll einen weiteren Auftrag anzulegen, der einen Komplettabgleich durchführt, um den Datenbestand im Content-Repository auf dem aktuellsten Stand zu halten. Hierfür müssen die in FirstSpirit gelöschten Daten ebenfalls auch im externen Repository gelöscht werden.

Es empfiehlt sich folgende Vorgehensweise:

1) Vollgenerierung der statischen Seiten in FirstSpirit.

2) UX-Bridge-Auftrag ausführen, um alle Daten ins ContentRepository zu schreiben (siehe dazu den vorherigen Abschnitt). Diese Daten bekommen einen aktuellen Timestamp übergeben, der im ContentRepository im Feld „lastmodified“ gespeichert wird.

3) Skript ausführen, welches die cleanup Methode mit einem Timestamp als Parameter aufruft, der die neueste Projektrevision des Auftrags beschreibt. Die cleanup Methode löscht dann alle Daten, die eine ältere Zeit, als die übergebene, im lastmodified Feld gespeichert haben. Das sind die Daten, die im FirstSpirit-Projekt bereits gelöscht wurden und somit im Schritt 2 keinen neuen Timestamp in „lastmodified“ gespeichert haben.

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

2.1.4.3 Historische Daten

Um historische Daten mittels UX-Bridge zu generieren, muss eine neue Script-Aktion vor der Aktion "UX-Bridge - Activate Generation" hinzugefügt werden z.B.

```
import java.util.Date;
```

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

Das gesetzte Datum wird dann in der folgenden Aktion ("UX-Bridge - Activate Generation") berücksichtigt.

2.1.4.4 Auftrag Start / Ende im Adapter erkennen

Um im Adapter auf den Start und / oder das Ende eines Auftrags unter Umständen gezielt reagieren zu können, kann die UX-Bridge zu Beginn und am Ende der Generierung jeweils automatisch eine separate Nachricht versenden (s. Installationshandbuch).

Die Startnachricht besitzt dabei folgendes Format:

```
<uxb entity projectName="PROJEKT NAME" status="start"
schedulerId="AUFTRAGS_ID" createTime="ZEITPUNKT_DER_NACHRICHT"
projectId="PROJEKT_ID" startTime="STARTZEITPUNKT_DES_AUFTRAGS" />
```

Bei einer Vollgenerierung wird noch das Attribut command=“startMaintenanceMode“ hinzugefügt.

Die Endnachricht besitzt folgendes Format:

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

Bei einer Vollgenerierung wird noch das Attribut command=“stopMaintenanceMode“ hinzugefügt.

2.1.4.5 Root-Attribute erweitern

Die Wurzelknoten der über die UX-Bridge verschickten Nachrichten besitzen standardmäßig bereits einige Attribute, wie beispielsweise den Projektnamen oder die ID des Auftrags (siehe auch Kapitel 2.1.4.4, Seite 15). Diese lassen sich durch beliebige, weitere Attribute erweitern. Zur Angabe dieser Root-Attribute muss dem Generierungsauftrag als erste Aktion das Skript „Configure UXB Message Header“ hinzugefügt werden, welches den folgenden Inhalt besitzt:

```
import java.util.HashMap;

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

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

Das Skript erzeugt eine HashMap, in der die gewünschten Attribute in Form von Key/Value-Paaren hinzugefügt werden können. Die HashMap wird anschließend an den Kontext des Auftrags übergeben, damit der UXBService während der Erzeugung der Nachrichten auf sie zugreifen kann.

2.1.5 Überspringen von Seiten bei der Generierung

Sollen bei der Nachrichtengenerierung Seiten übersprungen werden, so ist dies durch das setzen der Seitenvariable „uxbSkipMessage“ möglich.

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

Die Benutzung von „stopGenerate“ (s. Online Dokumentation für FirstSpirit: Vorlagenentwicklung\Vorlagensyntax\Systemobjekte\#global\vorschaubezogen\Abbruch einer Vorschau/Generierung) wird nicht unterstützt und führt in diesem Fall zu ungültigem XML und dadurch Fehlermeldungen im Log.

2.1.6 Erweiterte Projektinformationen auslesen

Die UX-Bridge kann zu Beginn der Generierung eine Nachricht mit erweiterten Projektinformationen verschicken (s. Installationshandbuch). Zu diesen Informationen zählen zurzeit die definierten Projektsprachen und Auflösungen.

Beispiel:

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

2.1.7 Workflow-Kopplung

Die Publikation von Inhalten über die UX-Bridge kann sowohl direkt über Skripte und Aufträge, als auch indirekt über Workflows gestartet werden.

2.1.7.1 Release Workflow

Um Inhalte zu publizieren, muss ein bestehender Workflow lediglich um ein Workflow-Skript erweitert werden, das einen Auftrag startet, der neben Generierung und Deployment auch die XML-Nachrichten erzeugt und an den UXB-Service weiterreicht (s. Teilgenerierung).

2.1.7.2 Delete Workflow

Um Inhalte zu löschen, muss ein bestehender Löschworkflow um ein Workflow-Skript erweitert werden, das eine XML-Nachricht erzeugt, die an den UXB-Service weitergereicht wird.

Der Aufruf des UXB-Services im Skript lautet wie folgt, wobei „msg“ (String) der XML-Nachricht entspricht:

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

uxbService.removeUxbEntry(msg);
```

Die XML-Nachricht folgt dabei folgendem Muster:

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

Property	Beschreibung	Beispiel	Pflichtfeld
Uuid	Eindeutiger Bezeichner der Objektes z.B. fs_id	12345	Ja
Destinations	Ziel(e) der Nachricht (kommasepariert)	postgres	Ja
Command	Auszuführendes Kommando vom Adapter	delete	Ja
Language	Sprache der Nachricht	DE	Nein
objectType	Objekttyp, der vom Adapter ausgewertet wird (z.B. News, Products)	news	Nein

2.2 Adapter

Adapter dienen dazu, die Daten aus den JMS-Nachrichten auszulesen und in die ausgewählten Repositories zu schreiben. Im Tutorial werden zwei Adapter beispielhaft als Webanwendungen realisiert, aber auch andere Implementierungen (z.B. standalone Java) sind möglich.

2.2.1 Rückmeldung

FirstSpirit erwartet eine Antwort in Form eines XML-Dokumentes vom Adapter, nachdem eine Nachricht in ein Repository geschrieben wurde. Die Antwort wird sowohl bei einer erfolgreichen, als auch bei einer fehlhaften Verarbeitung erwartet. Das Antwort XML-Dokument ist wie folgt aufgebaut:

```
<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	Beschreibung	Beispiel	Pflichtfeld
destinations	Das Ziel-Repository, in das das Objekt geschrieben wurde bzw. geschrieben werden sollte.	postgres	Ja
startTime	Timestamp des Startes der Aktion (Wird von FirstSpirit bei der Aktion ins XML Dokument eingehängt)	1314567899516	Ja
finishTime	Timestamp der Fertigstellung des Kommandos	1314567899516	Ja
path	FirstSpirit interner Pfad (Wird von FirstSpirit bei der Aktion ins XML Dokument eingehängt)	the/Path/to/	Ja

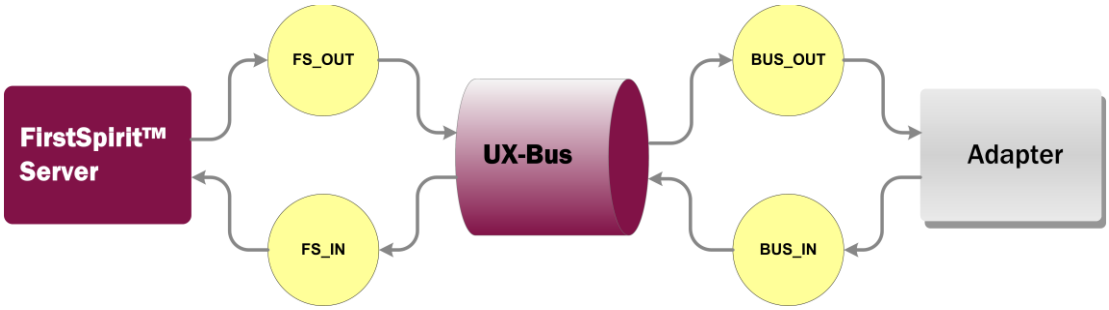
status	Status der Aktion. Mögliche Werte: „OK“ bei Erfolg, „FAIL“ im Fehlerfall	OK	Ja
uuid	Eindeutiger Bezeichner der Objektes z.B. fs_id	123456	Ja
schedulerId	Eindeutige ID des Auftrages (Wird von FirstSpirit bei der Aktion ins XML Dokument eingehängt)	123456	Ja
command	Ausgeführtes Kommando vom Adapter	delete	Nein
language	Sprache der Nachricht	DE	Nein
createTime	Timestamp der Erstellung der Aktion (Wird von FirstSpirit bei der Aktion ins XML Dokument eingehängt)	1314567899516	Nein
uxb_error	Das Containerelement für die im Fehlerfall vorhandene Fehlermeldung	com.mongodb. MongoException	Nein

2.3 WebApplikation

Durch die offene Architektur der UX-Bridge und den Umstand, dass die Art und Anzahl der Repositories nicht vorgegeben wird, sind die Technologie und das Framework, um die WebApplikation zu entwickeln, frei wählbar. Es bietet sich an, die Auswahl der Technologie und des Frameworks sowohl an den Anwendungsfall, als auch an das vorhandene KnowHow bzw. die Unternehmensstandards anzupassen.

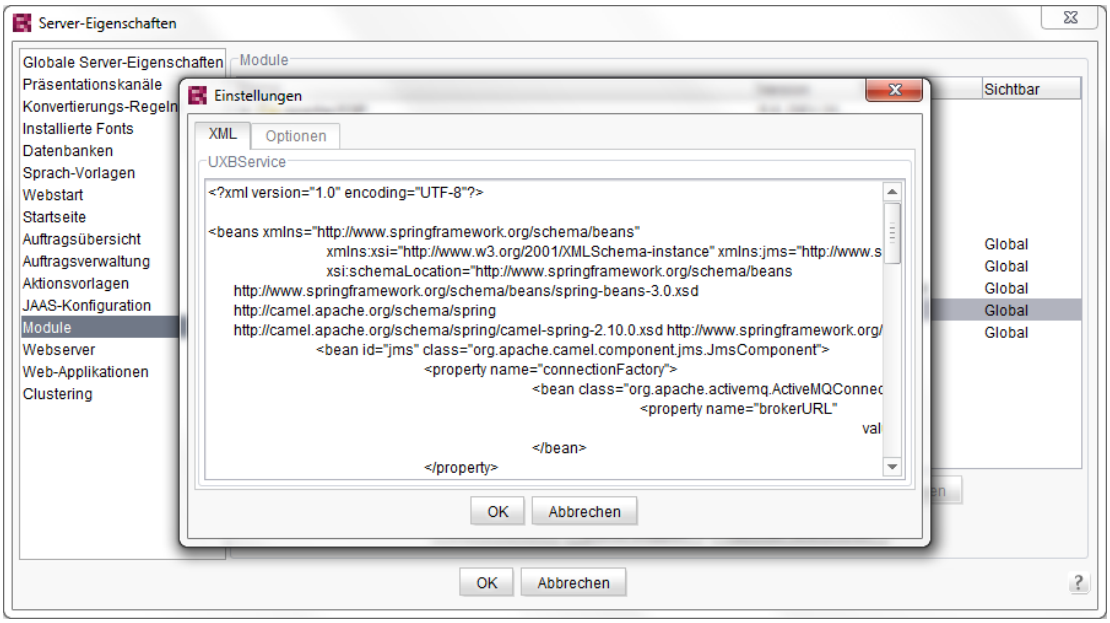
2.4 Routing

Die UX-Bridge nutzt Apache Camel zum Routen von Nachrichten. Als Transport- und Nachrichtenprotokoll kommt der Java Message Service (JMS) zum Einsatz. Die beiden beteiligten Komponenten FirstSpirit-Server und Adapter fungieren dabei jeweils sowohl in der Rolle eines Producers, der Nachrichten erzeugt und in einem Endpunkt zur Verfügung stellt, als auch in der Rolle eines Consumers, der Nachrichten von einem Endpunkt abholt und weiterverarbeitet. Der UX-Bus übernimmt in diesem Szenario lediglich das Routing der Nachrichten zwischen den beteiligten Endpunkten.



2.4.1 Endpunkte in FirstSpirit

Die Konfiguration des UXB-Service ist über die FirstSpirit-Server-Konfiguration und den Unterpunkt Module erreichbar. Aus dem ausgeklappten Modulbaum muss der UXB-Service selektiert werden und über den Button „Konfigurieren“ öffnet sich die Konfiguration des Services (Spring DSL), der auch die Endpunkte und eine Route enthält. Die Konfiguration muss in der Regel nicht angepasst werden, es sei denn die Namen der im Bus konfigurierten Endpunkte werden geändert. Dann müssen diese auch in der Konfiguration in FirstSpirit angepasst werden. Die Adapter-Statistics-Response-Route mit dem UxbServiceStatisticsResponseHandler-Bean sind zwingend zu benutzen, wenn das UX-Bridge-eigene Monitoring benutzt werden soll (s. Installationshandbuch: Monitoring im Auftrag).



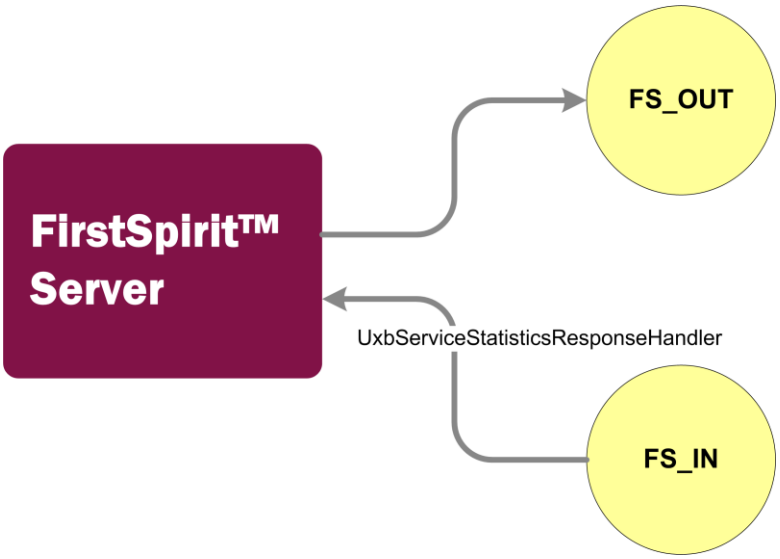
Innerhalb der Spring DSL ist ein Camel Context beschrieben, der die Routen und Endpunkte enthält:

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

<package>com.espirit.moddev.uxbridge.service</package>

<template id="producerTemplate"/>

<endpoint id="FS-OUT" uri="activemq:topic:FS_OUT"></endpoint>

<onException>

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

          <handled>

                    <constant>true</constant>

          </handled>

          <process ref="uxbExceptionProcessor" />

</onException>

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

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

          <convertBodyTo
type="com.espirit.moddev.uxbridge.api.v1.service.UXBEntity"/>

          <bean ref="UxbServiceStatisticsResponseHandler"
method="print"/>

</route>

</camelContext>


<bean id="UxbServiceStatisticsResponseHandler"
class="com.espirit.moddev.uxbridge.service.UxbServiceStatisticsRes
```

```
ponseHandler">
<constructor-arg ref="camelContext"/>
</bean>
<bean id="uxbExceptionProcessor"
class="com.espirit.moddev.uxbridge.inline.UxbExceptionProcessor"/>
```

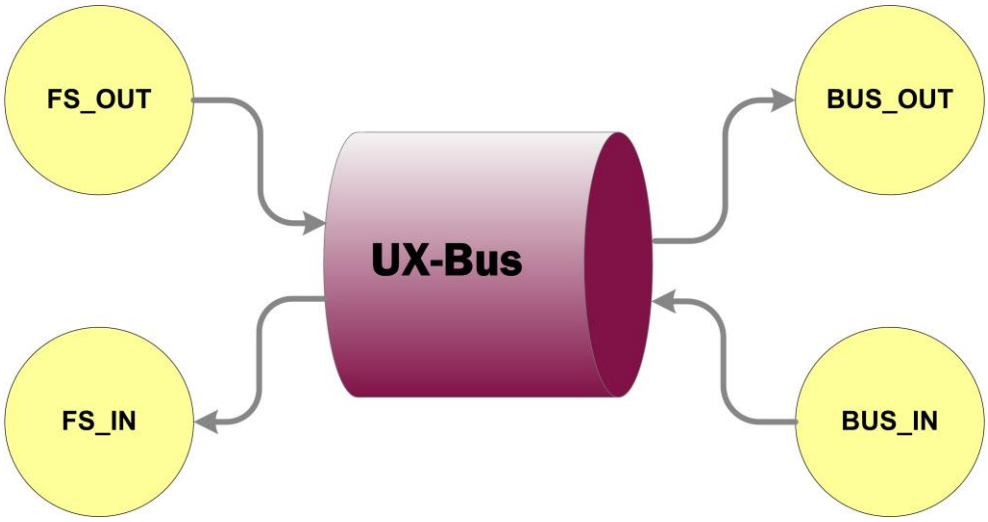
Im Beispiel gibt es einen Endpunkt mit der Id „FS_OUT“, der als Endpunkt für Nachrichten dient, die vom Service geschickt werden.

Daneben ist die Route „Adapter-Statistics-Response-Route“ definiert, die vom Endpunkt „jms:topic:FS_IN“ Nachrichten konsumiert. Die Nachrichten werden vom UXB-Service wieder in ein Objekt (UXBEntity) konvertiert und es wird danach der UxbServiceStatisticsResponseHandler auf den Objekten angewendet, so dass diese z.B. zu Timingzwecken wieder ausgewertet werden können.



2.4.2 Routing im UX-Bus

Die Spring DSL im UX-Bus enthält einen Camel Context mit vier Endpunkten, die zwei Routen bilden. Die erste Route geht vom Endpunkt von FirstSpirit zum Endpunkt des Adapters und die zweite vom Endpunkt des Adapters in entgegengesetzter Richtung zum Endpunkt des FirstSpirit-Services.




```
<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 der Standardkonfiguration kommen virtuelle Endpunkte zum Einsatz (s. http://activemq.apache.org/virtual-destinations.html). Der Vorteil von virtuellen Endpunkten ist, dass für weitere Adapter keine Modifikation am Routing vorgenommen werden muss. Die virtuellen Endpunkte folgen dem Namensschema VirtualTopic.%Ziel-Endpunkt%“. Durch die Virtualisierung können Nachrichten nicht wie bei einer Queue nur von einem Adapter gelesen werden, sondern alle entsprechenden Adapter erhalten die Nachricht.

Die erste Route schickt Nachrichten vom Endpunkt „activemq:topic:FS_OUT“ in

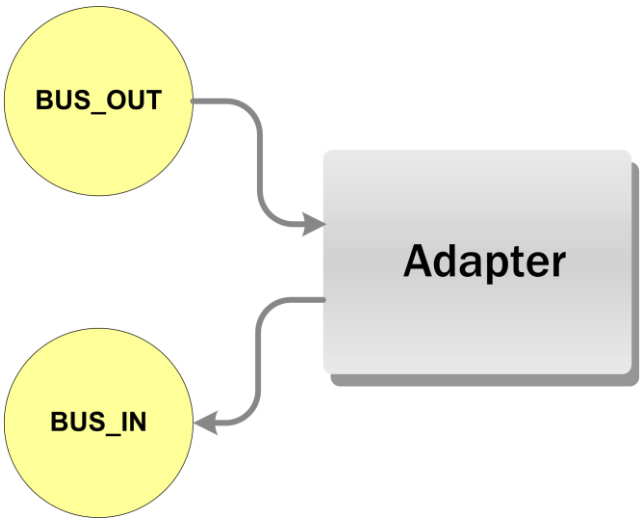
Richtung der Adapter zu dessen Endpunkt „activemq:topic:VirtualTopic.BUS_OUT“. Über XPath wird hier beispielhaft noch eine Unterscheidung für mehrere Adapter vorgenommen. „@objecttype“ bezieht sich hier auf den Header der JMS-Nachricht (s. Ausgabekanal anlegen und befüllen).

Nachrichten, die von den Adaptern aus an den Endpunkt „activemq:topic:BUS_IN“ in Richtung FirstSpirit-Service geschickt werden, werden an den Endpunkt „activemq:topic:FS_IN“ weitergeleitet.

In dieser Konfiguration sind in der Regel nur Anpassung in der Route zum Adapter vorzunehmen, wenn sich der Name des Endpunkts ändern sollte, oder besondere Routingmechanismen wie z.B. eine Fallunterscheidung für mehrere Adapter wie im Beispiel vorzunehmen ist.

2.4.3 Endpunkte im Adapter

Der Adapter kann im Gegensatz zur Umsetzung in FirstSpirit und im UX-Bus frei implementiert werden. Einzige Voraussetzung ist, dass der Adapter JMS-Nachrichten von einem Endpunkt entgegennehmen und in einem weiteren Endpunkt erzeugen kann. Zwei Beispiele von mit Camel umgesetzten Adaptern finden Sie in den „Tutorials“ unten.



3 Tutorials

In den folgenden Tutorials werden zwei Beispiele Schritt für Schritt erläutert. Diese Beispiele können in eigene Projekte übernommen oder als Anregungen für eigene Implementierungen verwendet werden.

Als Grundlage für die Beispiele diente jeweils ein frisch aufgesetztes „Mithras Energy“ Projekt, welches als Beispielprojekt jeder FirstSpirit Installation beiliegt.

Aktuelle Versionen des Quellcodes für die Beispiele finden sie unter: https://github.com/e-Spirit/uxbridge-samples

Für diese Beispiele ist grundlegendes Wissen folgender Technologien von Nutzen.

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

Außerdem ist es erforderlich, dass der UX-Bus in Betrieb und erreichbar ist. Informationen für den Betrieb des UX-Bus finden sie in der Installations-Dokumentation.

Bei den Anwendungen ist es erforderlich, die Datenbankkonfiguration an die lokalen Gegebenheiten anzupassen. Informationen, wo die jeweilige Konfigurationen zu finden sind, entnehmen Sie bitte dem jeweiligen Kapitel.

3.1 News Widget Szenario

In diesem Beispiel wird ein einfaches Widget erstellt, welches die neuesten Artikel anzeigt. Die Anzeige wird per JavaScript automatisch aktualisiert sobald neue Artikel

in dem Live-Repository hinzugefügt werden.

FirstSpirit ist das führende System, d.h. die Seiten werden statisch generiert und auf dem Server abgelegt. Das dynamische Widget wird zur Laufzeit per JavaScript in die Seite gebaut.

In der Beispiel-Anwendung wird das Widget auf der Startseite in der rechten Spalte eingebunden.

Den Quellcode für dieses Beispiel finden Sie im Github-Repository unter newsWidget.

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

3.1.1 Webanwendung

Die Webanwendung liefert nur das JavaScript als jQuery-Plugin und einen Service mit JSONP Unterstützung zum Aktualisieren der Daten. Das Grundgerüst des Widgets wird in FirstSpirit verwaltet.

Die Webanwendung wurde mit dem Web-Framework Grails in Version 2.1.0 erstellt.

3.1.1.1 Konfiguration

Alle Konfigurationsdateien liegen in dem für Grails Anwendungen typischen Ordner <grailswidget>/grails-app/config.

Wichtig an dieser Stelle sind die Dateien DataSource.groovy und Config.groovy

3.1.1.1.1 DataSource.groovy

Hier werden für die verschiedenen Environments (test, development und production) die Datenbankverbindungen konfiguriert.

3.1.1.1.2 Config.groovy

Hier wird neben den Urls für die unterschiedlichen Environments auch die Verbindung zu MongoDB konfiguriert.

3.1.1.1.3 UrlMappings.groovy

Hier wurden 2 Mappings hinzugefügt:

„/rest/v1/articles“ verweist auf die Action „list“ des ArticleRestControllers.

„/rest/v1/article/$id“ verweist auf die Action „show“ des ArticleRestControllers.

3.1.1.2 Domainklasse

In dieser Anwendung gibt es eine einzelne Domainklasse: com.espirit.moddev.examples.uxbridge.widget.Article

Grails verwendet, wie der Adapter auch, das Persistence Framework Hibernate. Daher ist es nötig, darauf zu achten, dieselben Namen für die Attribute, Tabellen und Indizes zu verwenden, die auch schon im Adapter verwendet wurden.

3.1.1.3 Rest-Controller

Im Package com.espirit.moddev.examples.uxbridge.widget findet sich der ArticleRestController. Über diesen Controller lädt das Widget die Liste der Artikel.

Der ArticleRestController bietet die zwei Methoden list und show an.

3.1.1.3.1 Methode: list

Diese Methode liefert eine bestimmte Anzahl von Artikeln im JSONP Format zurück.

3.1.1.3.2 Methode: show

Diese Methode liefert einen Artikel anhand der FirstSpirit-ID und der Sprache im JSONP Format. Wird für den übergebenen Parameter kein Artikel gefunden, liefert die Methode einen 404 Fehlercode.

3.1.1.4 Service

Der ArticleService liegt im Package com.espirit.moddev.examples.uxbridge.widget. Für dieses Beispiel wurden die beiden Methoden getLatestArticles und ellipsis implementiert.
Der ArticleService wird im ArticleRestController verwendet

3.1.1.4.1 getLatestArticles

Die Methode liefert die aktuellsten Artikel aus dem Live-Repository

3.1.1.4.2 ellipsis

Diese Methode wird verwendet um den Text für das Widget auf eine bestimmte Anzahl von Zeichen zu kürzen.

3.1.1.5 SQL und NoSQL

Im Gegensatz zu den Adaptern ist eine Anpassung des Quellcodes der Web-Anwendung in der Regel nicht nötig. Dank der Verwendung des Grails Frameworks können die Domainklassen sowohl in einer Relationalen, als auch einer NoSQL Datenbank gespeichert werden.

3.1.1.6 Starten der Beispiel Anwendung

Gestartet wird die Anwendung über die Kommandozeile:

```
grails run-app
```

Zum Starten der Anwendung mit dem MongoDB Live-Repository muss das entsprechende Environment angegeben werden

```
grails mongo run-app
```

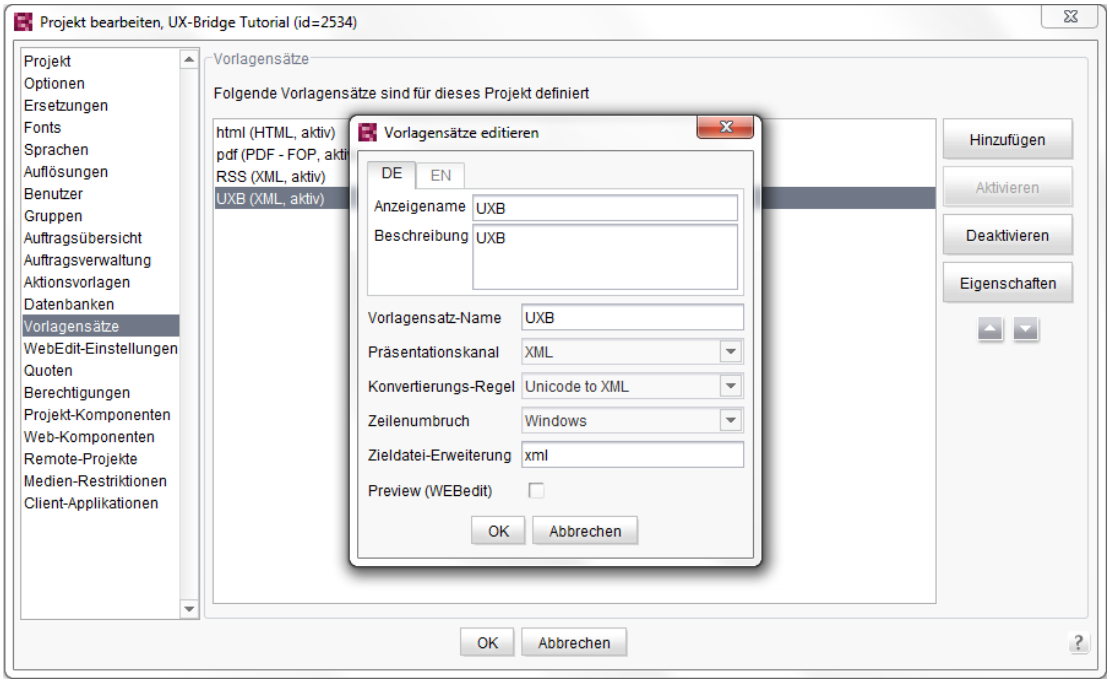
3.1.2 FirstSpirit Entwicklung

Das News Widget wird in diesem Tutorial in das Standard Projekt „Mithras Energy“ eingebaut. Deswegen importieren Sie dieses zuerst und nehmen Sie alle folgenden Änderungen in diesem Projekt vor.

Das komplett fertiggestellte Beispielprojekt wird unter dem Namen „uxbridge_tutorial_newsWidget.tar.gz“ mit ausgeliefert und kann genutzt werden, um den Templatecode und die Einstellungen zu betrachten.

3.1.2.1 Projekt-Konfiguration

Im ersten Schritt sollte in der Projekt-Konfiguration ein neuer Vorlagensatz für die UX-Bridge angelegt werden, der wie folgt zu konfigurieren ist.



Um Nachrichten an den UX-Bus zu schicken, sind im entsprechenden Template, das die Nachrichten erzeugen soll, die Felder, die im Datenmodell definiert wurden, in Form von XML auszugeben (vgl. Kapitel 2.1.3 „Ausgabekanal anlegen und

befüllen“).

3.1.2.2 Projekteinstellungen

In den Projekteinstellungen wird die Url zur Web-Applikation definiert, aus der später die passenden Artikel dynamisch nachgeladen und in dem Widget angezeigt werden. Dafür wird das Template für die Projekteinstellungen um ein Feld erweitert.

```
<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>
```

Sobald dies erfolgt ist, können die URLs für die UX-Bridge gepflegt werden. Die Base URL ist die der Grails Anwendung widgetExample, also z.B.:

http://localhost:8080/widgetExample

In den globalen Inhalten muss eine neue globale Seite, basierend auf der Vorlage „multilanguagelabel“, mit dem eindeutigen Namen „latestarticles“ angelegt werden.

Deutsch: Neueste Artikel

Englisch: Latest articles

3.1.2.3 Seitenvorlage

Die Seitenvorlagen, die die UX-Bridge nutzen sollen, werden im Beispiel um eine Eingabekomponente erweitert. Diese Eingabekomponente bewirkt, dass die Marginalspalte vergrößert wird, so dass dem einzubauenden Widget mehr Platz zur Verfügung steht. Grund dafür ist, dass die Marginalspalte in ihrer Standardbreite zu

klein ist, um das News Widget in einer angemessenen Auflösung darzustellen.

```
<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>
```

Diese Änderungen dienen des Weiteren dazu, die UX-Bridge nur auf den Seiten zu aktivieren, die diese auch einbinden.

Neben der Erweiterung des Eingabeformulars muss auch der Code in der Vorlage

angepasst werden.

Zu Beginn der Seitenvorlage muss der Body-Bereich in eine Variable ausgelagert werden, damit die Variablen die im Body gesetzt werden, schon zu Beginn der Seitenvorlage zur Verfügung stehen z.B.:

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

Die Ausgabe an der ursprünglichen Stelle des Bodies lautet dann z.B.:

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

Der Header der Seitenvorlage muss noch um folgenden Aufruf erweitert werden, die die Seitenvariablen der UX-Bridge initialisiert:

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

Der HTML-Header muss danach um folgenden Aufruf erweitert werden, um die Java-Scripte zu importieren:

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

3.1.2.4 Absatzvorlage

Das News Widget wird über eine Absatzvorlage in die Marginalspalte der gewünschten Seite eingebunden. Dazu wird zunächst eine neue Absatzvorlage mit dem Namen „uxb_widget“ wie folgt angelegt.

```
<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>


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

Die Klassen clearfix und teasermodule verwenden CSS Eigenschaften aus Mithras und sind dafür ausgelegt, im Bereich „Content right“ der Standardseitenvorlage oder der Homepage eingesetzt zu werden.

Über die im Formular angelegte Eingabekomponente Number kann definiert werden, wie viele News Einträge in dem Widget angezeigt werden sollen.

```
<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 Formatvorlage

Im Beispiel wird eine neue Formatvorlage (uxbridge_widget_head) benutzt, die den benötigten JavaScript- und CSS-Code enthält. Die Parameter, mit denen das jQuery-Plugin „uxb_widget“ initialisiert wird, sind konfigurierbar.

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


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

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

3.1.2.6 Seite anlegen

Um die UX-Bridge zu nutzen, wird ein neuer Absatz vom Typ „uxb_widget“ in eine beliebige Seite eingebaut. Ggf. muss dazu vorher der Absatz in der Seitenvorlage für die Marginalspalte noch erlaubt werden.

3.1.2.7 Tabelle und Tabellenvorlage (XML)

Basierend auf der im Schema bereits definierten Tabelle Press_Releases sollte dann eine Tabellenvorlage angelegt werden, die das XML erzeugt, das an den UXB-Service weitergereicht wird:

```
<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>
```

Die Kind-Elemente im uxb_content-Tag sind gleichzeitig die Inhalts-Felder, die vom Adapter in das angeschlossene Content-Repository geschrieben werden und daher auch bei der Erstellung der Datenstruktur berücksichtigt werden sollten.

Dieses Codebeispiel stellt dabei nur die Struktur dieser Tabellenvorlage dar. Den kompletten Code finden Sie im Beispiel-Projekt in der Tabellenvorlage mit dem Referenznamen „Products.press_releases“.

3.1.2.8 Deployment

Im Beispielprojekt ist es möglich, die Generierung der JMS-Nachrichten und damit auch die Einträge in den angebundenen Content-Repositories automatisch über einen Workflow direkt aus der Datenquelle zu starten. Ebenso ist es möglich, über einen weiteren Workflow Objekte in FirstSpirit und in den angebundenen Content-Repositories direkt aus den Datenquellen heraus zu löschen. Die Workflows nutzen dazu neben Skripten auch Tabellenabfragen und Aufträge, die zunächst zu konfigurieren sind.

3.1.2.8.1 Tabellenabfragen anlegen

Es müssen Tabellenabfragen für das Erzeugen eines Datensatzes und allen Datensätzen der News-Tabelle angelegt werden. Die Abfrage für einen Datensatz muss dabei noch eine Einschränkung auf die „fs_id“ Spalte erhalten mit dem neu anzulegenden Parameter „Id“.

Den kompletten Code finden Sie im Beispiel-Projekt in der Tabellenabfrage mit dem Referenznamen „Products.pressdetailfilter“.

3.1.2.8.2 Auftrag anlegen

Es muss ein neuer Auftrag angelegt werden, der neben der Generierung der JMS-Nachrichten für den UXB-Service auch die Generierung und das Deployment der Übersichtsseiten übernimmt. Dazu ist dem Auftrag zunächst eine Generierungsaktion hinzuzufügen, die die Übersichtsseiten erzeugt. Das Delta-Deployment erweitert diese Aktion zur Laufzeit um die Detailseite des aktuell zu generierenden Datensatzes. Danach muss eine Skriptaktion erfolgen, die die UX-

Bridge aktiviert:

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

In der danach anzulegenden Generierung sollte dann eine Teilgenerierung erfolgen. Seite und Datensatz werden später durch das Delta-Deployment automatisch eingetragen, so dass nur die gewünschte JMS-Nachricht erzeugt wird. Danach kann das Deployment der Webseiten wie gewohnt erfolgen.

Soll die Durchlaufzeit für die Nachrichten im Bus bis zur Veröffentlichung auf der Webseite gemessen werden, so muss als letztes noch die Aktion „UX-Bridge Statistics Report“ hinzugefügt werden, die folgendes Skript enthält:.

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

Der Service wartet maximal den in der Servicekonfiguration der UX-Bridge definierten Zeitraum, bis die Antworten der Adapter ausgewertet werden. Kommt in diesem Zeitfenster keine Antwort, so gilt die Nachricht als fehlerhaft zugestellt. Da die Antwortzeiten je nach Nachricht und System variieren können, ist dieser Wert konfigurierbar.

3.1.2.8.3 Workflow-Skripte importieren

Im nächsten Schritt müssen die benötigten Workflow-Skripte importiert werden:

- uxb_content_release_init
- uxb_content_release_script
- uxb_content_delete_init
- uxb_content_delete_script

Die Init-Skripte initialisieren in diesem Fall Variablen und schreiben diese in die Session, so dass die Methoden des UXB-Moduls, die in den anderen Skripten aufgerufen werden, auf diese zugreifen können.

Konfiguriert werden müssen in uxb_content_release_init folgende Parameter:

Parameter	Beispielwert	Beschreibung
detail_page	pressreleasesdetails	Seitenreferenz der Seite die die JMS-Nachrichten enthält
query_uid	Products.pressdetailsfilter	Tabellenabfrage die alle Datensätze erzeugt
single_query_uid	Products.pressdetailfilter	Tabellenabfrage die als Parameter die Id des Datensatzes erhält der erzeugt werden soll
query_param	Id	Parametername der Tabellenabfrage
schedule_name	UX-Bridge	Name des Auftrags der die JMS-Nachrichten erzeugen soll
scheduler_uxb_generate	UX-Bridge Generate	Name der Generierungsaktion der JMS Nachrichten
scheduler_generate	Generate	Name der Generierungsaktion für die HTML-Seiten

Das Skript „uxb_content_release_script“ startet danach den zuvor konfigurierten Auftrag und führt die definierte Transition aus.

Konfiguriert werden müssen in uxb_content_delete_init folgende Parameter:

Parameter	Beispielwert	Beschreibung
Destinations	postgres	Name der Content Repositories aus denen der Datensatz gelöscht werden soll

transition_name	release	Name der Transition im Workflow (s. Workflow) die nach dem content_delete_script geschaltet werden soll
object_type	news	Typ des Objekts der gelöscht werden soll

Innerhalb des nachfolgenden Skripts „uxb_content_delete_script“ wird der selektierte Datensatz in FirstSpirit gelöscht und eine Nachricht über den UXB-Service und den Bus an das angeschlossene Content Repository geschickt, die dort die Löschaktion auslöst.

3.1.2.8.4 Workflows importieren

Um dem Redakteur eine einfache Möglichkeit zu bieten, die zuvor definierten Skripte auf einem Datensatz auszuführen, werden die beiden Workflows „uxb_content_release“ und „uxb_content_delete“ aus dem Demoprojekt verwendet. Die Workflows führen dabei die gewünschten Operationen (Freigeben, Löschen) sowohl in FirstSpirit, als auch über die UX-Bridge im konfigurierten Content-Repository durch.

3.1.2.8.5 Komplettabgleich

In dem FirstSpirit Beispielprojekt wurde der Komplettabgleich in dem Auftrag „UX-Bridge Full Deployment“ umgesetzt.

Hinweise zu dem Vorgehen des Komplettabgleich finden Sie im Kapitel 2.1.4.2 „Komplettabgleich“ ab Seite 14.

3.1.3 Adapter

Dieses Beispiel enthält zwei Adapter. Einen für eine relationale Datenbank (PostgreSQL) und einen für eine NoSql Datenbank (MongoDB).

Unter https://github.com/e-Spirit/uxbridge-samples/newsWidget/adapter befindet sich neben den Projekten für die beiden Adapter (hibernate, mongodb) auch ein drittes Projekt, das Java-Klassen enthält, die in beiden Adaptern verwendet werden.

3.1.3.1 JAXB

Zum Verarbeiten des Austauschformats wird JAXB verwendet. Die entsprechenden Klassen befinden sich im Projekt https://github.com/e-Spirit/uxbridge-samples/newsWidget/adapter/base im Package com.espirit.moddev.uxbridge.entity.

JAXB ermöglicht das einfache Arbeiten mit Java-Objekten, ohne sich um das Parsen des XML Gedanken machen zu müssen. Ähnlich wie bei JPA wird hier mit Annotationen gearbeitet.

```
@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 für das Datumsformat

Datumsangaben werden im FirstSpirit Ausgabekanal nach dem Format "yyyy-MM-dd'T'HH:mm:ssZ" formatiert. Damit dieses Format in den JAXB Klassen eingelesen werden kann, wurde die Klasse DateAdapter implementiert. Diese Klasse befindet sich im Package com.espirit.moddev.examples.uxbridge.widget.entity.type.

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

3.1.3.1.2 UXBEntity und UXBContent

Die beiden Klassen implementieren den von der UX-Bridge vorgegebenen Teil (UXBEntity) und den projektspezifischen Teil (UXBContent) des Austauschformats.

3.1.3.2 Relationale Datenbank

Der Adapter für relationale Datenbanken wurde mit Hilfe von Hibernate umgesetzt. In diesem Beispiel wird zwar PostgreSQL verwendet, der Adapter sollte aber ohne größeren Aufwand auch mit anderen, von Hibernate unterstützten Datenbanken funktionieren.

3.1.3.2.1 Domainklasse: Article

Die Domain Klasse Article befindet sich im Projekt widgetExample/adapter/base im Package com.espirit.moddev.examples.uxbridge.widget.

Die Klasse verfügt über eine generierte ID:

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

Das Attribut id wird verwendet, damit die Domainklasse kompatibel zur Grails Implementierung in der Web-Anwendung ist.

Um komplexe Datenbankstrukturen zu verhindern, wurde in diesem Beispiel pro Sprache ein Objekt erzeugt. D.h. die FirstSpirit-ID ist in diesem Kontext nicht mehr eindeutig. Der Zugriff auf die Daten erfolgt daher über die FirstSpirit-ID (aid) und die Sprache (language).

Sinnvoll wäre an dieser Stelle der Einsatz eines zusammengesetzten Primärschlüssels. In diesem Beispiel wurde aber auf Grund der Komplexität bewusst auf diese Möglichkeit verzichtet.

Zu beachten ist, dass sich nach einem Löschen und erneutem Hinzufügen eines Artikels in das Live-Repository die ID ändert. Daher ist es erforderlich, für den Zugriff immer die FirstSpirit-ID und die Sprache zu verwenden.

3.1.3.2.2 ArticleHandler

Der Zugriff auf die Datenbank erfolgt im ArticleHandler (Package com.espirit.moddev.examples.uxbridge.widget.jpa). Er kümmert sich um das

Entgegennehmen und Bearbeiten der Daten. In dem Beispiel wurde im Handler für jedes der unterstützten Kommandos eine eigene Methode implementiert.

3.1.3.2.2.1 Command add

Speichern oder Aktualisieren einer Pressemitteilung im Live-Repository.

3.1.3.2.2.2 Command delete

Löschen einer Pressemitteilung im Live-Repository.

3.1.3.2.2.3 Command cleanup

Löschen aller Pressemitteilungen, die älter sind, als das gegebene Datum.

3.1.3.2.3 Konfiguration

Die Konfiguration für diesen Adapter befindet sich in der Datei WEB-INF/applicationContext.xml. In dieser Spring-Xml Datei werden neben der Datenbank auch das JMS, der ArticleHandler und die Camel-Routen konfiguriert.

3.1.3.2.3.1 CamelContext

In diesem Context wird konfiguriert, welche Nachrichten für diesen Adapter interessant sind und von ihm verarbeitet werden.

```
<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>
```

Eine detaillierte Erläuterung zur Erstellung der Rückmeldung befindet sich im Kapitel

3.5 „Verwendung der Camel Komponente zur Antwortgenerierung“.

3.1.3.2.3.1.1 Empfang von Nachrichten

Über den From-Tag (<from uri=" activemq:Consumer.newsWidgetHibernate.VirtualTopic.BUS_OUT" />) wird die URI konfiguriert, über die Nachrichten eingelesen werden. An dieser Stelle kommt ein virtueller Endpunkt zum Einsatz (s. http://activemq.apache.org/virtual-destinations.html). Der Vorteil von virtuellen Endpunkten ist, dass für weitere Adapter keine Modifikation am Routing vorgenommen werden muss. Neue virtuelle Endpunkte müssen lediglich dem Namensschema „Consumer.%beliebiger Adaptername%.VirtualTopic.%Quell-Endpunkt%“ folgen. Durch die Virtualisierung können Nachrichten nicht wie bei einer Queue nur von einem Adapter gelesen werden, sondern alle entsprechenden Adapter erhalten die Nachricht.

Soll also zum Beispiel auch der neue Adapter „myAdapter“ Nachrichten konsumieren, die an den Endpunkt FS_OUT ausgeliefert werden, so könnte ein möglicher Endpunkt wie folgt aussehen:

activemq:Consumer.myAdapter.VirtualTopic.BUS_OUT

3.1.3.2.3.1.2 Filtern für das Live-Repository

Mittels des XPath-Ausdrucks (//uxb_entity[contains(@destinations, 'postgres')]) werden die Nachrichten gefiltert, die in dieses Live-Repository geschrieben werden sollen.

3.1.3.2.3.1.3 Filtern des Objekt-Types

Mit diesem Ausdruck //uxb_entity[@objectType = 'news'] werden die Nachrichten auf Objekte des Typs news beschränkt.

3.1.3.2.3.1.4 Hinzufügen und Löschen von Artikeln

Mit diesen Ausdrücken wird nach dem entsprechenden Kommando gefiltert. Bei „//uxb_entity[@command = 'add']“ handelt es sich um ein Hinzufügen zu dem Repository und bei „//uxb_entity[@command = 'delete']“ um Löschen aus dem Repository.

Vor dem eigentlichen Methoden-Aufruf wird das Austauschformat mittels JAXB und der Camel-Anweisung <convertBodyTo

type="com.espirit.moddev.examples.uxbridge.widget.entity.UXBEntity"/>
umgewandelt. Der Aufruf der entsprechenden Methode im ArticleHandler erfolgt dann mit einem Objekt vom Typ UXBEntity.

3.1.3.2.3.2 ArticleHandler

Der ArticleHandler ist der Teil des Adapters, der die Kommandos verarbeitet und die Artikel in das Repository schreibt oder sie daraus löscht.

Der ArticleHandler benötigt zum einen die EntityManagerFactory für den Zugriff auf die Datenbank, zum anderen den CamelContext und den Namen der Routen, auf denen Nachrichten zurück an FirstSpirit gesendet werden können.

3.1.3.3 MongoDB

Für das NoSQL Live-Repository wurde ausschließlich der MongoDB-Datenbanktreiber verwendet. Auf den Einsatz eines Persistence Frameworks wurde bewusst verzichtet, da in dem Adapter die DB-Struktur der Web-Anwendung nachgebildet werden musste.

3.1.3.3.1 Domainklasse Article

Der MongoDB-Adapter verwendet dieselbe Domainklasse, die auch von dem Hibernate-Adapter verwendet wird. Die JPA Annotationen werden dabei nicht berücksichtigt.

3.1.3.3.1.1 Id Generierung

In der Web-Anwendung wird Grails GORM für den Datenbankzugriff verwendet. Damit der Adapter die identische Datenbankstruktur verwendet, musste eine Helfer-Methode generateIdentifier eingeführt werden. In dieser Methode werden über eine extra Collection (http://www.mongodb.org/display/DOCS/Collections) IDs verwaltet.

3.1.3.3.2 ArticleHandler

Das Vorgehen im ArticleHandler unterscheidet sich nicht von dem Vorgehen des Hibernate-ArticleHandlers (Siehe hierzu auch das Kapitel 3.1.3.2.2 „ArticleHandler“).

3.1.3.3.3 Konfiguration

Die Konfiguration unterscheidet sich nur geringfügig von der Konfiguration des Hibernate-Adapters.

Der Destination-Filter filtert Nachrichten für die Destination mongodb. Die Parameter für die Verbindung zur Datenbank werden direkt an den ArticleHandler übergeben.

3.1.3.4 Starten der Beispiel Adapter

Mit dem folgendem Aufruf kann die API in das lokale Maven Repository geladen werden:

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

Eine Beispielumsetzung kann wie folgt aussehen:

```
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
```

Die Beispiel-Adapter können über die Kommandozeile gebaut werden:

```
mvn package
```

Die daraus resultierende War-Datei kann auf einen beliebigen ServletContainer (Tomcat, Jetty usw.) deployed werden.

Alternativ können die Adapter auch über die Kommandozeile gestartet werden:

```
mvn tomcat7:run
```

Um den Port des dadurch gestarteten Tomcats anzupassen, muss die Datei pom.xml im Verzeichnis des jeweiligen Adapters angepasst werden.

3.1.3.5 Enthaltene Tests

In dem Beispielprojekt sind Unit- und Integrations-Tests enthalten. Für die Tests werden eine In-Memory Datenbank und jMockMongo (https://github.com/thiloplanz/jmockmongo) verwendet. Damit die Tests für den MongoDB-Adapter gestartet werden können, muss das jMockMongo Jar in das lokale Repository importiert werden oder folgendes Maven Repository verwendet

werden:

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

Die dependency muss dann wie folgt aussehen:

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

Die Integrations-Tests können mit folgendem Aufruf gestartet werden: **mvn verify -Pintegration-test**

3.2 News Widget Szenario ohne Programmierung

Wie im vorangegangenen Beispiel wird auch in diesem Beispiel ein einfaches Widget erstellt, welches die neuesten Artikel anzeigt. Der Unterschied liegt in der Umsetzung des Adapters. Dieser wurde ohne Programmierung, nur durch den Einsatz von Camel realisiert.

Es kann auf die Benutzung des ArticleHandlers verzichtet werden. Die Funktionen des ArticleHandlers werden dabei durch eine Konfiguration des CamelContextes ersetzt.

Die meisten Punkte sind identisch zu dem vorherigen Beispiel. Deshalb werden im folgendem nur die Änderungen beschrieben, die notwendig sind, um das Beispiel ohne Programmierung umzusetzen.

3.2.1 CamelContext

In Teilen ist der CamelContext mit dem im vorherigen Beispiel erläutertem identisch. Im folgendem wird trotzdem der gesamte Context erläutert.

```
<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>
```

Der CamelContext beginnt mit einem Exception Handling. Dazu wird innerhalb des onException-Tags definiert, welche Exceptions und wie diese verarbeitet werden. In vorliegenden Fall ist lediglich eine java.io.Exception angegeben. Es sind jedoch auch mehrere Exceptions gleichzeitig möglich.

Ist der handled-Tag auf true gesetzt, wird die Exception behandelt. In diesem Fall wird die Exception nicht weiter geworfen und es kommt nicht zu einem Abbruch des gesamten Vorgangs. Dies entspricht einem try-catch Block um alle Routen. Sollen Exceptions in bestimmten Bereichen gesondert behandelt werden, ist ein expliziter

try-catch Block für diese Bereiche möglich.

Abschließend wird angegeben, was im Fall einer Exception getan werden soll. In diesem Fall wird eine Nachricht an BUS_IN gesendet. Der genaue Aufbau wird im Kapitel 3.5 „Verwendung der Camel Komponente zur Antwortgenerierung“ beschrieben.

Innerhalb der Route wird mittels filter und xpath das übergebene XML analysiert und die entsprechenden Aufrufe getätigt.

Um mit einer Mongo Datenbank kommunizieren zu können, muss ein DBObject erzeugt werden. Die Erzeugung erfolgt mit <camel:convertBodyTo type="com.mongodb.DBObject" />. Als Übergabeparameter wird ein JSON Objekt als String erwartet. Hierzu wird innerhalb des XML Dokuments ein JSON Objekt übergeben. Mit text() wird der Inhalt eines XML-Tags, in diesem Fall das JSON Objekt, ausgelesen. Sind innerhalb des JSON Objekts Daten enthalten, die durch den XML Parser bereits interpretiert werden, muss das JSON durch einen CDATA Bereich eingeschlossen werden, um die ungewünschte Interpretation zu verhindern. Dieser Bereich muss vor der Erstellung des DBObjects entfernt werden. Das erfolgt mit <language language="groovy"><![CDATA[request.getBody().substring(request.getBody().indexOf("<![CDATA[")+9,request.getBody().lastIndexOf("]]]><![CDATA[>"))]]></language>.

Um das DBObject an die Mongo Datenbank zu übermitteln, wird <to uri="mongodb:myDb?database=newsWidget&collection=article&operation=save" /> aufgerufen. Als Parameter werden die Datenbank, die Collection und die Operation mit übergeben.

Im delete und cleanup Bereich werden die JSON Objekte mittels groovy erstellt. Die benötigten Informationen (uuid,language,createTime) werden aus dem uxb_entity Tag des XML-Dokumentes geparst und an die entsprechende Stelle im JSON Objekt gesetzt. Somit ist es nicht notwendig ein JSON Objekt innerhalb des XML Dokumentes zu übergeben.

3.2.2 Anpassungen in FirstSpirit

Um das News Widget Szenario ohne Programmierung verwenden zu können, ist noch eine kleinere Anpassung innerhalb von FirstSpirit notwendig. Wie bereits im Kapitel zuvor beschrieben, ist es notwendig, die Informationen im JSON Format verschachtelt in einem XML Dokument zu übergeben.

3.2.2.1 Content hinzufügen

Zum Hinzufügen von Content muss der UXB-Kanal des Datenbank-Schemas Products.press_release angepasst werden. Der UXB-Kanal muss wie folgt aussehen:

```
<?xml version="1.0" encoding="UTF-8" ?>
$CMS_SET(_id)$$CMS_VALUE(#row.id)$$CMS_VALUE(#global.language.hash
Code())$$CMS_END_SET$
<uxb_entity uuid="$CMS_VALUE(#row.id)$"
language="$CMS_VALUE(#global.language)$"
destinations="postgres,mongodb" command="add" objectType="news">
<uxb_content><![CDATA[
{
"_id":$CMS_VALUE(_id)$,
"aid":$CMS_VALUE(#row.id)$,
"language":"$CMS_VALUE(#global.language)$",
"url":"$CMS_REF(#global.node, contentId:#row.getId(),abs:1,
templateSet:"html")$",
$CMS_IF(#global.preview)$"lastmodified":$CMS_VALUE(#global.now.get
TimeInMillis())$,
$CMS_ELSE$
"lastmodified":$CMS_VALUE(#global.getScheduleContext().getStartTim
e().getTimeInMillis())$,
$CMS_END_IF$
$CMS_IF(!cs_date.isEmpty)$"date":{"$date":"$CMS_VALUE(cs_date.form
at("yyyy-MM-dd'T'HH:mm:ss'Z'"))$"},$CMS_END_IF$
$CMS_IF(!cs_headline.isEmpty)$"title":"$CMS_VALUE(cs_headline.conv
ert2)$",$CMS_END_IF$
$CMS_IF(!cs_subheadline.isEmpty)$"subHeadline":"$CMS_VALUE(cs_subh
eadline.convert2)$",$CMS_END_IF$
$CMS_IF(!cs_teaser.isEmpty)$"teaser":"$CMS_VALUE(cs_teaser.convert
2)$",$CMS_END_IF$
$CMS_IF(!cs_content.isEmpty)$"content":"$CMS_FOR(section,
cs_content)$$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>
```

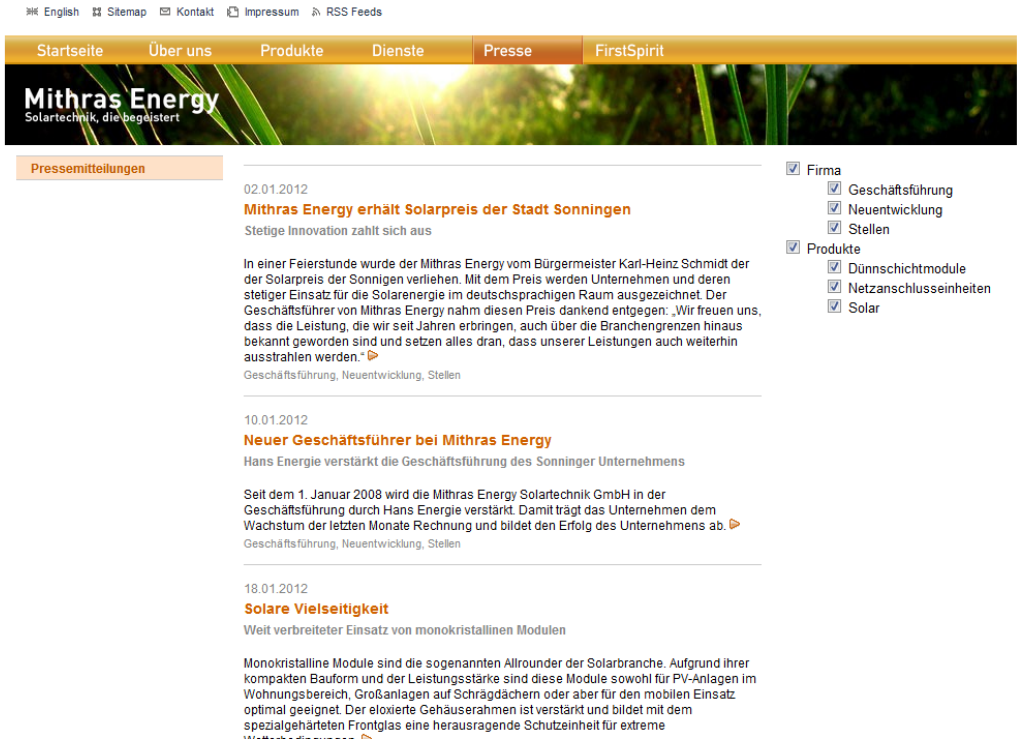
Wie bereits beschrieben, wird ein XML Dokument erzeugt, in dem ein JSON Objekt eingebettet ist.

3.3 News Drill-Down Szenario

In diesem Beispiel wird eine Übersicht von Pressemitteilungen erzeugt, die über eine Drill-Down-Funktion nach Kategorien gefiltert werden können.

Die Web-Anwendung ist dabei das führende System. D.h. die Drill-Down-Funktion und die Übersichtsseite werden dynamisch erstellt, die Detailseiten und die übrigen Seiten werden statisch generiert. Header und Footer werden auf der Übersichtsseite als HTML-Fragmente eingebunden. Diese Fragmente werden ebenfalls von FirstSpirit generiert.

Die News-Artikel, Kategorien und Meta-Kategorien werden mithilfe der UX-Bridge in ein Content-Repository geschrieben, auf welches die WebApp Zugriff hat. Diese Implementierung ist für das Beispiel einfach gehalten und nicht performance-optimiert, da bei jedem Update einer News sowohl auf die Kategorie, als auch auf die Meta-Kategorie zugegriffen wird und diese ggf. aktualisiert wird. In einem realen Adapter würde man dies natürlich optimieren und Kategorien und Meta-Kategorien nur einmal auslesen und nur bei Änderungen an den Kategorien ein Update vornehmen. Alle Kategorien und Meta-Kategorien werden in der WebApp in einem Drill-Down-Menü angezeigt, wo die Kategorien, deren News angezeigt werden sollen, mithilfe von Checkboxen markiert werden können. Bei jedem An- und Abwählen einer Checkbox wird eine AJAX-Anfrage gesendet. Das zurückgelieferte HTML wird in den Bereich der Newsliste auf der Seite eingebaut. Es wurde eine Pagination realisiert, um die Übersichtlichkeit zu gewährleisten. Diese nutzt ebenfalls AJAX, denn die Anzahl der aufzulistenden News variiert mit den gewählten Kategorien.



Die Beispielanwendung newsExample besteht aus den Komponenten Adapter (Hibernate), der Web-Anwendung (Grails) und dem FirstSpirit Beispielprojekt.

3.3.1 WebApp Entwicklung

Die Web-Anwendung wurde mit dem Web-Framework Grails in Version 2.1.0 erstellt.

3.3.1.1 Konfiguration

Alle Konfigurationsdateien liegen in dem für Grails-Anwendungen typischen Ordner <newsExample>/grails-app/conf.

Wichtig an dieser Stelle sind die Dateien DataSource.groovy und Config.groovy.

3.3.1.1.1 DataSource.groovy

Hier werden für die verschiedenen Environments (test, development und production) die Datenbankverbindungen konfiguriert.

3.3.1.1.2 Config.groovy

Hier werden die Urls für die aus FirstSpirit generierte Navigation definiert.

3.3.1.2 Domainklassen

Erstellen Sie drei Domainklassen mit den Namen News, Category und MetaCategory. Grails verwendet, wie der Adapter auch, das Persistence Framework Hibernate. Daher ist es notwendig, darauf zu achten, dieselben Namen für die Attribute, Tabellen und Indizes zu verwenden, die auch schon im Adapter verwendet wurden.

3.3.1.3 Rest-Controller

Erstellen Sie den passenden Controller zu der Domainklasse News und implementieren Sie die Methode „list“. Über diese Methode lädt die Web-Anwendung die Liste der Artikel.

3.3.1.3.1 Methode: list

Diese Methode wird genutzt um die gleichnamige gsp zu rendern.

3.3.1.3.2 Methode: listNews

Diese Methode rendert das gsp-Template „newsListing“ für eine bestimmte Liste von News, die vom FilterService geholt werden.

3.3.1.3.3 Methode: drilldown

Diese Methode rendert das gleichnamige Template, welches das Drill-Down Menü und das dafür notwendige JavaScript bereitstellt.

Das Drill-Down Menü wird beim Seitenaufruf direkt in die Seite hineingerendert.

Das JavaScript darin nutzt jQuery und verwaltet das an- und abhaken der Checkboxen für die einzelnen Kategorien und Metakategorien.

Bei jedem Klick einer Checkbox wird eine AJAX-Anfrage mit den aktuell ausgewählten Kategorien an die Methode „listNews“ des Controllers gesendet. Das zurückgelieferte HTML wird dann in das dafür vorgesehene div auf der Newsübersichtsseite eingefügt. Damit die Liste der News übersichtlich bleibt, wird

mit einer Pagination gearbeitet, welche ebenfalls dynamisch über AJAX-Abfragen die richtigen Seiten mit den richtigen Artikeln nachlädt.

3.3.1.4 Service

3.3.1.4.1 FilterService

Dieser Service stellt Methoden zum Holen von News nach ihren Kategorien zur Verfügung.

3.3.1.4.1.1 filter

Diese Methode liefert eine Map mit folgenden Schlüsseln zurück:

newsInstanceList: Eine Liste der News in den angefragten Kategorien

newsInstanceTotal: Die Gesamtanzahl der News in den angefragten Kategorien (wird für Pagination benötigt)

msg: Wenn Kategorien anhand einer ID nicht gefunden werden, wird der String „noCategory“ zurückgeliefert, der vom Controller genutzt wird, um eine Nachricht hierüber anzuzeigen

Mithilfe des Parameters „categories“, können alle Kategorien angegeben werden, die angezeigt werden sollen. Man übergibt diesem einen String mit dem Format „cat_1cat_2_cat_4“ um beispielsweise die Kategorien mit den IDs 1, 2 und 4 anzuzeigen. Wenn der String „all“ enthält, werden alle Kategorien zurückgegeben.

Die Parameter „max“ und „offset“ werden benötigt, um Pagination nutzen zu können.

3.3.1.4.1.2 filterForCategory

Diese Methode liefert alle News einer gegebenen Kategorie zurück. Sie wird von der filter-Methode für jede einzelne Kategorie aufgerufen.

3.3.1.4.2 RenderService

Dieser Service stellt eine Methode zum Rendern von HTML zur Verfügung.

3.3.1.4.2.1 renderHtml

Man übergibt der Methode die Url. Es wird ein http-Request ausgeführt, der den HTML-Schnipsel holt. Der korrekt formatierte HTML-Schnipsel wird anschließend zurückgegeben.

3.3.1.5 RenderTagLib

Diese TagLib stellt 3 Tags bereit, um den Header, den Footer und die linke Navigationsspalte zu rendern. Diese Tags werden in der main.gsp genutzt.

3.3.1.6 Starten der Beispielanwendung

Gestartet wird die Anwendung über die Kommandozeile:

```
grails run-app
```

3.3.1.7 Übersichtsseite als Grails-App

Sobald die Anwendung erfolgreich gestartet wurde, kann die News-Übersichtsseite unter folgender URL aufgerufen werden:

http://localhost:8080/newsDrilldown/

Die Links zu den News-Artikeln auf der dynamischen Übersichtsseite verweisen auf die statisch generierten News-Detailseiten. So kann eine hohe Dynamik der Webseite erreicht werden, ohne Performance einbüßen zu müssen.

3.3.2 FirstSpirit Entwicklung

Das News Szenario wird in diesem Tutorial in das Standard Projekt Mithras Energy eingebaut. Deswegen importieren Sie dieses zuerst und nehmen Sie alle folgenden Änderungen in diesem Projekt vor.

Das komplett fertiggestellte FirstSpirit-Beispielprojekt wird unter dem Namen „uxbridge_tutorial_newsDrilldown.tar.gz“ mit ausgeliefert und kann genutzt werden um den Templatecode und die Einstellungen zu betrachten.

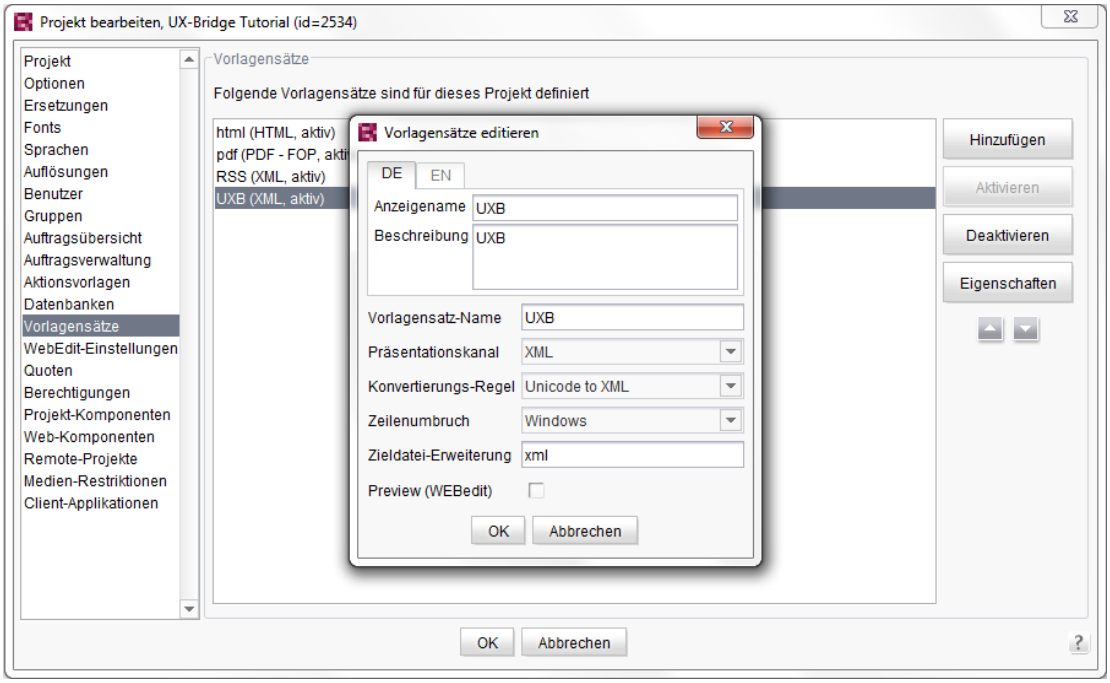
3.3.2.1 Server-Konfiguration

Im ersten Schritt sollte eine neue Konvertierungs-Regel in den Server-Eigenschaften

angelegt werden. Die entsprechende Regel sollte zuvor als Textdatei angelegt werden.

3.3.2.2 Projekt-Konfiguration

In der Projekt-Konfiguration sollte ein neuer Vorlagensatz für die UX-Bridge angelegt werden, der wie folgt zu konfigurieren ist.



Um Nachrichten an den UX-Bus zu schicken, sind im entsprechenden Template, das die Nachrichten erzeugen soll, die Felder, die im Datenmodell definiert wurden, in Form von XML auszugeben (vgl. Kapitel 2.1.3 „Ausgabekanal anlegen und befüllen“).

3.3.2.3 Absatzvorlagen

Erstellen Sie zunächst vier neue Absatzvorlagen mit den Namen „navigation_header“, „navigation_footer“, „navigation_left“ und „navigation_css“ und füllen Sie den HTML-Ausgabe Kanal mit den benötigten HTML- und CSS-Fragmenten der Mithras Energy Navigation. Diese werden nachher separat ausgegeben und in die Web-App eingebaut.

In dem Beispiel-Projekt finden Sie diese in dem Ordner „Header / Footer“ in den Absatzvorlagen.

3.3.2.4 Seitenvorlage

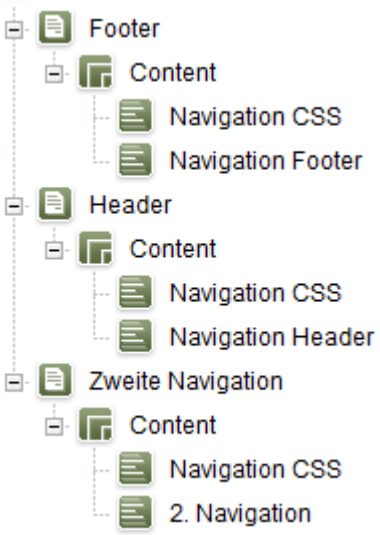
Legen Sie eine neue Seitenvorlage an und fügen Sie Ihre zuvor erstellten Absatzvorlagen zu den erlaubten Inhaltsbereichen im Register Eigenschaften hinzu. Schlussendlich editieren Sie Ihren HTML-Ausgabekanal wie folgt:

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

Achten Sie darauf, dass Sie in Ihrer Seitenvorlage kein HTML Grundgerüst verwenden!

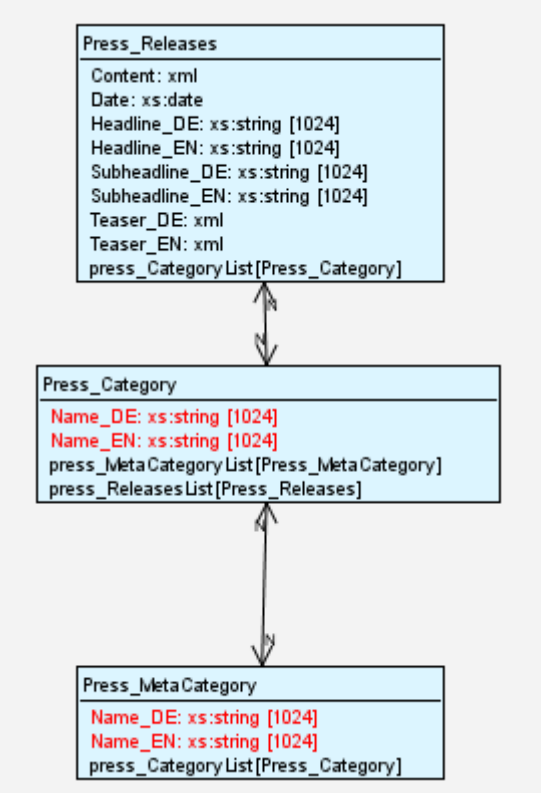
3.3.2.5 Seiten anlegen

Legen Sie nun auf Basis der zuvor erstellen Seitenvorlage drei neue Seiten in Ihrer Inhaltsverwaltung an und fügen Sie folgende Absatzvorlagen hinzu.



3.3.2.6 Tabelle und Tabellenvorlage (XML)

Im Schema muss zunächst die Datenstruktur für die News, die Kategorien und die Metakategorien definiert werden, sofern nicht schon vorhanden.



In der Tabelle „Press_Releases“ wird der allgemeine Inhalt der Pressemitteilung definiert. Dazu gehören unter anderem eine Überschrift, der Text und das Datum. Über eine n:m Beziehung wird auf die Tabelle „Press_Category“ referenziert, in der sich der Name einer Kategorie speichern lässt. Über eine weitere m:n Beziehung können einer Kategorie mehrere Metakategorien hinzugefügt werden.

Basierend auf der News-Tabelle sollte nach folgendem Schema dann eine Tabellenvorlage angelegt werden, die das XML erzeugt, das an den UXB-Service weitergereicht wird.

```
<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>
```

Die Kind-Elemente im uxb_content-Tag sind gleichzeitig die Inhalts-Felder, die vom Adapter in das angeschlossene Content-Repository geschrieben und daher auch bei der Erstellung der Datenstruktur berücksichtigt werden sollen.

Legen Sie neben der News-Tabelle auch zwei Tabellenvorlagen für die Kategorie und Metakategorie an und befüllen Sie diese anschließend in Ihren Datenquellen. In dem Beispiel Projekt finden Sie diese im Schema „Products“ mit den Referenznamen „Products.press_category“ und „Products.press_metacategory“.

3.3.2.7 Deployment

Im Beispielprojekt ist es möglich, die Generierung der JMS-Nachrichten und damit auch der Einträge in den angebundenen Content-Repositories automatisch über einen Workflow direkt aus der Datenquelle zu starten. Ebenso ist es möglich über einen weiteren Workflow Objekte in FirstSpirit und in den angebundenen Content-Repositories direkt aus den Datenquellen heraus zu löschen. Die Workflows nutzen dazu neben Skripten auch Tabellenabfragen und Aufträge, die zunächst zu konfigurieren sind.

3.3.2.7.1 Tabellenabfragen anlegen

Es müssen Tabellenabfragen für das Erzeugen von einem Datensatz und allen Datensätzen der News-Tabelle angelegt werden. Die Abfrage für einen Datensatz muss dabei noch eine Einschränkung auf die „fs_id“ Spalte mit dem neu anzulegenden Parameter „Id“ erhalten.

3.3.2.7.2 Auftrag anlegen

Es muss ein neuer Auftrag angelegt werden, der neben der Generierung der JMS-Nachrichten für den UXB-Service auch die Generierung und das Deployment der Übersichtsseiten übernimmt. Dazu ist dem Auftrag zunächst eine Generierungsaktion hinzuzufügen, die die Übersichtsseiten erzeugt. Das Delta-Deployment erweitert diese Aktion zur Laufzeit um die Detailseite des aktuell zu generierenden Datensatzes. Danach muss eine Skriptaktion erfolgen, die die UX-Bridge aktiviert:

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

In der danach anzulegenden Generierung sollte eine Teilgenerierung erfolgen. Seite und Datensatz werden später durch das Delta-Deployment automatisch eingetragen, so dass nur die gewünschte JMS-Nachricht erzeugt wird. Danach kann das Deployment der Webseiten wie gewohnt erfolgen.

Soll die Durchlaufzeit für die Nachrichten im Bus bis zur Veröffentlichung auf der Webseite gemessen werden, so muss als letztes noch die Aktion „UX-Bridge Statistics Report“ hinzugefügt werden, die folgendes Skript enthält:.

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

Der Service wartet maximal den in der Servicekonfiguration der UX-Bridge definierten Zeitraum, bis die Antworten der Adapter ausgewertet werden. Kommt in diesem Zeitfenster keine Antwort, so gilt die Nachricht als fehlerhaft zugestellt. Da die Antwortzeiten je nach Nachricht und System variieren können, ist dieser Wert konfigurierbar.

3.3.2.7.3 Workflow-Skripte importieren

Im nächsten Schritt müssen die benötigten Workflow-Skripte importiert werden:

- uxb_news_example_release_init
- uxb_news_example_release_script
- uxb_news_example_delete_init
- uxb_news_example_delete_script

Die Init-Skripte initialisieren in diesem Fall Variablen und schreiben diese in die Session, so dass die Methoden des UXB-Moduls, die in den anderen Skripten aufgerufen werden, auf diese zugreifen können.

In uxb_news_example_release_init müssen folgende Parameter konfiguriert werden:

Parameter	**Beispielwert**	**Beschreibung**
detail_page	pressreleasesdetails	Seitenreferenz der Seite, die die JMS-Nachrichten erzeugt
query_uid	Products.pressdetailsfilter	Tabellenabfrage, die alle Datensätze erzeugt
single_query_uid	Products.pressdetailfilter	Tabellenabfrage, die als Parameter die Id des Datensatzes erhält, der erzeugt werden soll
query_param	Id	Parametername der Tabellenabfrage
schedule_name	UX-Bridge	Name des Auftrags, der die JMS-Nachrichten erzeugen soll
scheduler_uxb-generate	UX-Bridge Generate	Name der Generierungsaktion der JMS Nachrichten

scheduler_generate	Generate	Name der Generierungsaktion für die HTML-Seiten
transition_name	Release	Name der Transition im Workflow (s. Workflow), die nach dem content_release_script geschaltet werden soll

Das Skript „uxb_news_example_release_script“ startet danach den zuvor konfigurierten Auftrag und führt die definierte Transition aus.

In uxb_news_example_delete_init müssen folgende Parameter konfiguriert werden:

Parameter	**Beispielwert**	**Beschreibung**
destinations	postgres	Name der Content Repositories, aus denen der Datensatz gelöscht werden soll
transition_name	release	Name der Transition im Workflow (s. Workflow), die nach dem content_delete_script geschaltet werden soll
object_type	news	Typ des Objekts, der gelöscht werden soll

Innerhalb des nachfolgenden Skripts „uxb_news_example_delete_script“ wird der selektierte Datensatz in FirstSpirit gelöscht und eine Nachricht über den UXB-Service und den Bus an das angeschlossene Content Repository geschickt, um dort die Löschaktion auszulösen.

3.3.2.7.4 Workflows importieren

Die Workflows „uxb_news_example_release“ und „uxb_news_example_delete“ rufen die zuvor konfigurierten Skripte auf.

3.3.2.7.5 Komplettabgleich

In dem FirstSpirit Beispielprojekt wurde der Komplettabgleich in dem Auftrag „UX-Bridge Full Deployment“ umgesetzt.

Hinweise zu dem Vorgehen des Komplettabgleich finden sie in Kapitel 2.1.4.2 „Komplettabgleich“ auf Seite 14.

3.3.3 Adapter

Der Adapter ist die Komponente, die die Daten vom UX-Bus einliest und in das entsprechende Live-Repository schreibt.

3.3.3.1 JAXB – XML Verarbeitung

Für die Verarbeitung des im Ausgabekanal definierten XML wird in diesem Bespiel JAXB verwendet. Die entsprechenden Klassen befinden sich im Package **com.espirit.moddev.examples.uxbridge.newsdrilldown.entity**.

Wie JPA wird bei JAXB mit Annotations gearbeitet um die XML-Tags an ein Java-Objekt zu binden.

```
@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 für das Datumsformat

Datumsangaben werden im FirstSpirit Ausgabekanal nach dem Format "yyyy-MM-dd'T'HH:mm:ssZ" formatiert. Damit dieses Format in den JAXB Klassen eingelesen werden kann, wurde die Klasse DateAdapter implementiert. Diese Klasse befindet sich im Package **com.espirit.moddev.examples.uxbridge.newsdrilldown.entity.type**.

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

3.3.3.1.2 UXBEntity, UXBContent, UXBMetaCategory und UXBCategory

Diese Klassen repräsentieren das im Ausgabekanal definierte Austauschformat.

3.3.3.1.3 UXBEntity

Diese Klasse entspricht dem von der UX-Bridge vorgegebenen Grundgerüst des Austauschformats.

3.3.3.1.4 UXBContent, UXBMetaCategory und UXBCategory

Die projektspezifischen JAXB Klassen zum Verarbeiten des Austauschformats. Hier sind die eigentlichen Informationen der Objekte enthalten, die über die UX-Bridge verteilt werden.

In diesem Beispiel sind dies also die Pressemitteilungen mit den entsprechenden Meta-Kategorien und Kategorien.

3.3.3.2 Hibernate-Domainklassen

Die Domainklassen befinden sich im Package **com.espirit.moddev.uxbridge**.

Um Mehrsprachigkeit abzubilden enthält jedes Objekt eine Sprache und jede Sprache wird als eigenständiges Objekt im Repository gespeichert.

Dieses Vorgehen hat zur Folge, dass die FirstSpirit-ID (UUID) nicht mehr eindeutig ist. Daher wurden Standard Hibernate/JPA Mechanismen verwendet um eine eindeutige ID zu erzeugen.

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

Diese ID im Live-Repository wird sich nach dem Löschen aus dem Repository und erneutem Hinzufügen ändern. Sollte ihr Anwendungsfall ein anderes Verhalten benötigen, bietet sich der Einsatz eines zusammengesetzten Primärschlüssels, aus FirstSpirit-ID und Sprache an.

3.3.3.2.1 News, NewsCategory, NewsMetaCategory

Der Aufbau der Klassen entspricht in etwa dem in FirstSpirit definierten Datenbank-Schema. Dieses Vorgehen ist nicht zwingend nötig, soll an dieser Stelle aber für das Verständnis hilfreich sein.

3.3.3.3 NewsHandler

Der NewsHandler im Package **com.espirit.moddev.exsamples.uxbridge.news.jpa** ist die Klasse, welche die Daten entgegen nimmt und bearbeitet. In dem Beispiel wurde im Handler für jedes der unterstützten Kommandos eine eigene Methode implementiert.

3.3.3.3.1 Command add

Speichern oder Aktualisieren einer Pressemitteilung im Live-Repository.

In diesem Fall ist darauf zu achten, dass die Meta-Kategorien und Kategorien im Austauschformat innerhalb der Pressemittelung übermittelt werden. Im Repository werden Kategorien und Meta-Kategorien aber getrennt gespeichert.

Für dieses Beispiel bedeutet dies, dass bei diesem Kommando neben der Pressemitteilung auch die enthaltenen Kategorien und Meta-Kategorien aktualisiert bzw. neu angelegt werden müssen.

3.3.3.3.2 Command delete

Löschen einer Pressemitteilung im Live-Repository und der dazugehörigen und im Auftragsscript definierten Detailseite (s. Auftrag anlegen) auf dem Webserver. Damit die Methode die richtige Seite auf dem Webserver finden kann, muss in der applicationContext.xml im Newshandler-Bean der Parameter „webpath“ auf den Pfad zum Verzeichnis des Webservers (z.B. „/home/tomcat/ webapps“) gesetzt werden.

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

Zu beachten ist, dass die Implementierung in diesem Beispiel nicht dafür sorgt, dass Meta-Kategorien oder Kategorien gelöscht werden, wenn es keine Pressemitteilung in einer dieser Kategorien gibt.

3.3.3.3.3 Command cleanup

Löschen aller Pressemitteilungen, die älter sind, als das gegebene Datum.

3.3.3.4 Routing

In der Spring-XML-Datei **WEB-INF/applicationContext.xml** werden die Komponenten konfiguriert. D.h. es werden die Datenbankverbindung, ConnectionPooling, JMS und das Routing definiert.

Das Routing wird in dem XML Bereich <camelContext id="camelContext" ...> definiert.

```
<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>
```

Weiterführende Informationen und Möglichkeiten finden Sie unter http://camel.apache.org/spring.html.

Eine detaillierte Erläuterung zur Erstellung der Rückmeldung befindet sich im Kapitel 3.5 „Verwendung der Camel Komponente zur Antwortgenerierung“.

3.3.3.4.1 Die Route uxbridge-commands

Es können beliebig viele Routen definiert werden, wobei die im Adapter definierten Routen nicht mit den Routen des UX-Bus verwechselt werden oder deren Aufgabe übernehmen sollten. Die Routen im Adapter sollten nur die für diesen Adapter wichtigen Routen beinhalten.

In dieser Beispielanwendung wurde eine Route definiert:

<route id="uxbridge-commands">

3.3.3.4.2 Nachrichtenquelle

Mit dem From-Tag wird dem Integration-Framework (Apache Camel) angegeben, von wo Daten eingelesen werden. In diesem Beispiel sieht das From-Tag folgendermaßen aus

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

Die Daten oder auch Nachrichten werden über das JMS Topic "BUS_OUT" eingelesen. An dieser Stelle kommt ein virtueller Endpunkt zum Einsatz (s. http://activemq.apache.org/virtual-destinations.html). Der Vorteil von virtuellen Endpunkten ist, dass für weitere Adapter keine Modifikation am Routing vorgenommen werden muss. Neue virtuelle Endpunkte müssen lediglich dem Namensschema „Consumer.%beliebiger Adaptername%.VirtualTopic.%Quell-Endpunkt%“ folgen. Durch die Virtualisierung können Nachrichten nicht wie bei einer Queue nur von einem Adapter gelesen werden, sondern alle entsprechenden Adapter erhalten die Nachricht.

Soll also zum Beispiel auch der neue Adapter „myAdapter“ Nachrichten konsumieren, die an den Endpunkt FS_OUT ausgeliefert werden, so könnte ein möglicher Endpunkt wie folgt aussehen:

activemq:Consumer.myAdapter.VirtualTopic.BUS_OUT

3.3.3.4.3 Filter

Durch das Filtern von Nachrichten erfasst der NewsHandler keine für ihn unwichtigen Nachrichten, also zum Beispiel Nachrichten für ein anderes Repository oder von einem anderen Objekt-Typ.

Zum Filtern der Nachrichten werden in diesem Beispiel einfach XPath-Ausdrücke verwendet.

Bei diesem funktional eingeschränkten Beispiel sind nicht alle Filter-Optionen nötig, wurden aber trotzdem integriert um Anhaltspunkte für eigene Ideen zu geben.

3.3.3.4.3.1 Filter Destination

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

An dieser Stelle werden Nachrichten gefiltert, die in der PostgreSQL Datenbank landen sollen. Alle anderen Nachrichten, auf die dieser Ausdruck nicht passt, werden nicht weiter behandelt. Da Nachrichten gleichzeitig in verschiedene Live-Repositories geschrieben werden können, wird an dieser Stelle mit contains gearbeitet.

3.3.3.4.3.2 Filter Object-Type

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

Der NewsHandler kann nur Objekte vom Typ "news_article" bearbeiten. Auch hier gilt, dass die Nachrichten, für die dieser Ausdruck nicht zutreffend ist, nicht weiter behandelt werden. In der Regel enthalten Nachrichten immer nur ein Objekt eines Typs, daher wird hier mit "=" gearbeitet.

3.3.3.4.3.3 Filter Command

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

Im letzten Schritt werden die Nachrichten nach den Kommandos gefiltert.

3.3.3.4.3.4 JAXB Konvertierung

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

Das XML der Nachricht wird mittels JAXB in eine Java-Klasse konvertiert.

3.3.3.4.3.5 Methoden-Aufruf

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

Zu guter Letzt wird am Ende der Filterkette die entsprechende Methode in dem NewsHandler aufgerufen.

3.3.3.5 Starten der Beispiel Adapter

Mit dem folgendem Aufruf kann die API in das lokale Maven Repository geladen

werden:

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

Eine Beispielumsetzung kann wie folgt aussehen:

```
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
```

Die Beispiel-Adapter können über die Kommandozeile gebaut werden:

```
mvn package
```

Die daraus resultierende War-Datei kann auf einen beliebigen ServletContainer (Tomcat, Jetty usw.) deployed werden.

Alternativ können die Adapter auch über die Kommandozeile gestartet werden:

```
mvn tomcat7:run
```

Um den Port des dadurch gestarteten Tomcat anzupassen, muss die Datei pom.xml im Verzeichnis des jeweiligen Adapters angepasst werden.

3.3.3.6 Enthaltene Tests

In dem Beispielprojekt sind Unit- und Integrations-Tests enthalten. Für die Tests werden eine In-Memory Datenbank und jMockMongo (https://github.com/thiloplanz/jmockmongo) verwendet. Damit die Tests für den MongoDB-Adapter gestartet werden können, muss das jMockMongo Jar in das lokale Repository importiert werden.

Die Integrations-Tests können mit folgendem Aufruf gestartet werden: **mvn verify -Pintegration-test**

3.4 Verwendung des UXB-Service-API

Zur Verwendung des UXBServices in einem Modul oder in einem Skript, muss das API-Jar in der entsprechenden Version in den Classpath aufgenommen werden.

Der Zugriff auf den UXBService erfolgt dann über den Aufruf:

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

Eine Beispiel-Implementierung für Delete- und Release-Executables finden sie im Github-Repository unter uxbridge-api-example.

3.4.1 Demo-Projekt erstellen

Um das Demo-Projekt zu erstellen, wird Apache-Maven benötigt. Außerdem ist es erforderlich, dass das fs-access.jar als Artefakt im lokalen Maven-Repository installiert ist.

Sind diese Vorrausetzungen erfüllt, kann das Projekt mit **„mvn clean package“** gebaut werden. In diesem Schritt wird im target Verzeichnis des Projektes ein FSM erstellt, dass über die FirstSpirit-Server Adminkonsole installiert werden kann. Danach können die enthaltenen Beispiel Executables verwendet werden.

3.4.2 Verwendung

Die beiden Beispiel-Implementierungen können in den beiden vorangegangenen Tutorials verwendet werden. Dafür muss wie folgt vorgegangen werden:

3.4.2.1 Skript „Datensatz löschen“ (uxb_content_delete_script)

Das Skript wurde als Executable Implementiert, daher wird an dieser Stelle nur noch das Executable aufgerufen.

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

3.4.2.2 Skript „Datensatz veröffentlichen“ (uxb_content_release_script)

Dieses Skript wird ebenfalls durch das entsprechende Executable ersetzt.

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

3.4.2.3 CamelContext Rückgabe

FirstSpirit erwartet nach einer Interaktion mit einem Adapter über die UX-Bridge eine Rückmeldung in Form eines XML-Dokumentes. Hierfür steht eine Camel Component

zur Verfügung, die dieses XML Dokument erstellt.

Um Zugriff auf die Component im CamelContext zu haben, muss die Component wie folgt eingebunden werden:

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

Um die Funktion zu verwenden, muss sie bei der Weitergabe der Daten an BUS_IN mit aufgerufen werden.

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

Dies geschieht sowohl bei der erfolgreichen Übermittlung der Daten an die Datenbank, als auch im Falle einer Exception. Die Funktion erstellt jeweils die passende Antwort.

3.5 Verwendung der Camel Komponente zur Antwortgenerierung

Mittels dieser Komponente ist es möglich, die Antwort, die FirstSpirit von einem Adapter erwartet, zu generieren. Dies gilt sowohl für eine reguläre Antwort, als auch für eine Antwort im Fehlerfall. Voraussetzung für den Einsatz dieser Komponente ist, dass der Adapter mit Apache Camel umgesetzt wird. Mehr Informationen zu Apache Camel sind auf Apache Camel Homepage (http://camel.apache.org/) zu finden.

3.5.1 Einbinden der Komponente

Um einen Zugriff auf die Camel Komponente zu bekommen, muss diese eingebunden werden. Dies geschieht über die mit ausgelieferte Datei uxbridge-camel-component-<version>.jar. Diese muss in die Java Class Path des Projektes eingebunden werden. (Bei Eclipse: Rechts Klick auf das Projekt->Java Build Path->Libraries->Add external JARs)

3.5.2 Einbindung der Komponente

Um die Komponente innerhalb eines Adapters nutzen zu können, muss die Komponente als bean eingebunden werden. Der Aufruf hierzu sieht wie folgt aus: <bean id="adapterReturn" class="org.uxbridge.camel.component.AdapterReturnComponent"></bean>

Die ID kann frei gewählt werden. Eine Änderung der ID erfordert jedoch eine entsprechende Anpassung des im nachfolgenden Unterkapitel dargestellten Aufbaus

der URL.

3.5.3 Aufbau der URL

Der Aufbau der URL beginnt mit dem Aufruf der Komponente. Dies geschieht über die ID, die beim Einbinden der Komponente angegeben wird. Danach folgt der Aufruf der Ziels. Am Ende wird noch der Parameter destination an die URL angehängt.

Der gesamte Aufbau kann dann wie folgt aussehen:

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

Beim Aufruf innerhalb des Exception Handlings ist ein zweiter Parameter notwendig, um welchen der Aufbau der URL ergänzt werden muss:

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

3.5.4 Parameter

Es werden zwei Parameter an die Komponente übergeben.

Der erste Parameter ist die destination, von der die Antwort erzeugt wird. Dieser Parameter wird direkt mittels eines Fragezeichens an die URL angehängt. Da innerhalb des Aufrufs des Adapters mehrere destinations übergeben werden können, dient dieser Parameter der Unterscheidung des Ziels, das zu dieser Antwort geführt hat.

Der zweite Parameter wird nur im Fall einer Exception benötigt. Da im Fall einer Exception der aktuelle Status der Nachricht an den Exception Handler übergeben wird, kann es vorkommen, dass der Inhalt der Nachricht nicht mehr vollständig und beispielsweise das root-Element des XML-Dokuments nicht mehr vorhanden ist. Da für die Verarbeitung allerdings das gesamt XML-Dokument benötigt wird, ist es notwendig, dieses vor der Verarbeitung im Header der Nachricht zwischen zu speichern. Das Zwischenspeichern erfolgt mit <setHeader headerName="bodyTemp"><simple>${body}</simple></setHeader>. Der headerName ist dabei frei wählbar, muss der Komponente aber mitgeteilt werden. Die geschieht über den zweiten Parameter bodyValue.

4 Erweiterungsmöglichkeiten

4.1 Eigene Nachrichten aus FirstSpirit erzeugen

Durch eine Erweiterung ist es möglich, über den UXB-Service eigene Nachrichten, die ein frei definierbares Format besitzen, an den Bus zu schicken. Dazu muss das Interface des UxbMessageGenerators implementiert werden. Das Interface stellt bereits den Kontext des Auftrags sowie einige Methoden, die Informationen zum generierten Element liefern, zur Verfügung. Über den Kontext ist ein Zugriff auf das gesamte Projekt und dessen Elemente möglich.

4.1.1 Interface UxbMessageGenerator

Um einen eigenen UxbMessageGenerator zu implementieren, muss das API-Jar der UX-Bridge in der entsprechenden Version in den Classpath aufgenommen werden (vgl. Kapitel **Fehler! Verweisquelle konnte nicht gefunden werden. Fehler! Verweisquelle konnte nicht gefunden werden.**). Zusätzlich besteht eine Abhängigkeit zum fs-access.jar. Dieses wird mit jedem FirstSpirit-Server installiert und muss ebenfalls im Classpath verfügbar gemacht werden.

Die neue Klasse implementiert zunächst das UxbMessageGenerator-Interface:

```
public class DemoMessage implements UxbMessageGenerator
```

Innerhalb der Klasse müssen darüber hinaus einige Methoden implementiert werden, die vom UXB-Service genutzt werden um Informationen zu übergeben und die im Folgenden kurz beschrieben werden:

setData(byte[ ] data)

Das data-Objekt enthält den gerenderten Inhalt des Ausgabekanals.

setCreateTime(long createTime)

Die createTime ist die Zeit, zu welcher der Auftrag gestartet wurde.

setStartTime(long startTime)

Die startTime ist der Zeitpunkt, an dem die Nachricht erzeugt wurde.

setSchedulerId(long schedulerId)

Die schedulerId ist die Id des gestarteten Auftrags.

setProjectId(long projectId)

Die Id des Projekts, das generiert wird.

setGenerationContext(GenerationContext generationContext)

Im generationContext befindet sich der gesamte Generierungs-Kontext des Auftrags. Über diesen ist ein Zugriff auf das gesamte Projekt und dessen Elemente sowie auf das aktuell generierte Element möglich.

generate()

Die generate-Methode wird vom UXB-Service zur Nachrichtenerzeugung aufgerufen und muss die generierte Nachricht vom Typ ‚Document' zurück liefern.

4.1.2 Aufruf des eigenen UxbMessageGenerators

Innerhalb des Auftrags, der die UXB-Nachrichten generiert, muss vor der Script-Aktion „Activate Generation" (vgl. 2.1.4.1 Teilgenerierung) eine weitere Script-Aktion ausgeführt werden.

Innerhalb des Scripts muss die Kontextvariable ‚MessageGenerator' mit dem vollqualifizierten Klassennamen der Klasse belegt werden, der die Nachrichtengenerierung übernehmen soll.

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

Die Klasse muss dabei innerhalb des FS-Servers verfügbar gemacht werden, z.B. über ein Modul. Um ein korrektes Classloading zu gewährleisten, kann die Klasse als Public-Komponente mit modul-lokalen Resourcen konfiguriert werden.

4.1.2.1 Aufruf im Cluster-Betrieb

Da im Cluster-Betrieb auf den Slave-Systemen keine Scripte gestartet erfolgt hier die Übergabe des Klassennamens der Klasse die die Nachrichtengenerierung übernehmen soll nicht über die Script-Aktion, sondern im Template der Projekteinstellungsseite. Dieses muss um folgende Zeile erweitert werden:

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

Darüber hinaus ist darauf zu achten, das in der Generierungsaktion im Auftrag keine

„Medien im Generierungsverzeichnis erzeugt“ werden.

5 Anhang

5.1 Konvertierungsregel 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]
```