

PublishMQ SPI Implementation

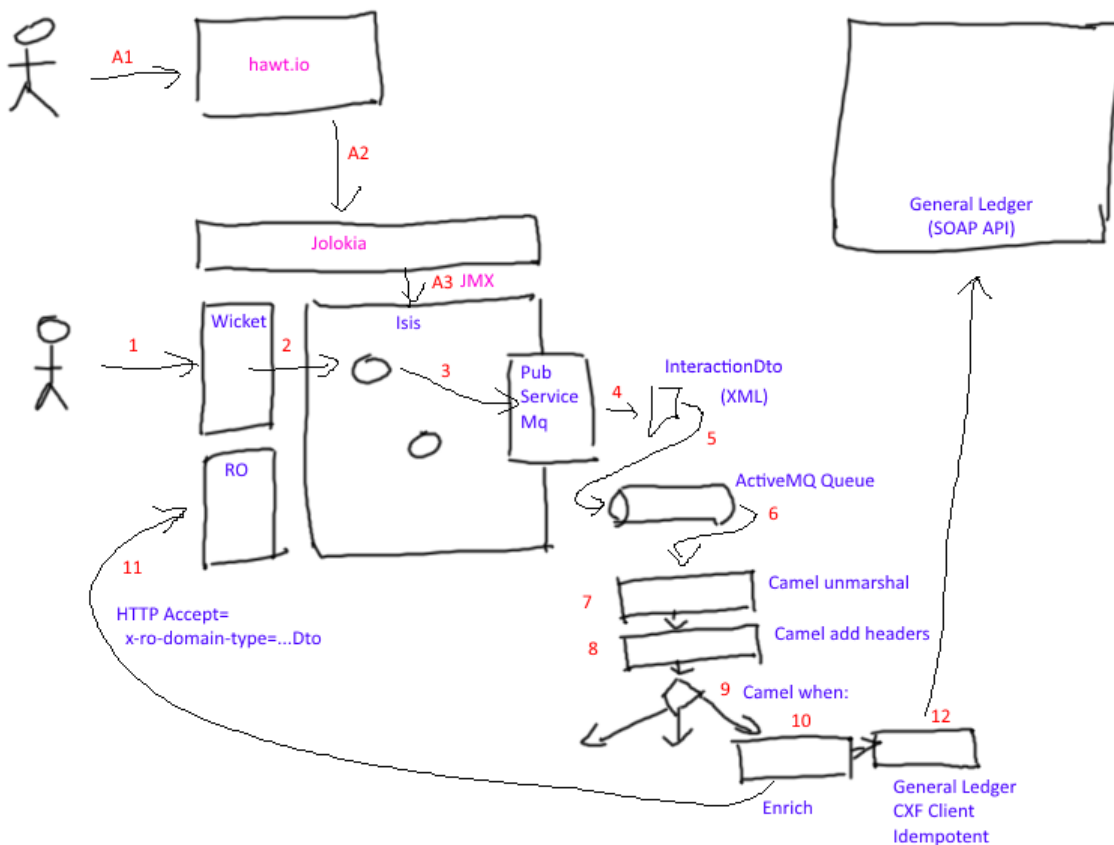
Table of Contents

Screenshots	2
Invoke an action	3
How to configure/use	6
Classpath	6
Bootstrapping	7
Configure ActiveMQ	7
Canonical DTOs	9
Generate Canonical DTO referencing Apache Isis' DTOs	9
Centralized Spring configuration	9
Known issues	10
Dependencies	11
Dependencies	12
publishmq/servicespi	12
publishmq/jdo	12
publishmq/camel	13
publishmq/statusclient	13

This module (`isis-module-publishmq`) provides an implementation of Apache Isis' `PublisherService` SPI that submits an XML representation of an `MemberInteractionDtos` to an `ActiveMQ` queue.

The `quickstart` app also demonstrates how this member interaction event (action invocation or property edit) can be routed using `Apache Camel`, whereby the payload is enriched using Apache Isis' own `Restful Objects` viewer (obtaining additional information);

The diagram below shows the moving parts:



One of the design objectives for the `PublishMq` module is to allow the `ActiveMQ` queue (and therefore any `Camel` routing) to be either embedded (as in the example app) or to be remote. This is one of the reasons why the payload posted to the queue is the XML representation of a JAXB object (the `InvocationDto`).

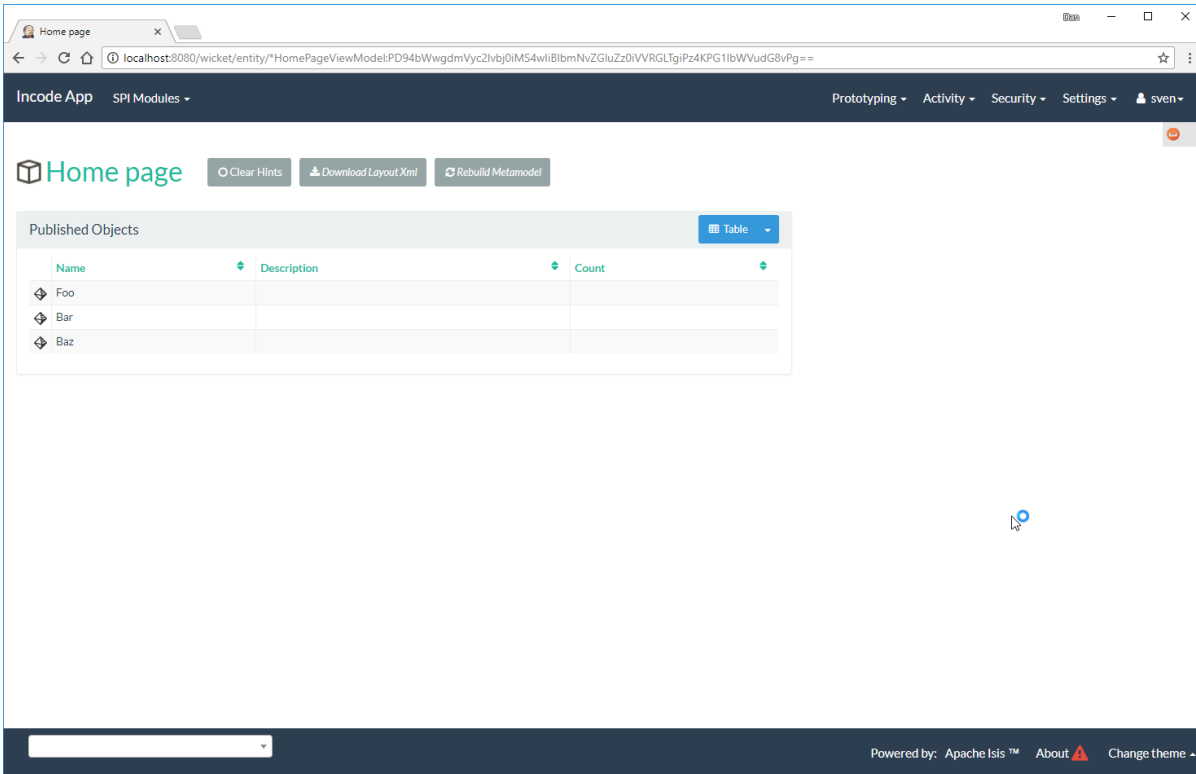


Note that the example app does *not* include an external system (`General Ledger` in the diagram), ie step #12.

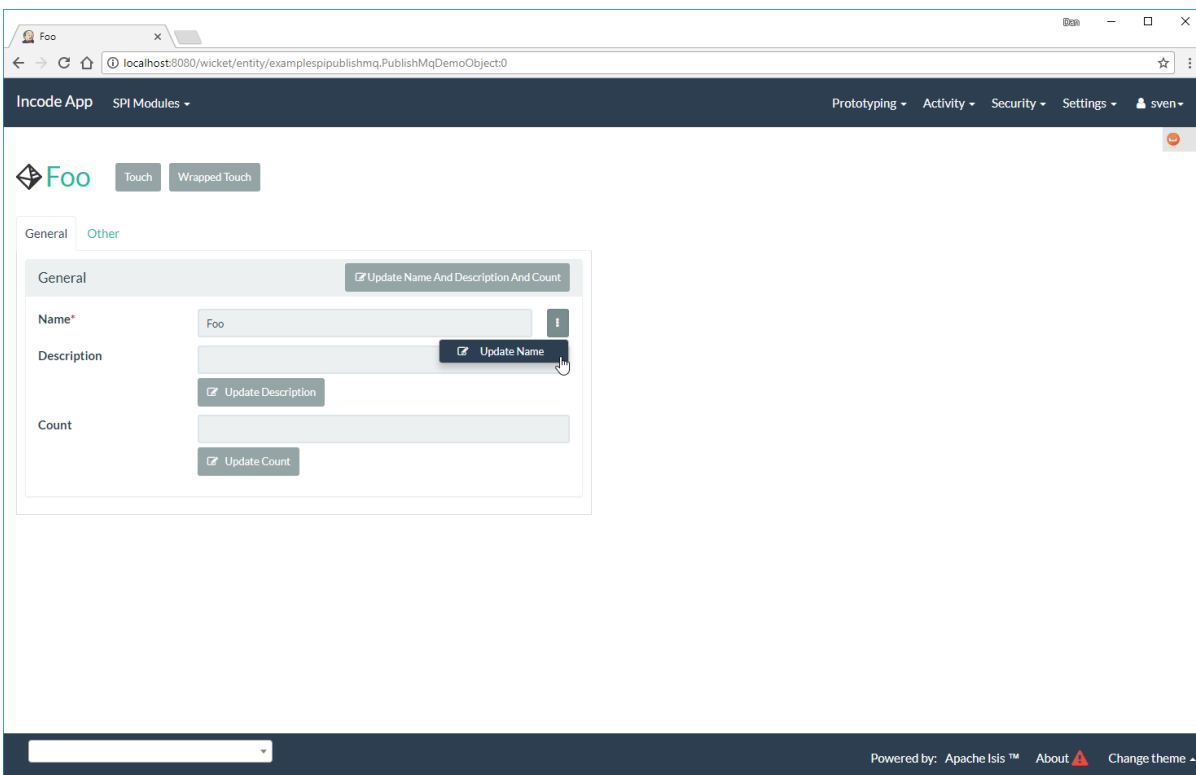
Screenshots

The module's functionality can be explored by running the [quickstart with example usage](#) using the `org.incode.domainapp.example.app.modules.ExampleDomSpiPublishMqAppManifest`.

A home page is displayed when the app is run:



This returns the first demo object (an instance of `PublIshMqDemoObject`):



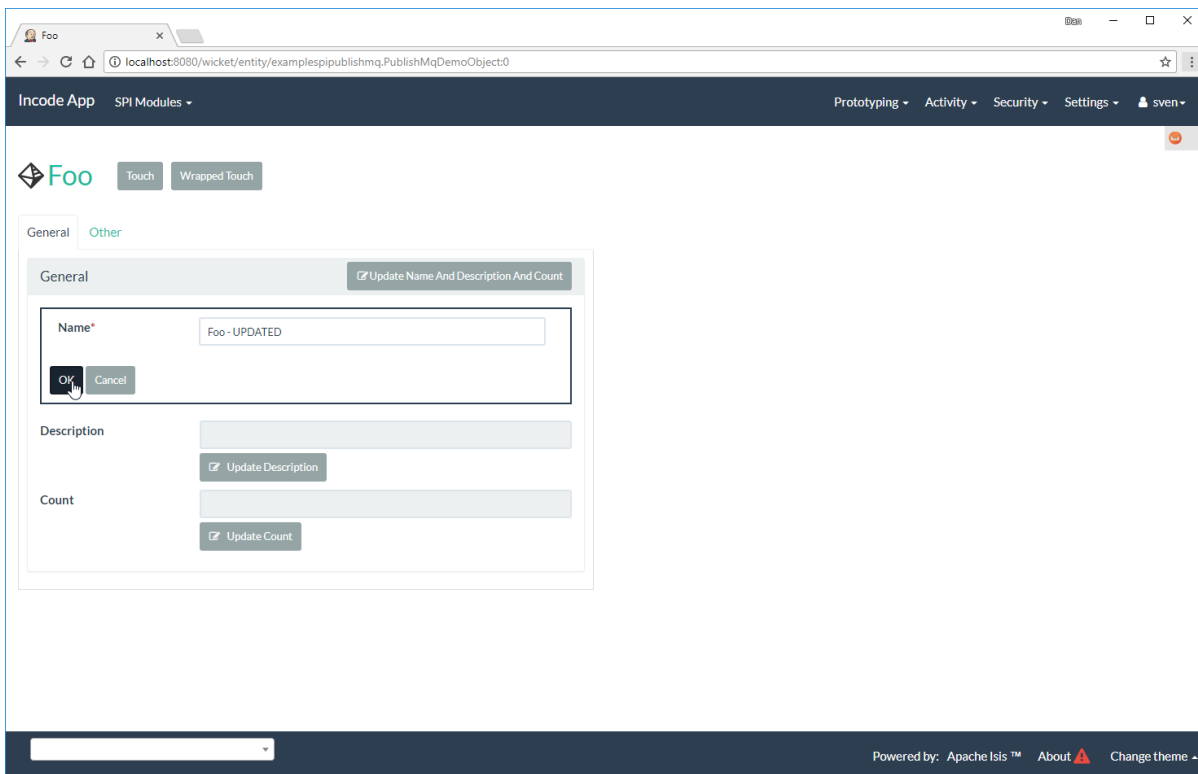
Invoke an action

The `updateName()` action is defined as:

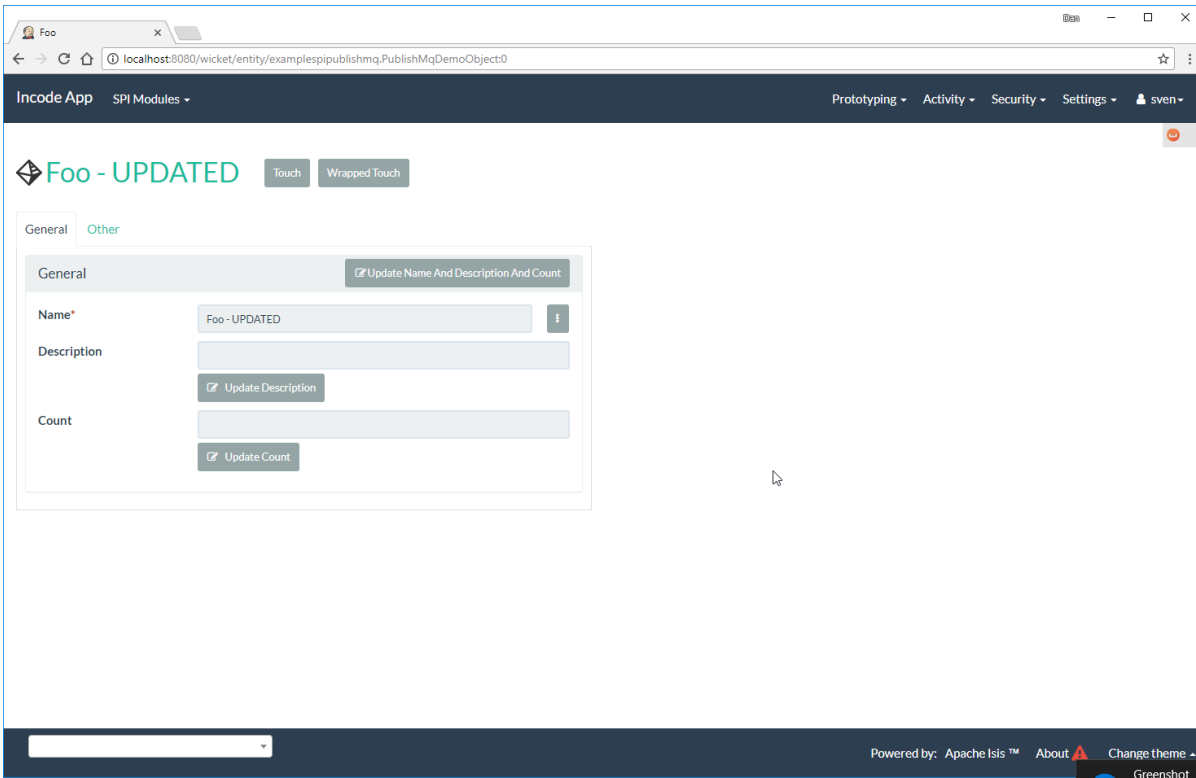
```
@Action(  
    semantics = SemanticsOf.IDEMPOTENT,  
    publishing = Publishing.ENABLED ①  
)  
public PublishMqDemoObject updateName(  
    @ParameterLayout(named="Name") final String name) {  
    setName(name);  
    return this;  
}
```

① invocations of this action will be published to the configured implementation of `PublishingService`.

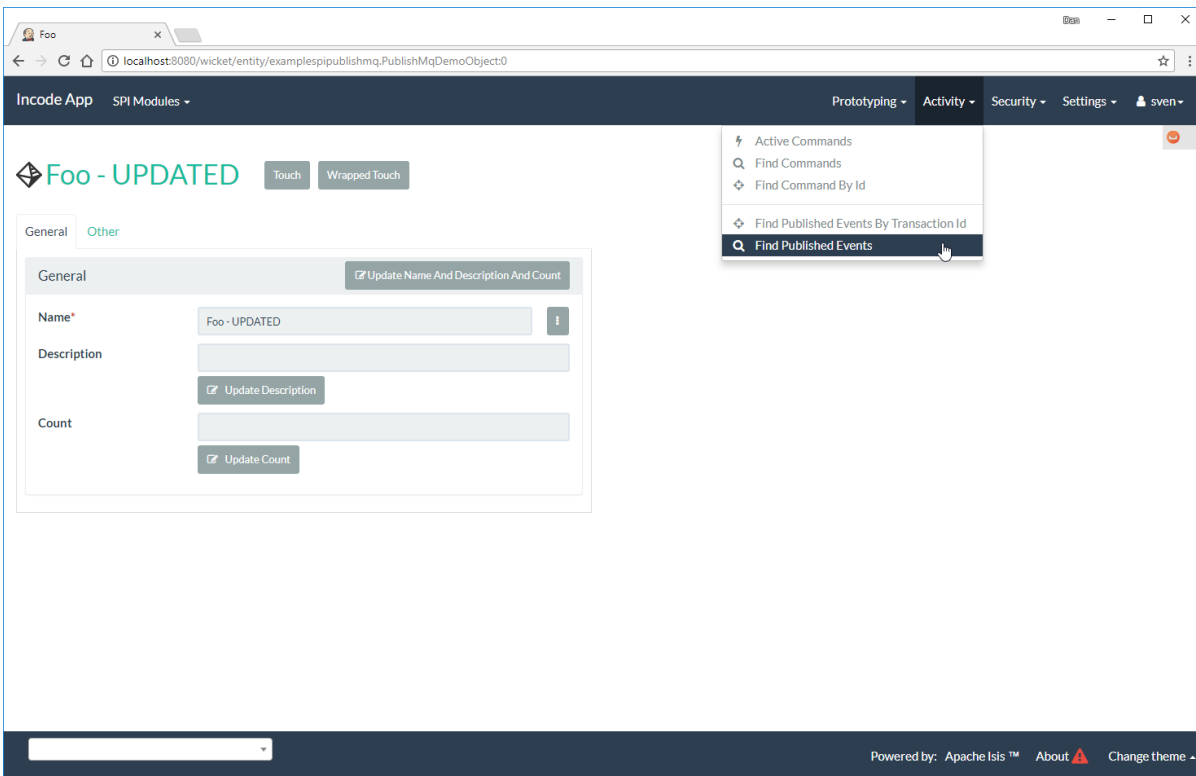
Invoke the action:



the value of the `name` property should, of course, be updated:



From the activity menu the published events (also persisted as entities) can be inspected:



... one of which is to update the name:

Type	User	Timestamp	Transaction Id	Sequence	Event Type	Class	Action	Property Id	Object	Pre Value	Post Value
🔍	sven	2017-09-09 14:01:30.948	227cd195-5ecf-41fb-be37-a924228f215f	8	Changed Objects						
🔍	sven	2017-09-09 14:01:30.945	227cd195-5ecf-41fb-be37-a924228f215f	7	Changed Objects						
🔍	sven	2017-09-09 14:01:30.674	227cd195-5ecf-41fb-be37-a924228f215f	0	Action Invocation	Publish Mq Demo Object	Update Name		examplesipublishmq.PublishMqDemoObject:0		
🔍	initialisation	2017-09-09 13:54:56.023	58fcc42f-0025-43c3-a321-9153aaf095a5	2	Changed Objects						
🔍	initialisation	2017-09-09 13:54:55.872	58fcc42f-0025-43c3-a321-9153aaf095a5	1	Changed Objects						
🔍	initialisation	2017-09-09 13:54:55.804	58fcc42f-0025-43c3-a321-9153aaf095a5	0	Changed Objects						

The published entity contains XML which captures the details of the member interaction:

Action Invocation of 'Update Name' on Foo - UPDATED

Recent Events Published By User

Identifiers Republish

User* sven Open Recent Commands By User

Timestamp* 2017-09-09 14:01:30.674

Transaction Id* 227cd195-5ecf-41fb-be37-a924228f215f Command

Sequence* 0

Event Type* Action Invocation

Target

Class Publish Mq Demo Object

Action Update Name

Object examplesipublishmq.PublishMqDemoObject:0 Open

Detail

Member Identifier org.incode.domainapp.example.dom.spi.publishmq.dom.demo.PublishMqDemo

Serialized Form

```

updateName|</ix:title>
<ix:metrics>
<ix:timings>
<com:startedAt>2017-09-09T14:01:30.674+01:00</com:startedAt>
<com:completedAt>2017-09-09T14:01:30.676+01:00</com:completedAt>
</ix:timings>
<ix:objectCounts>
<ix:loaded before="1" after="1"/>
<ix:dirty before="0" after="1"/>
</ix:objectCounts>
</ix:metrics>
<ix:parameters>
<cmd:parameter name="Name" type="string" null="false">
<com:string>Foo - UPDATED</com:string>
</cmd:parameter>
</ix:parameters>
<ix:returned type="reference" null="false">
<com:reference type="examplesipublishmq.PublishMqDemoObject" id="0"/>
</ix:returned>
</ix:execution>
</ix:interactionDto>

```

How to configure/use

You can either use this module "out-of-the-box", or you can fork this repo and extend to your own requirements.

The module itself consists of several submodules:

- the `publishmq-dom-servicespi` submodule
which contains the `PublishingService` SPI implementation that actually publishes to an ActiveMQ queue
- the (optional, but recommended) `publishmq-dom-jdo` submodule
which allows published events to be persisted as `PublishedEvent` entities
- the (optional) `publishmq-dom-camel` submodule
which provides utility class to help route messages
- the (optional) `publishmq-dom-statusclient`
that provides utility classes to log status messages with the originating system via the RestfulObjects viewer's REST API.

Classpath

Update your classpath:

- by adding importing the parent module's dependency into in your parent module's `pom.xml`:

```
<dependencyManagement>
  <dependencies>
    <dependency>
      <groupId>org.isisaddons.module.publishmq</groupId>
      <artifactId>isis-module-publishmq-dom</artifactId>
      <version>1.16.1</version>
      <type>pom</type>
      <scope>import</scope>
    </dependency>
    ...
  </dependencies>
</dependencyManagement>
```

- by adding the `-dom-servicespi` dependency in your project's `dom` module's `pom.xml`:


```
<dependencies>
  <dependency>
    <groupId>org.isisaddons.module.publishmq</groupId>
    <artifactId>isis-module-publishmq-dom-servicespi</artifactId>
  </dependency>
  ...
</dependencies>
```

- (if you are using Camel for routing and want to use the `AddExchangeHeaders` utility class) by adding (in the appropriate module within your app) the dependency:

```
<dependencies>
  <dependency>
    <groupId>org.isisaddons.module.publishmq</groupId>
    <artifactId>isis-module-publishmq-dom-camel</artifactId>
  </dependency>
  ...
</dependencies>
```



Note that the [quickstart with embedded camel](#) configures this already, so use as a guide if need be.

Check for later releases by searching [Maven Central Repo](#).

For instructions on how to use the latest `-SNAPSHOT`, see the [contributors guide](#).

Bootstrapping

In the `AppManifest`, update its `getModules()` method, eg:

```
@Override
public List<Class<?>> getModules() {
    return Arrays.asList(
        ...
        org.isisaddons.module.publishmq.PublishMqModule.class,
        ...
    );
}
```

You might also need to specify the package for any new services that you have written, eg implementation of `ContentNegotiationService` or similar.

Configure ActiveMQ

Configure ActiveMQ so that the publishing service implementation can post to a queue called

memberInteractionsQueue.

In the [quickstart with embedded camel](#) app this is done using Spring ([activemq-config.xml](#)):

+

```
<beans
  xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
http://activemq.apache.org/schema/core
http://activemq.apache.org/schema/core/activemq-core.xsd">
  <broker xmlns="http://activemq.apache.org/schema/core"
    brokerName="broker"
    dataDirectory="${activemq.data}"
    useShutdownHook="false"
    useJmx="true"
  >
    ...
    <destinations>
      <queue physicalName="memberInteractionsQueue"/>
    </destinations>
    ...
  </broker>
</beans>
```

This is bootstrapped in the [web.xml](#):

```
<listener>
  <listener-class>org.springframework.web.context.ContextLoaderListener</listener-
class>
</listener>
<context-param>
  <param-name>contextConfigLocation</param-name>
  <param-value>
    classpath:activemq-config.xml
  </param-value>
</context-param>
```

Canonical DTOs

The [quickstart with embedded camel](#) app contains a few other little tricks that may be useful if you are looking to deploy a similar architecture for your own application.

Generate Canonical DTO referencing Apache Isis' DTOs

As of 1.13.0 Apache Isis includes the `ixn.xsd` (member interaction) schema (replacing and generalizing the `aim.xsd` provided from 1.9.0 through 1.12.x). The `PublishingServiceMq` uses this `ixn.xsd` schema (or rather, its Java JAXB equivalent, `InteractionDto`), directly.

The similar `common.xsd` is *also* used by the demo app in the construction of its own canonical `DemoObjectDto` (use of `OidDto` to represent a bookmark to a published domain object).

Centralized Spring configuration

In the example app Spring is used to bootstrap ActiveMQ (`activemq-config.xml`), and Camel (`camel-config.xml`), and also the fake SOAP Subscriber (`externalSystemFakeServer-config.xml`). The configuration for all is centralized through a `propertyPlaceholderConfigurer` bean (defined in `propertyPlaceholderConfigurer-config.xml`). The location of the property file is specified in the `web.xml`:

```
<context-param>
  <param-name>spring.config.file</param-name>
  <param-value>classpath:spring.properties</param-value>
</context-param>
```

where `spring.properties` is:

```
activemq.data=activemq-data
enrichWithCanonicalDto.base=http://localhost:8080/restful/
enrichWithCanonicalDto.username=sven
enrichWithCanonicalDto.password=pass
updateExternalSystemAdapter.endpointAddress=http://localhost:8080/soap/ExternalSystemA
dapter/DemoObject
```

If necessary the location of this config file can be overridden; see [this topic](#) in the Apache Isis user guide.

Known issues

None known at this time.

Dependencies

In addition to Apache Isis, this module also depends upon:

- [ActiveMQ](#)
- (optional) [Camel](#)

Dependencies

The `publishmq` module actually consists of four distinct submodules, which can be used to some extent independently.

`publishmq/servicespi`

Maven can report modules dependencies of this submodule using:

```
mvn dependency:list -o -pl modules/spi/publishmq/impl/servicespi -D
excludeTransitive=true
```

which, excluding Incode Platform and Apache Isis modules, returns these compile/runtime dependencies:

```
org.apache.activemq:activemq-all:jar:5.11.1
```

For further details on 3rd-party dependencies, see:

- [Apache ActiveMQ](#)

`publishmq/jdo`

This submodule can be considered optional (though its use *is* recommended). If not included then published messages are simply not persisted as JDO entities.

Maven can report modules dependencies of this submodule using:

```
mvn dependency:list -o -pl modules/spi/publishmq/impl/jdo -D excludeTransitive=true
```

which, excluding Incode Platform and Apache Isis modules, returns these compile/runtime dependencies:

```
org.slf4j:slf4j-api:jar:1.7.21
```

From the Incode Platform it uses:

- `publishmq/servicespi` submodule, above.

For further details on 3rd-party dependencies, see:

- [Slf4j](#)

publishmq/camel

This submodule is considered optional because it merely provides a supporting utility class ([AddExchangeHeaders](#)).

Maven can report modules dependencies of this submodule using:

```
mvn dependency:list -o -pl modules/spi/publishmq/impl/camel -D excludeTransitive=true
```

which, excluding Apache Isis modules, returns these compile/runtime dependencies:

```
org.apache.camel:camel-core:jar:2.15.2
org.apache.camel:camel-spring:jar:2.15.2
org.apache.camel:camel-spring-javaconfig:jar:2.15.2
org.apache.camel:camel-jms:jar:2.15.2
```

For further details on 3rd-party dependencies, see:

- [Apache Camel](#)

publishmq/statusclient

This submodule is considered optional; it provides the mechanism for a beans within a Camel route to report status back to the originating system via the RestfulObjects viewer's REST API.



Using the status client requires the originating system to have configured `publishmq/jdo` to persist the status messages.

Maven can report modules dependencies of this submodule using:

```
mvn dependency:list -o -pl modules/spi/publishmq/impl/statusclient -D
excludeTransitive=true
```

which, excluding Apache Isis modules, returns these compile/runtime dependencies:

```
org.slf4j:slf4j-api:jar:1.7.21
org.jboss.spec.javax.ws.rs:jboss-jaxrs-api_2.0_spec:jar:1.0.0.Final
com.fasterxml.jackson.core:jackson-databind:jar:2.8.0
```

For further details on 3rd-party dependencies, see:

- [Slf4j](#)
- [Jackson](#)