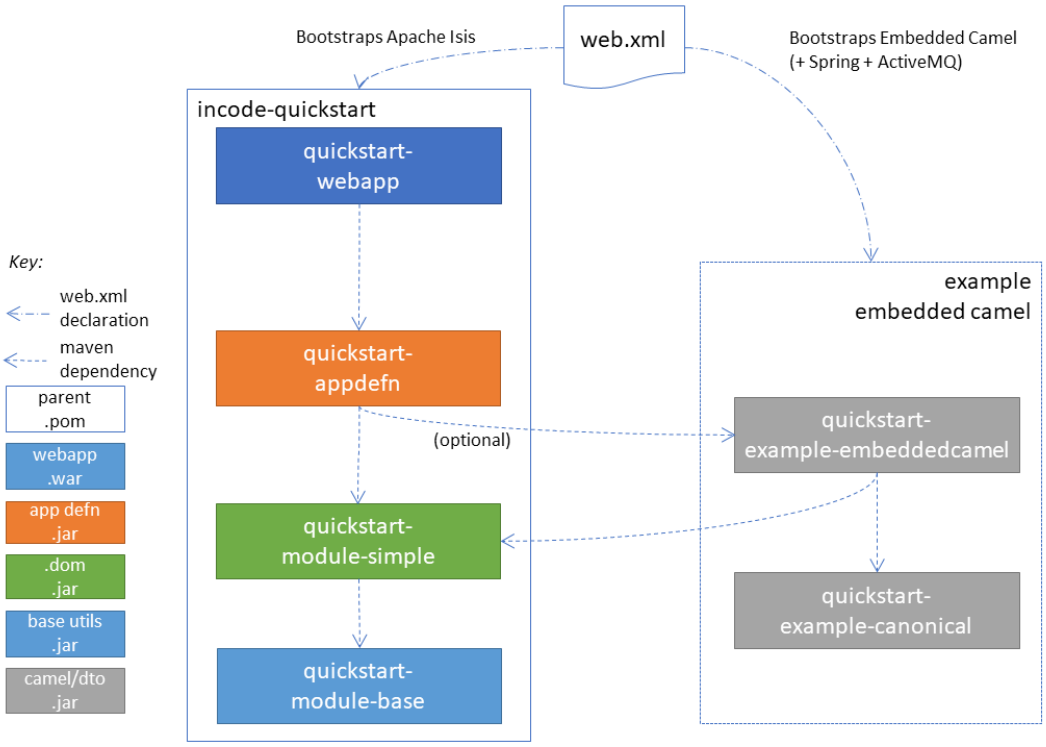
Quickstart with Embedded Camel

Table of Contents

The application generated by the quickstart archetype can be extended to also configure the application to bootstrap a *Spring Framework* context alongside the Apache Isis framework. This Spring context hosts an embedded *Apache Camel* instance, configured to consume from the ActiveMQ message queue published to by the publishmq module.

Project Structure

The diagram below shows how the structure of the application is extended to support this:



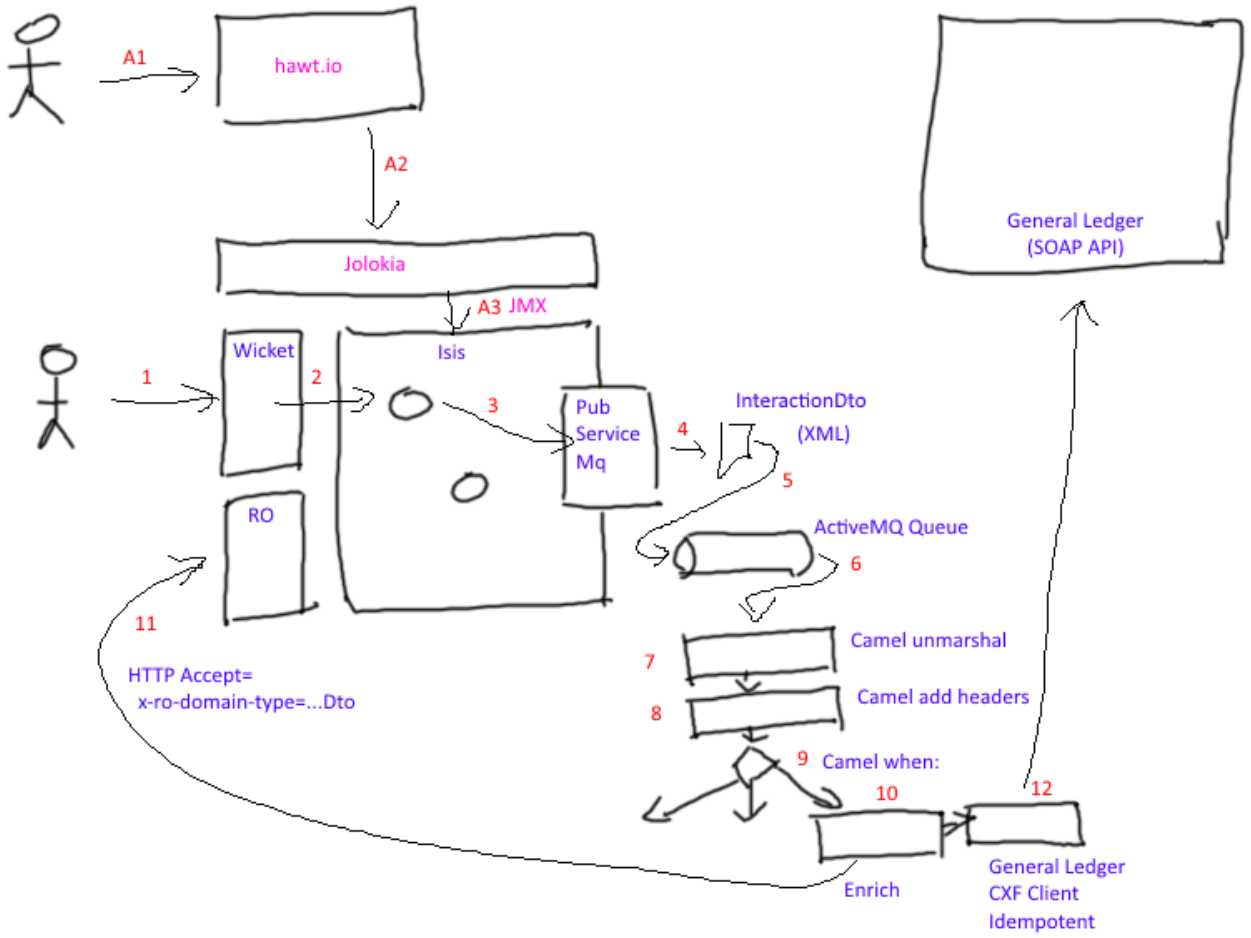
where:

- `quickstart-example-embeddedcamel` provides a Spring context file defining a Camel route for consuming published events, as well as supporting Camel bean instances and other classes
- `quickstart-example-canonical` defines a DTO for the `SimpleObject` domain object.

The idea here is that the event published by the publishmq module only sends a notification that a significant interaction (action invocation or property edit) has occurred. It does *not* attempt to gather together all the information that might be required by downstream consumers interested in this interaction.

Instead, the subscribers (implemented within the Camel route) use the REST API to callback to the originating system, and exploits the ContentMappingService SPI to obtain DTOs specific to the use case.

The diagram below attempts to explain this:



That is, the `quickstart-exampe-embedded-camel` implements steps 5 through to 11 (step 12 is not implemented at all, just shown as a possible use case).

Including the Example Modules

To include the Embedded Camel modules, just uncomment the relevant blocks surrounded by these words:

```
Uncomment to include example modules that set up embedded camel: START
```

to

```
Uncomment to include example modules that set up embedded camel: END
```

You should find there are two such blocks in the various `pom.xml` files, one block in `web.xml`, and one block in the `DomainAppAppManifest.java`.

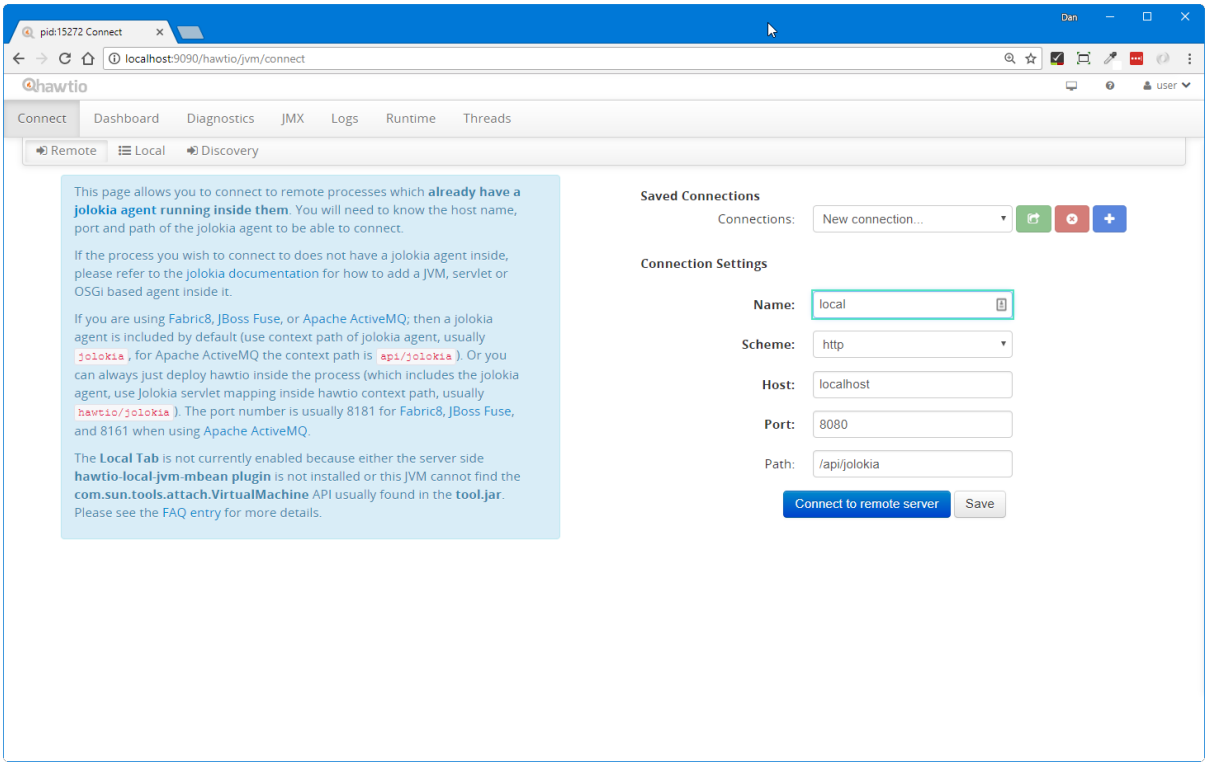
Camel Route

To monitor the Camel route, we can use the hawt.io console (as also used for monitoring ActiveMQ messages, see the base quickstart's publshing support).

Download the hawt.io JAR file and start using:

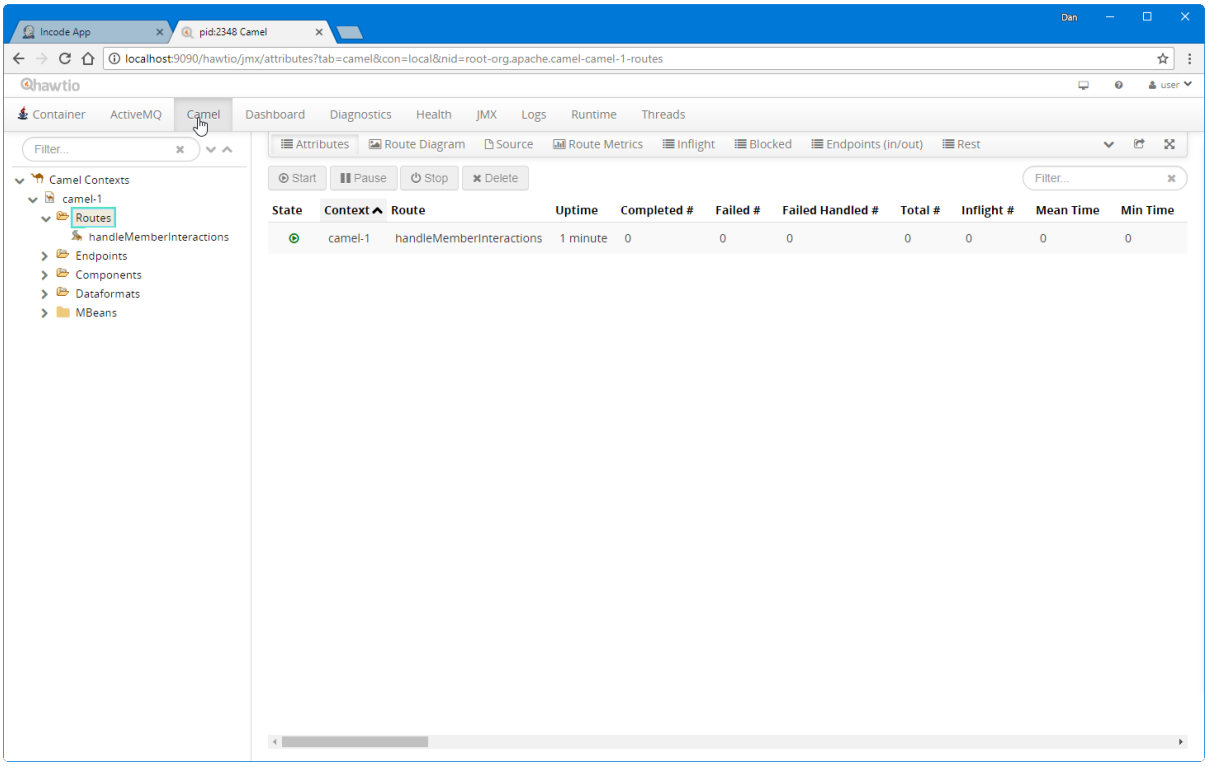
```
java -jar hawtio-app-1.5.3.jar --port 9090
```

Then connect to the jolokia servlet (configured in the quickstart's `web.xml`):

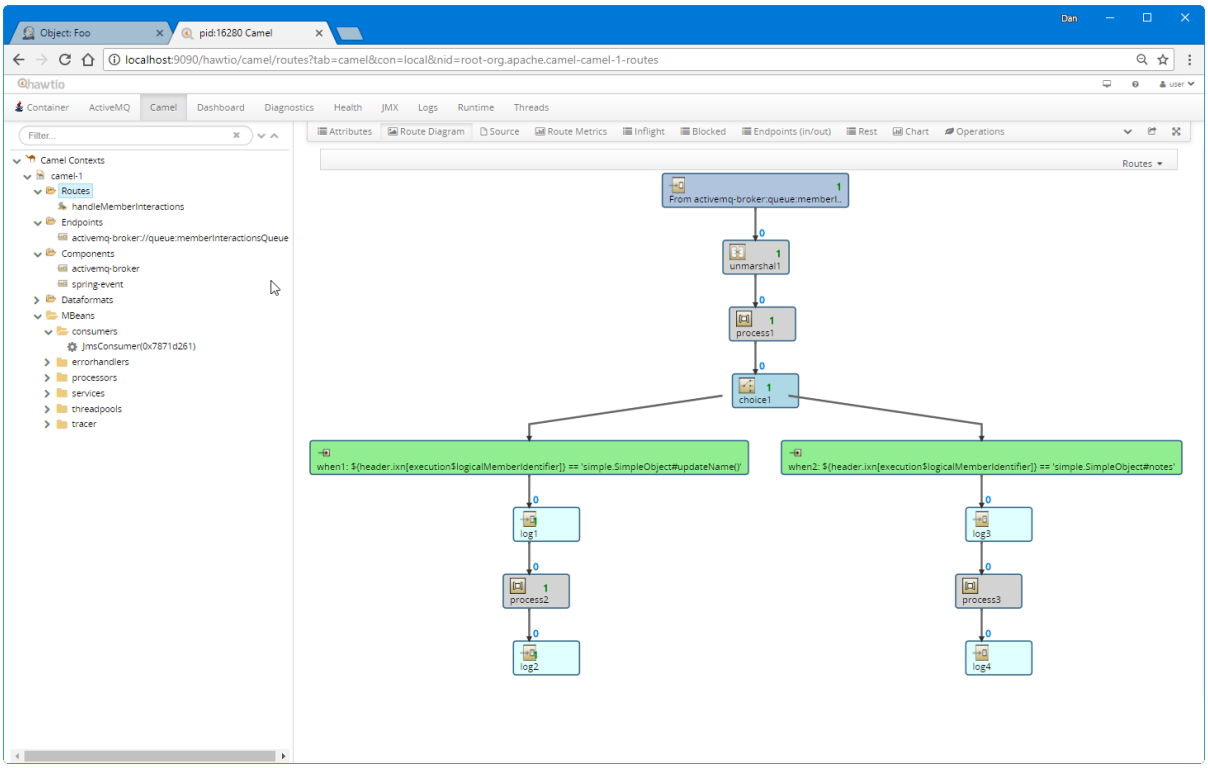


with the port set to `8080` and the path set to `/api/jolokia`.

Connecting this time should show a (new) "Camel" tab:



which also shows a diagram of the route (defined in `example-embeddedcamel-config.xml`):



If the `SimpleObject#updateName(…)` action is invoked, then this matches the left hand branch. The diagram (above) actually indicates this, with each node indicating the number of times it has been traversed. The "process2" component corresponds to this definition in the route:

```
<camel:process ref="attachCanonicalDtoUsingRestfulObjects"/>
<log message="DTO:
${header['org.incode.domainapp.example.canonical.SimpleObjectDto']}"/>
```

where `attachCanonicalDtoUsingRestfulObjects` is an alias to this bean:

```
<bean id="attachCanonicalDtoUsingRestfulObjects"
      class="domainapp.example.embeddedcamel.processor.AttachSimpleObjectDto"
      init-method="init">
    <!-- see propertyPlaceholderConfigurer-config.xml -->
    <property name="base" value="${restful.base}"/>
    <property name="username" value="${restful.username}"/>
    <property name="password" value="${restful.password}"/>
</bean>
```

The `AttachSimpleObjectDto` bean:

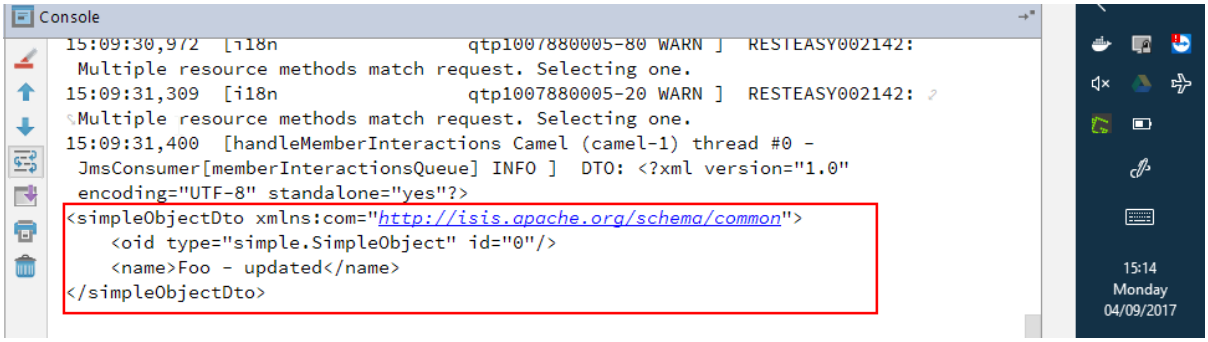
- logs using the `StatusMessageClient` utility (provided by the publishmq module) to log a message using the REST API
- calls the REST API to obtain the DTO for `SimpleObject`
- logs a further message using `StatusMessageClient`
- if successful, attaches the DTO retrieved to the message:

```
final SimpleObjectDto entity = response.readEntity(SimpleObjectDto.class);
inMessage.setHeader(SimpleObjectDto.class.getName(), entity);
```

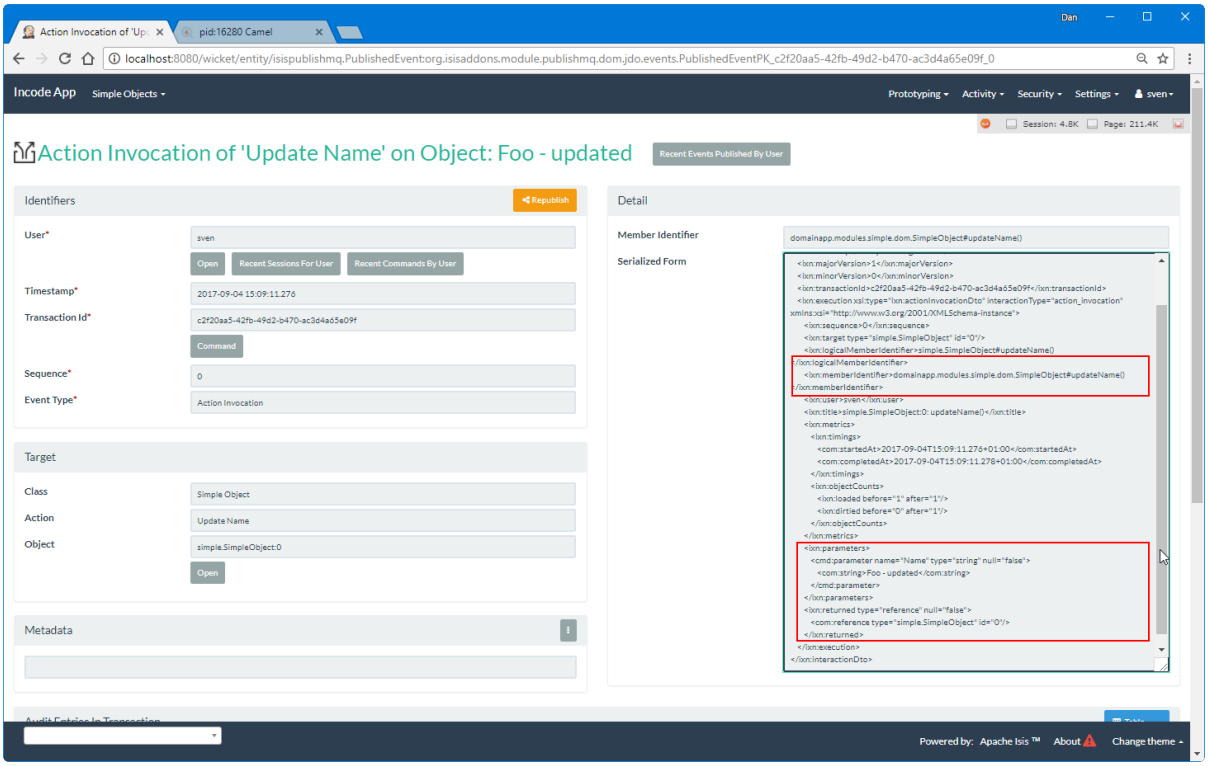
The last statement in the Camel route is:

```
<log message="DTO:
${header['org.incode.domainapp.example.canonical.SimpleObjectDto']}"/>
```

and thus in the console we see the DTO being logged:



Back in the Apache Isis application we can use the *Activity* menu to search for the persisted published event:



This object also has a collection of associated status messages logged by the Camel processor: