F4: External Integrations

Introducing Camel
What is Camel?

Good question. See: [http://camel.apache.org/what-is-camel.html](http://camel.apache.org/what-is-camel.html)

Too many buzzwords - what exactly is Camel?

Okay, so the description above is technology focused. There's a great discussion about Camel at [Stack Overflow](http://stackoverflow.com).

So really you want see this: [http://stackoverflow.com/questions/8845186/what-exactly-is-apache-camel](http://stackoverflow.com/questions/8845186/what-exactly-is-apache-camel)
In short...

- Camel is a framework for creating small message based applications... and then some.
- Camel formalizes working with messages so well it can be described in multiple formats: Java, Spring/Blueprint XML, and Scala.
- Camel is all the code you should not have to write in order to work with queues, files, databases, RESTful APIs, common data formats, command line utilities, etc... in a consistent and reliable manner.
Camel can run...

- As a stand-alone Java application
- In a servlet container like Tomcat or Jetty
- In an OSGi runtime such as Karaf
What is OSGi?

- Open Service Gateway Initiative

- Framework for modularizing and deploying Java applications
  - Hot deployment
  - Automatic reloading of configuration
  - Sophisticated dependency resolution
  - XML scripting for complex deployments (features)
Hot Deployment

Bundles can be started, stopped, updated, etc… at runtime!

In other words:
YOU DO NOT HAVE TO RESTART YOUR SERVER TO UPDATE CODE OR CONFIGURATION
Terminology

● Apache Camel
  ○ Endpoints
  ○ Components
  ○ Messages
  ○ Routes

● Apache Karaf -- OSGi
  ○ Bundles
  ○ Features
Available Camel Components

http://camel.apache.org/components.html

● ActiveMQ
● AWS SQS
● DropBox
● System calls
● Local files
● FTP

● HTTP resources
● LDAP
● SMTP
● SQL
● Twitter
● etc, etc, etc
Hands-On: Gitting the Examples

> vagrant ssh  
  or:

> ssh -p 2222 vagrant@localhost  
  password = vagrant
Hands-On: Inspect features

> /opt/karaf/bin/client
>> feature:list | grep fcrepo

fcrepo-camel
fcrepo-indexing-triplestore
fcrepo-audit-triplestore
fcrepo-indexing-solr
fcrepo-reindexing
fcrepo-fixity
Hands-On: Helpful Commands

>> feature:install fcrepo-audit-triplestore
>> feature:stop <whichever>
>> camel:route-list
>> bundle:list | grep fcrepo
>> ctrl-d
Hands-On: Watch the log

In a new vagrant ssh terminal:

> sudo tail -f /opt/karaf/data/log/karaf.log
Hands-On: Indexing in triplestore

http://localhost:8080/fuseki
Hands-On: Indexing in triplestore

```sparql
select * where {
}
```
Hands-On: Indexing in triplestore

PREFIX ldp: <http://www.w3.org/ns/ldp#>
PREFIX ebucore: <http://www.ebu.ch/metadata/ontologies/ebucore/ebucore#>

select * where {
    ?o ebucore:hasMimeType ?m
}

Hands-On: Indexing in triplestore

Audit

```sparql
prefix premis: <http://www.loc.gov/premis/rdf/v1#>
prefix xsd: <http://www.w3.org/2001/XMLSchema#>

select ?s ?d where {
  ?s premis:hasEventDateTime ?d .
  FILTER (?d > "2015-10-06T04:21:14Z"^^xsd:dateTime)
}
```
Hands-On: Indexing in Solr

http://localhost:8080/solr
Hands-On: Indexing in Solr
Hands-On: Reindexing - prep

> sudo service tomcat7 stop
> sudo rm -rf /etc/fuseki/databases/test_data/*
> sudo service tomcat7 start
Hands-On: Reindexing

> curl -XPOST localhost:9080/reindexing/cover
-H"Content-Type: application/json" -d
'"activemq:queue:triplestore.reindex"']'
> curl -XPOST localhost:9080/reindexing/cover -H "Content-Type: application/json" -d '['"activemq:queue:fixity"]'

> less /tmp/fixityErrors.log
Bonus Round
Git the examples

-- from within vagrant --

> cd
> git clone https://github.com/awoods/fcrepo-camel-workshop.git
Hands-On: Install a Feature

>> feature:install camel-exec

This installs the camel-exec feature...
  which we will use later on for executing command-line utilities.
1 - Make our own “feature”

*Use case:*

- Whenever a JPEG image is ingested in Fedora, generate a thumbnail
01: Hello World

<camelContext id="helloWorld">
  <route id="timerToLog">
    <from uri="timer:foo?period=5000"/>
    <setBody>
      <simple>Hello Whirled!</simple>
    </setBody>
    <to uri="log:demo"/>
  </route>
</camelContext>
Deploy the first route

> sudo cp /home/vagrant/fcrepo-camel-workshop/01-HelloWorld.xml /opt/karaf/deploy/
Routes

- Processing pipelines, with beginning and ending points
- Begin with a ‘from’ uri
  - A file
  - A queue
  - A timer
  - Another route
- Other routes are called by using ‘to’ and a uri
  - <to uri="direct:myRoute"/>
Camel Components / Endpoints

- From previous example
  - "timer:foo?period=5000"
- Prefixes in camel URIs are Components
- Components provide endpoints for communicating with other software
- Two types of endpoints
  - Producer
  - Consumer
- Options can be provided
Component: fcrepo-camel

- fcrepo-camel project provides an F4 component
- Used as a producer to interact with F4’s REST API
- URIs look like
  - fcrepo:hostname[:port]/resourceUrl[?options]
- Used as a consumer to handle messages published by F4
02: Responding to Fedora events

<route id="thumbnailRouter">  
  <from uri="activemq:topic:fedora"/>  
  <log message="GOT A MESSAGE FROM FEDORA"/>  
  <log message="HEADERS: ${headers}"/> 
</route>
Anatomy of a Message

● Body
  ○ The main content of message. Can be any text or binary data type:
    ■ html, json, xml, etc…
    ■ image, audio, video, etc…

● Headers
  ○ Key/Value properties for the message
    ■ HTTP: Accept, Content-Type, etc…
    ■ JMS: Timestamp, Expiry, Destination, etc…
Fedora Headers

- org.fcrepo.jms.baseURL
- org.fcrepo.jms.identifier
- org.fcrepo.jms.eventType
- org.fcrepo.jms.properties
- org.fcrepo.jms.timestamp
- and more…
03: Getting RDF from Fedora

```xml
<route id="thumbnailRouter">
  <from uri="activemq:topic:fedora"/>
  <to uri="fcrepo:localhost:8080/fcrepo/rest"/>
  <log message="${body}"/>
</route>
```
Watch out!

If you are scripting in XML, remember that you are still in XML! ...and have to escape your xml entities.

So things like:

- $x < y$
- $this && that$

Become:

- $x \lt; y$
- $this \&\& that$
04: Filtering using headers

<from uri="activemq:topic:fedora"/>
<filter>
  <simple>
    ${header[org.fcrepo.jms.eventType]} != 'http://fedora.info/definitions/v4/repository#NODE_REMOVED' 
    &
    ${header[org.fcrepo.jms.properties]} contains 'http://fedora.info/definitions/v4/repository#hasContent'
  </simple>
  <log message="Got an upsert event with content!"/>
</filter>
05: Filtering on body content

```
<route id="thumbnailRouter">
    <from uri="activemq:topic:fedora"/>
    <to uri="fcrepo:localhost:8080/fcrepo/rest"/>
    <setProperty propertyName="mimetype">
        <xpath>/rdf:RDF/rdf:Description/ebucore:hasMimeType/text()</xpath>
    </setProperty>
    <filter>
        <simple>${property.mimetype} == 'image/jpeg'</simple>
        <log message="JPEG!"/>
    </filter>
</route>
```
06a: Getting Non-Rdf Content

```xml
<route id="thumbnailRouter">
  <from uri="activemq:topic:fedora"/>
  <to uri="fcrepo:localhost:8080/fcrepo/rest?metadata=false"/>
  <log message="${body}"/>
</route>
```
06b: Generating a Thumbnail

```xml
<route id="thumbnailGenerate">
    <from uri="activemq:topic:fedora"/>
    <to uri="fcrepo:localhost:8080/fcrepo/rest?metadata=false"/>
    <to uri="exec:convert?args=-thumbnail 100x100 - png:-"/>
    <log message=""THUMBNAIL: ${body}""/>
</route>
```
Put it all together

- Ingest a JPEG file
- Derivative thumbnail at /tmp/cover/<> .png
- View thumbnail:
  
  > cp /tmp/cover/<> .png /vagrant

- View <> .png outside of vagrant
Success!