GETTING STARTED WITH C24 AND CAMEL

Iain Porter
Solutions Architect
C24 Technologies
2015

This is a guide to using the C24-Camel-Getting-Started project. This guide will walk through the various parts of the sample project and then discuss how you can start developing your own project.

The project can be found here:
https://github.com/C24-Technologies/c24-camel-getting-started

It can be considered a companion project to:
https://github.com/C24-Technologies/c24-api-gettingstarted

and:
https://github.com/C24-Technologies/c24-spring-getting-started

The project can be used as a starting point for building a new C24 Camel project. It also contains useful how to's and testing examples to help you get started with building robust applications that utilise the power of C24 message transformation with Camel's Enterprise Integration framework.

The main components are:

- C24 runtime libraries
- C24 Maven plugin for deploying data models and transformations
- C24 Camel
- Camel
- Spring Framework
- Spring Boot

See https://github.com/C24-Technologies/c24-camel for a reference guide to C24 Camel.

Building the project

Once you have cloned the project from git you can build it using the maven command:

> *mvn package*

This will build and deploy the data models and run the tests.

Outline of the Sample Project

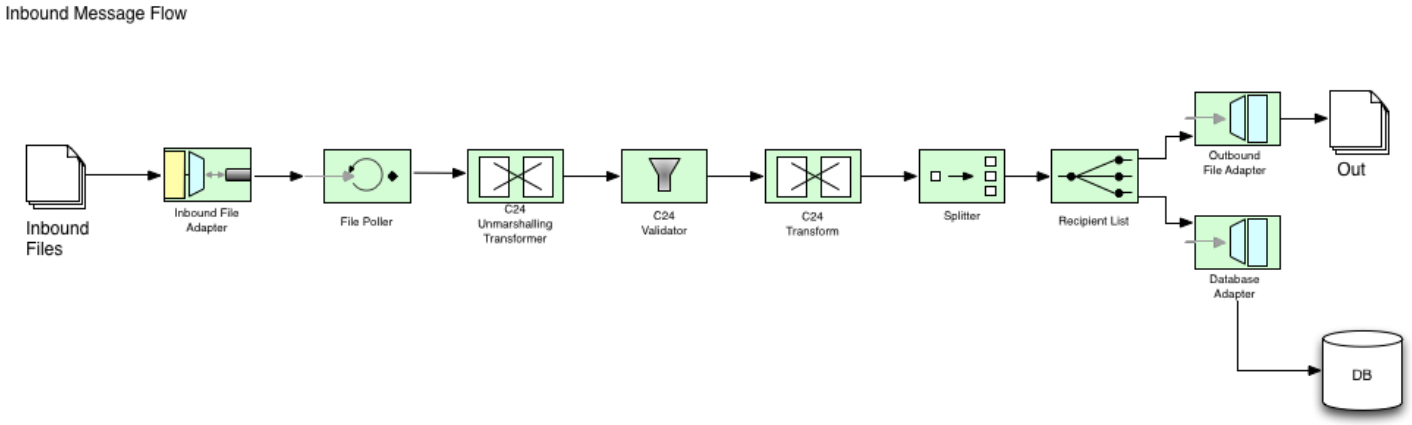
The project is broken into two parts:

1. Inbound - A File System poller listens for incoming files in a configured directory. Files are parsed into a C24 model. The model is validated and then transformed. A router routes the message to two flows. One route marshall's the messages to XML and writes them out to a directory. The other route also marshall's the messages to XML and persists them to a database.

2. Outbound - A Jdbc poller listens for updated rows in a database table. Result sets are unmarshalled into a C24 model. The model is validated and transformed. The transformed object is marshalled to JSON and pushed onto a JMS Queue. A copy of the message is written out to an archive directory.

The camel routes can be found in src/main/resources/camel-context.xml.
There are integration tests in the test directory that show various examples on how to test the message flows.

Inbound Message Flow

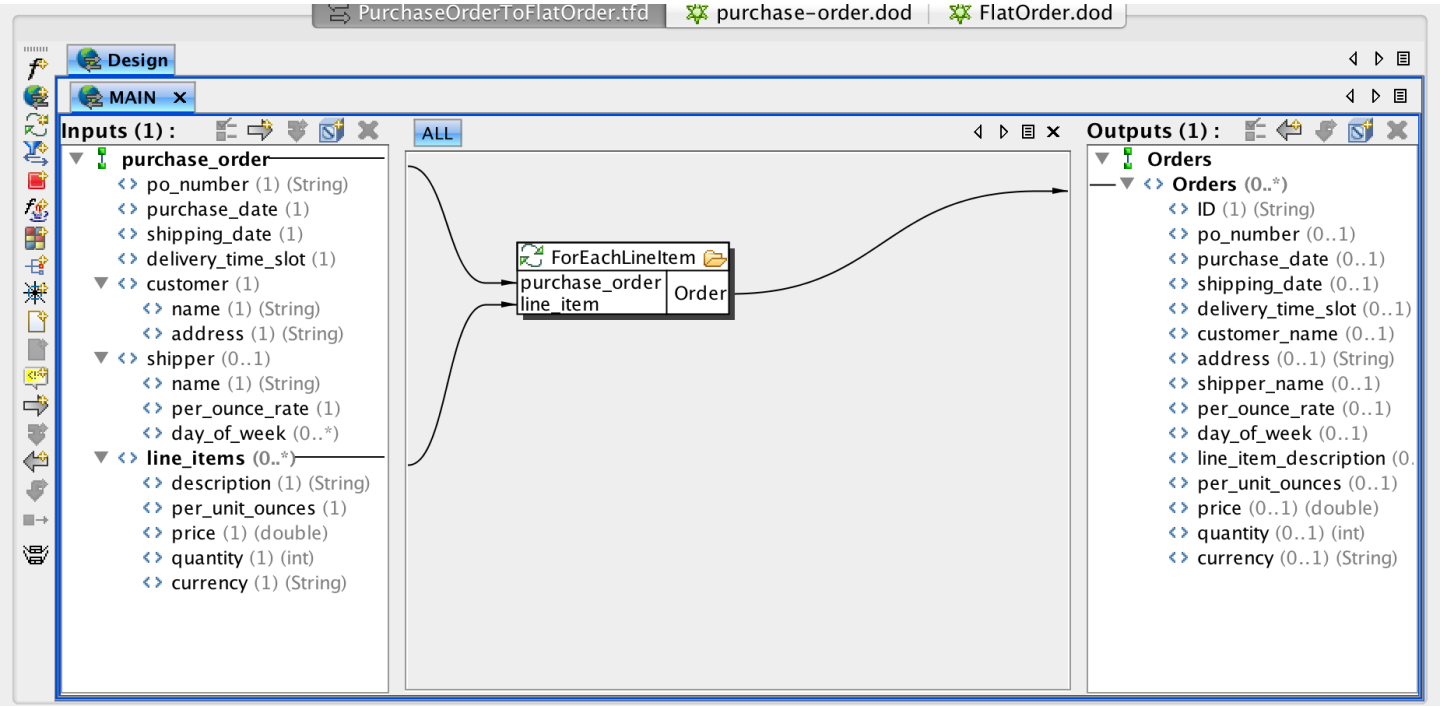


The Data Models and Transform

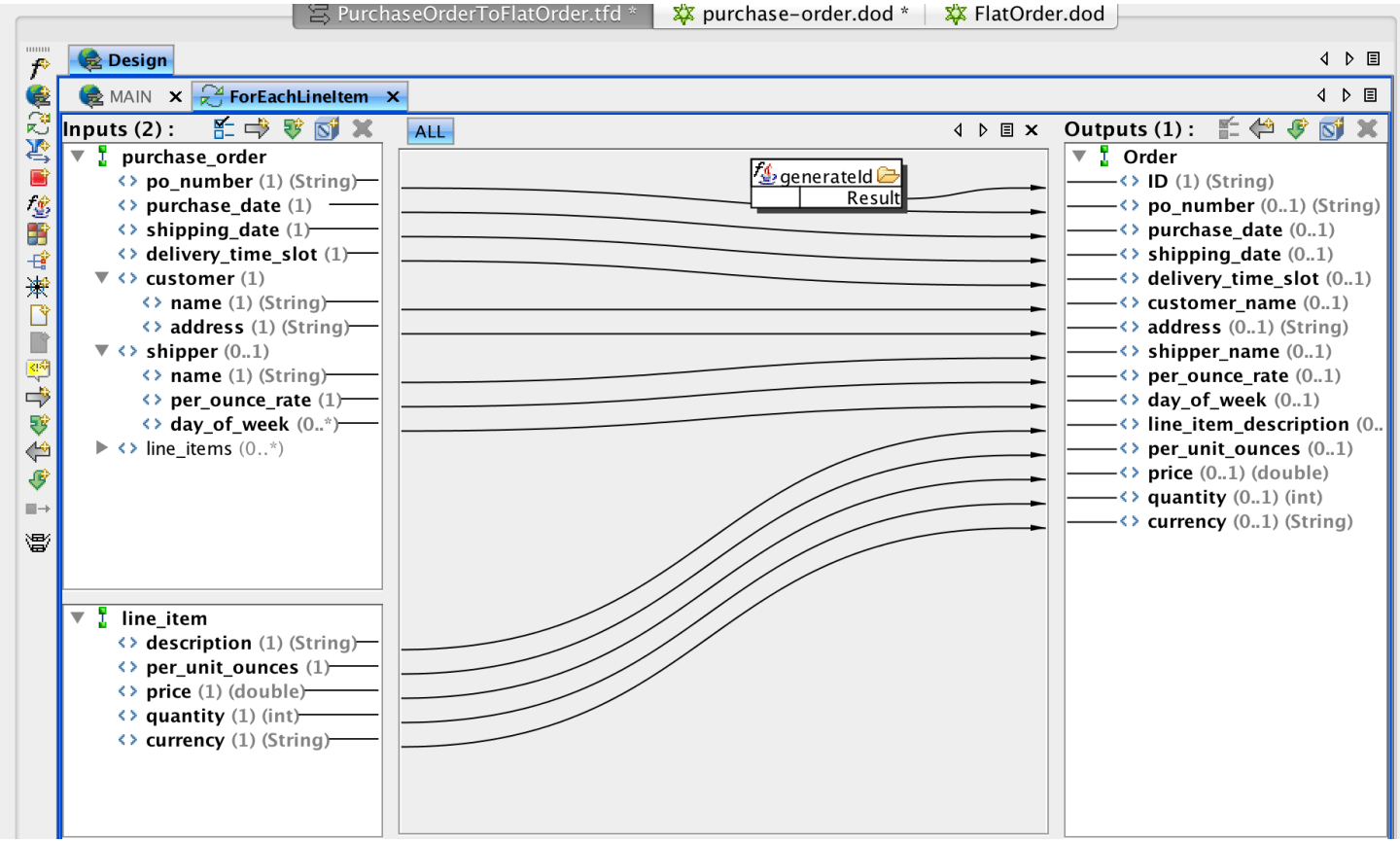
This sample project uses a Purchase Order example. The model is built in studio with XML as the default input and output.

The transform will take a purchase order and produce a flat file purchase order for each line item in the purchase order. This means that if there are three line items in a single purchase order then three flat file records will be created.

Purchase Order to Flat File transform



Transform detail



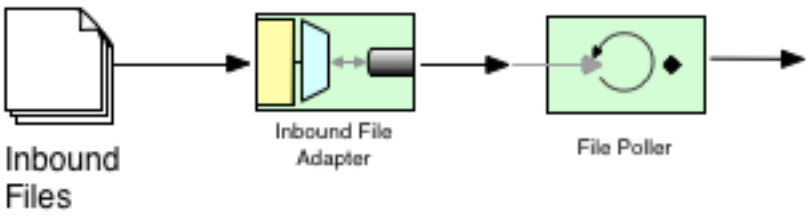
Deploying the models and transforms using the C24 Maven plugin

The java bindings for the data model and transforms are built dynamically using the C24 maven plugin. More details about the plugin can be found here:

http://dev.c24.biz/C24mavenPlugin/4.7.0/usage.html

They are built as part of the generate-sources phase. The model classes are in src/main/C24. The generated source classes will be written to target/generated-sources/src/main/java

File Polling

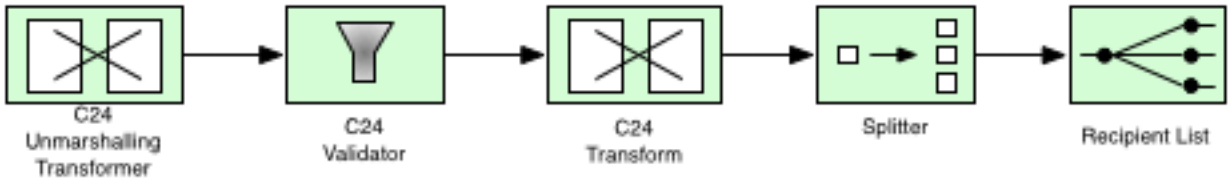


A File Poller is configured to look for files in a directory (see biz.c24.io.route.InboundFilePollingRoute)

When a file is detected it is read and the contents passing on to the next channel for parsing and transformation.

The file poller is transactional. If the message flow is successful the original file is moved to a processed directory. If the transaction is rolled back the file is moved to a failed directory.

Message Parsing and Transformation

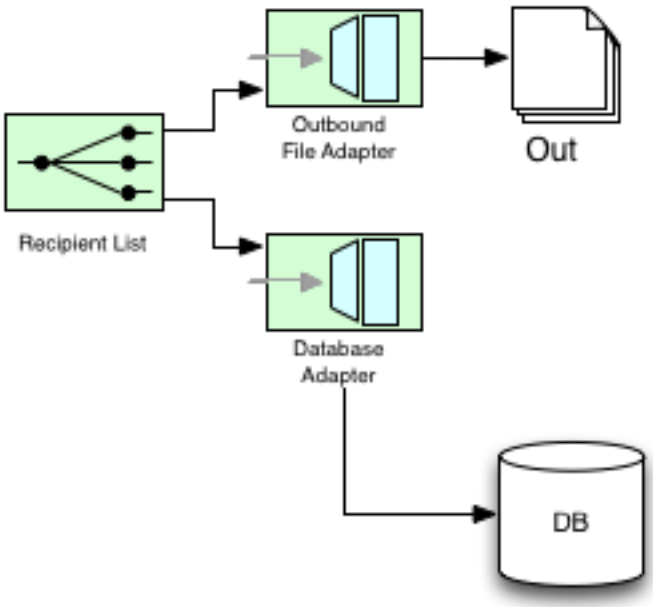


The inbound file is unmarshalled into a C24 ComplexDataObject. If successfully parsed it is then validated, before being transformed to the Flat File model. (see biz.c24.io.route.InboundMessageHandlingRoute)

A flat file message will be created for each line item in the purchase order so a splitter is used to process each message individually.

Each message is sent to a recipient list router for further processing.

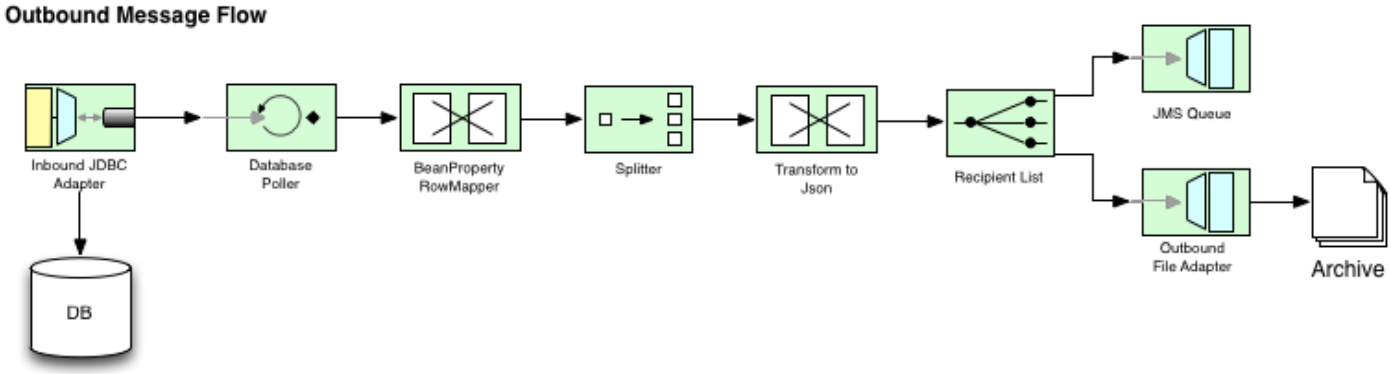
Message Routing



Each message is sent to a channel for persisting to a database and a channel for writing out to a directory. (see biz.c24.o.route.InboundMessagePersistenceRoute and biz.c24.io.route.InboundMessageWriteRoute)

When persisting the order to the database a biz.c24.io.spring.integration.jdbc.CdoSqlParameterSourceFactory is used to auto map values in the ComplexDataObject to database columns. This is a convenience class that is useful when the data model and database tables are similar.

Outbound Message Flow



The outbound flow is a lot simpler in terms of message handling.

A database poller polls for rows that match processed = false.

Results sets are converted to Orders by using a custom processor to map rows to Order instances.

If the poller returned more than one record in the result set a splitter breaks them into individual messages and passes on to the marshal channel.

The marshal component is shown to highlight the use of C24 Sink Factory component to marshall to format of choice; in this case Json.

The marshalled message in Json format is then sent to a JMS queue for downstream processing.

A copy of the message is written out to an archive directory.

Running the Application

Spring boot is used to bootstrap the spring contexts.

The class biz.c24.io.Application references the context file src/main/resources/application-context.xml as the context to load on startup.

To start you can either invoke the maven plugin:

```
> mvn spring-boot:run
```

or execute the jar:

```
>  java -jar target/c24-camel-getting-started-1.0.0.jar
```

On startup the project defaults will be loaded from application.properties. These can be overridden by adding a properties file and passing a system argument that point to it (-DPROPERTY_FILE_LOCATION). It can be used to specify properties specific to the environment such as JDBC properties, JMS properties, etc.

Create a properties file and add the following properties to it:

- inbound.read.path=
- inbound.processed.path=
- inbound.failed.path=
- inbound.write.path=
- outbound.archive.path=

Set the value to the absolute path of the directories that you create for each property.

Start the application again by referencing the property file. This property file will take precedence over the property file that is in src/main/resources/properties

> java -DPROPERTY_FILE_LOCATION=[your file location path] -jar target/c24-camel-getting-started-1.0.0.jar

Time to run a message through by copying src/test/resources/data/purchase-order.xml to your read directory specified in inbound.read.path

At the end of the inbound and outbound flow the directories should have the following entries:

- inbound.read.path - EMPTY
- inbound.processed.path - A copy of purchase-order.xml
- inbound.failed.path - EMPTY
- inbound.write.path - 3 purchase-order files with the sequence number appended
- outbound.archive.path - 3 json files

Creating your own project

The getting started project can also be used as a template for starting your own project. All of the maven dependencies are already in place as well as plenty of examples to use and customise for your own needs.

Adding an external Database and JMS provider

The project by default uses an embedded H2 database and embedded ActiveMQ broker. If you want to use a different datasource you can follow these steps:

1. Add a profile in the maven POM. There is already a mysql and sqlserver profile. The caveat for using sqlserver is that you will have to deploy the sqlserver driver to your local maven repository. The POM file has an example.
2. Add the jdbc properties to your properties file (see biz.c24.o.configuration.ExternalDatabaseConfiguration for the properties you will need) That class will be invoked for the spring profiles mysql or sqlserver. If no profile is specified the default profile loads biz.c24.io.configuration.EmbeddedDatabaseConfiguration which is a H2 instance.

To build and run it you would execute the following (using mysql as an example):

> mvn -Pmysql package

> java -DPROPERTY_FILE_LOCATION=[your file location path] -Dspring.profiles.active=mysql -jar target/c24-camel-getting-started-1.0.0.jar

Follow similar steps for adding a different JMS provider.

FOR MORE INFORMATION

The C24 Camel landing page - https://www.c24.biz/c24-io-with-fuse.html

C24 Camel git repository - https://github.com/C24-Technologies/c24-camel

C24 API Getting started sample project - https://github.com/C24-Technologies/c24-api-gettingstarted

Getting started with Camel and C24 - https://github.com/C24-Technologies/c24-camel-getting-started

C24 maven plugin - http://dev.c24.biz/C24mavenPlugin/4.7.0/index.html