****

Contents

[1. Introduction 1](#_Toc417395944)

[2. Prerequisites 2](#_Toc417395945)

[3. How to setup Camel component in MDW 2](#_Toc417395946)

[a. ServiceMix/Fuse 2](#_Toc417395947)

[b. Tomcat 2](#_Toc417395948)

[4. Setup Eclipse Environment 2](#_Toc417395949)

[a. Setup Git Repository (For Tomcat Container) 2](#_Toc417395950)

[b. Import Example Project (For Fuse/ServiceMix) 5](#_Toc417395951)

[c. Import Camel demo package (For Fuse/ServiceMix and Tomcat) 6](#_Toc417395952)

[5. Run Examples 8](#_Toc417395953)

[a. Run Service Process through a Camel Route 8](#_Toc417395954)

[i. Register Your Service through a Camel Route 8](#_Toc417395955)

[ii. How to create a spring context 8](#_Toc417395956)

[iii. Configure to call Service Process 9](#_Toc417395957)

[iv. Invoke Your Service through REST 10](#_Toc417395958)

[b. Invoke a Camel Route from a Process 13](#_Toc417395959)

[i. How to create Camel Route 13](#_Toc417395960)

# Introduction

MDW provides listeners for many common protocols out-of-box, such as JMS, Tibco Bus, RESTful webservice, MQ Series, MDW SOAP web service, EJB. Besides that Camel can also be integrated with MDW.

[Apache Camel](http://camel.apache.org) is an "integration framework" that enables you to link together services through declarative route definitions. As an alternative to the traditional MDW External Event Handler registration described above, you may choose to expose your service through a Camel route. The advantages of this include the advanced [transport](http://camel.apache.org/transport.html) and [filtering](http://camel.apache.org/message-filter.html) capabilities that Camel supports. For example, Camel goes beyond the basic XPath message filtering provided by MDW event handlers to enable filtering according to request header values or even by complex scripting language expressions.

You can read more on this topic at <http://lxdenvmtc099.dev.qintra.com:10001/MDWHub/doc/MDWCamelIntegration.html>

Also there is a recorded session at [Camel Integration](http://cshare.ad.qintra.com/sites/MDW/Training/Forms/AllItems.aspx?RootFolder=%2Fsites%2FMDW%2FTraining%2FCamel%20Integration)

All the examples in this cookbook are available in Git repo. You will see in later sections on how to get to that.

# Prerequisites

Either Tomcat or ServiceMix/Fuse container is setup with MDW 5.5.13 or later version with latest version of MDW Designer Plug-In (8.5.3 +) by following corresponding cookbooks.

# How to setup Camel component in MDW

## ServiceMix/Fuse

Since mdw-camel is an optional feature, you'll need to install it by typing the following command at the karaf command prompt:

**features:install mdw-camel**

## Tomcat

Camel is included in standard mdw.war. No additional setup is required

# Setup Eclipse Environment

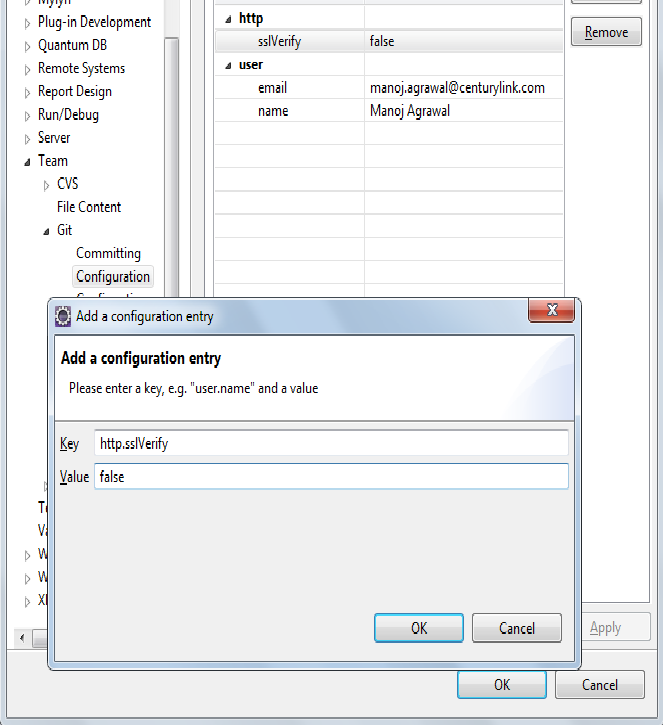
There are multiple ways to run camel examples in this cookbook, choose based on your environmental setup:

## Setup Git Repository (For Tomcat Container)

* + 1. In Eclipse Got to Windows-->Preferences
    2. Team-->Git-->Configuration

Add Entry following

* + - 1. Key = http.sslVerify
      2. Value=false

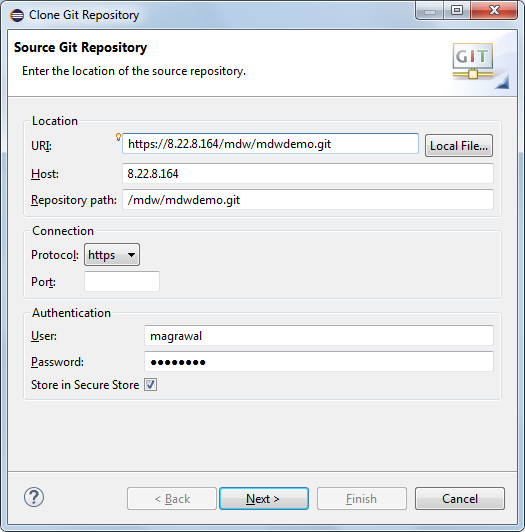


Open Eclipse Git Perspective:

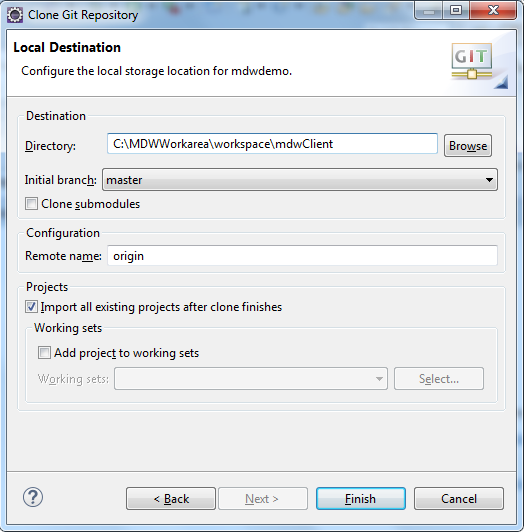
In Windows->Show View->Other-> Git->Git Reposnitory Perspective->

Click on “Clone a Git Repository and add the clone to this view”.

1. Enter URI : <https://8.22.8.164/mdw/mdwdemo.git> and
2. User Name / password : mdw/ldap\_0123 to connect Get Repository



1. Click Next



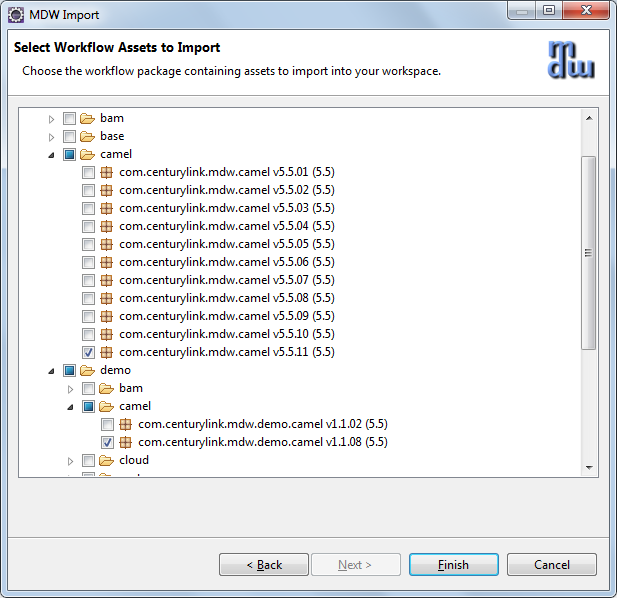
1. Click Finish
2. Once the project is downloaded, update mdw.properties for any changes related to DB
3. Update Tomcat configuration to point to mdw.properties in mdwdemo folder.
4. Run Maven build with package target and check if mdwdemo.jar is there under src/main/workflow/com/centurylink/mdw/tests/java
5. Start Tomcat Server

## Import Example Project (For Fuse/ServiceMix)

* + 1. Copy the “MDWCamelDemo - Vcs - Fuse -MDW 5.5.zip.zip” file from [\\eldnp1515dm4.ad.qintra.com\union\_station\IT\MDW\Camel](file:///\\eldnp1515dm4.ad.qintra.com\union_station\IT\MDW\Camel)
    2. Unzip it
    3. Then import into eclipse using Import Existing Project
    4. Run Maven build with package target
    5. Check if the bundled deployed and active in your fuse instance directory

## Import Camel demo package (For Fuse/ServiceMix and Tomcat)

* + 1. Right-click on workflow project in Process Explorer and select Import > Package. Choose the "Discover" option and leave the repository location as the default.
    2. Select latest versions of following packages
       1. com.centurylink.mdw.camel
       2. com.centurylink.mdw.demo.camel



* + 1. Click Finish, and the packages will be downloaded and become visible in your Process Explorer project tree.
    2. If using Fuse/ServiceMix then in orderContext.spring replace following with your instance info
       1. <propertyPlaceholder id="properties" location="file:${karaf.home}/instances/camel/etc/com.centurylink.mdw.cameldemo.cfg"/>
       2. Also you need to add following entry in your com.centurylink.mdw.cfg file

mdw.application.config.pids=com.centurylink.mdw.cameldemo

* + - 1. Create a file named com.centurylink.mdw.cameldemo.cfg in your etc dir and following in that file

mdw.cameldemo.jms.connection.pool=jmsConnectionPool

* + 1. In order to make use of MDW Camel support, you'll need to add the following dependency to your build.gradle file:

|  |
| --- |
| compile group: "com.centurylink.mdw", name: "mdw-camel", version: mdwVersion |

… or in your pom.xml:

|  |
| --- |
| <dependency>  <groupId>com.centurylink.mdw</groupId>  <artifactId>mdw-camel</artifactId>  <version>${mdw.version}</version>  </dependency> |

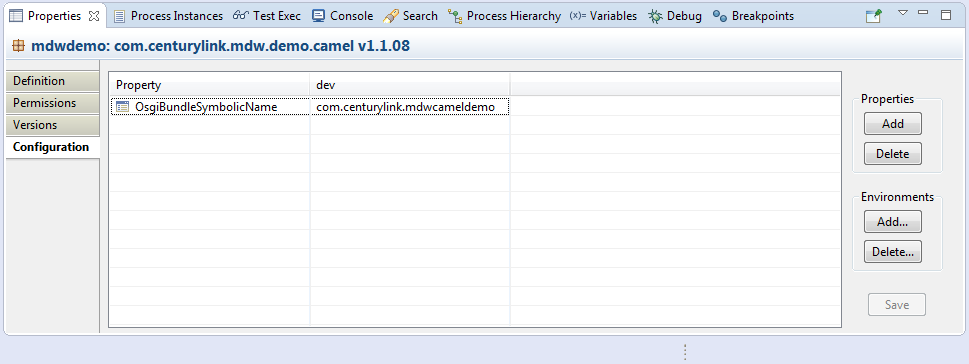
Also, since all the wiring magic is done by reflection, you'll need to explicitly add the following imports under the Import-Package section:

|  |
| --- |
| instruction "Import-Package",  ...  "com.centurylink.mdw.camel",  "com.centurylink.mdw.camel.plugins.tomcat",  ... |
|  |
|  |

.. or in your pom.xml:

|  |
| --- |
| <Import-Package>  ...  com.centurylink.mdw.camel,  com.centurylink.mdw.camel.plugins.tomcat,  ... |

* + 1. Then perform a Gradle or Maven build to pick up this new dependency.
    2. In order for the framework to identify the correct parent Spring app context to associate with your asset, you'll need to set a configuration parameter to tie your workflow package to your OSGi bundle. Right-click on your workflow package in Process Explorer and select Show Properties. Select the Configuration property tab and under Environments click the Add button and enter the name "dev". Add a property named OsgiBundleSymbolicName and under dev assign the value from the symbolicName in your project's build.gradle or pom.xml build file.



# Run Examples

## Run Service Process through a Camel Route

### Register Your Service through a Camel Route

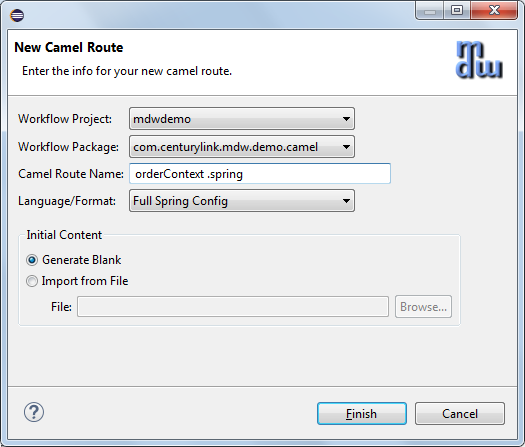
Among other ways, a Camel route can be defined using a Spring configuration file.

When you ran the MDW Eclipse wizard to create your workflow project back in Project Setup in ServiceMix/Fuse Cookbook, a Spring config file named bundle-context.xml was generated for you under src/main/resources/META-INF/spring. This file gets embedded into the META-INF directory of your workflow bundle. You could declare your MDW Camel route in bundle-context.xml, but then you'd be missing out on one of the major benefits of using MDW: dynamicism. Instead, we'll create a separate child Spring configuration as a workflow asset that can be changed without requiring that your bundle be redeployed. In fact, Spring configuration assets in MDW can be used to inject any Spring-enabled entity into your OSGi bundle context.

### How to create a spring context

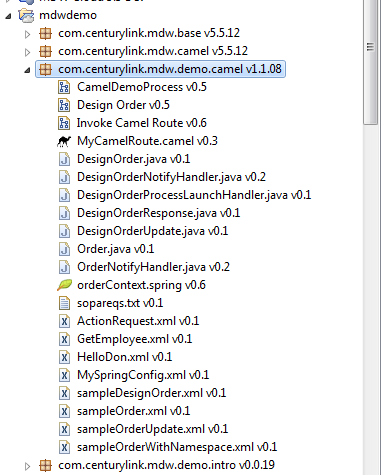
Below steps show how a workflow asset called orderContext.spring was created in com.centurylink.mdw.demo.camel package.

Right-click on your workflow package in Process Explorer view, and select New > Camel Route. Name your route and make sure and select "Full Spring Config" for the Language.



### Configure to call Service Process

In this example we will call a service process called “FindEmployee” using a came route. Open MDW Designer Perspective and locate orderContext.spring under package com.centurylink.mdw.demo.camel



Locate following text in orderContext.spring

<!-- MDW Web Service (REST) -->

<route>

<from uri="mdw:REST/GetEmployee"/> <!-- specify MasterRequestID HTTP header -->

<to uri="mdw:process?name=FindEmployee" pattern="InOut" />

</route>

The mdw:REST/GetEmployee route declared within this camelContext tells the MDW Camel component to register a REST-based web service and launch FindEmployee (process is in package: com.centurylink.mdw.test) when invoked. In order for the "mdw:" URI pattern to be supported, the MDW component bean is declared. Notice that with the "from" URI in an MDW Camel route you have the option of specifying a specific endpoint URL path (REST/ GetEmployee), in contrast to the standard MDW External Event handler which all use the same endpoint.

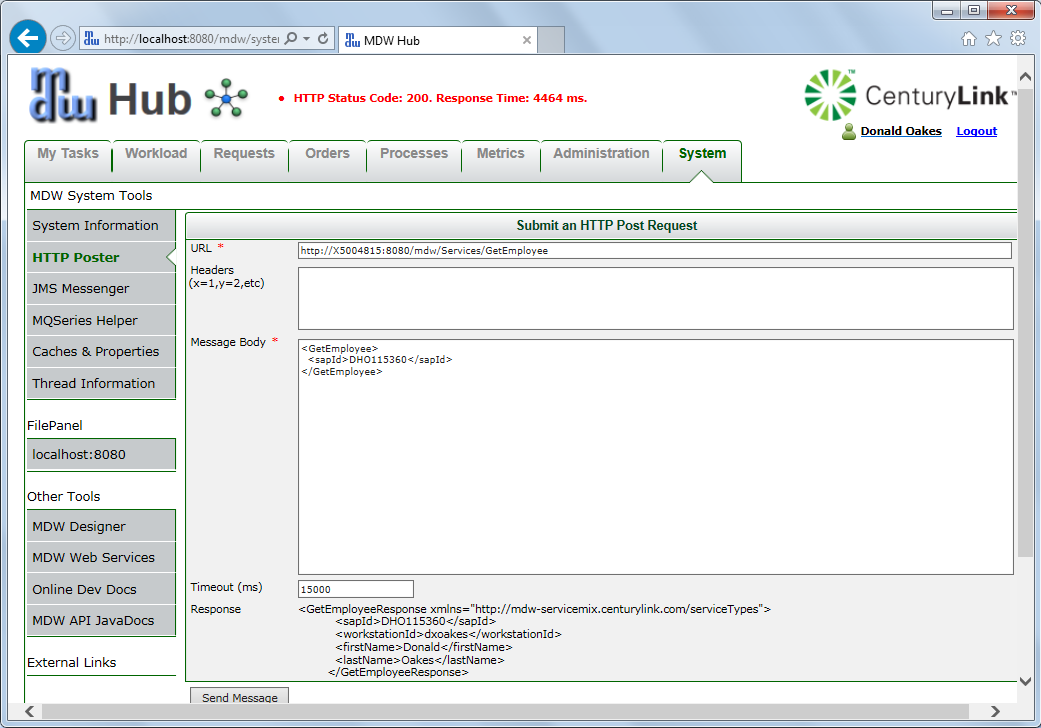
### Invoke Your Service through REST

As before with your SOAP request, you can use MDW Hub's HTTP Poster facility to submit your REST request to the server. The URL should include the path you assigned in the "from" URI in your Camel route Spring asset.

<GetEmployee>

<sapId>DHO115360</sapId>

</GetEmployee>



The URL for HttpPoster and Request for Tomcat is

<http://localhost:8080/mdw/system/httpHelper.jsf>

<http://localhost:8080/mdw/Services/GetEmployee>

The URL for HttpPoster and Request for Fuse/ServiceMix is

<http://localhost:8181/MDWHub/system/httpHelper.jsf>

<http://localhost:8181/MDWWeb/Services/GetEmployee>

You can see the response includes all the LDAP information of DHO115360 which was returned by the called service process

You should see following logs in console:

[(i)20150417.16:42:49.339 ~14] Adding Spring Config Asset: 'mdw:spring/com.centurylink.mdw.demo.camel/orderContext.spring ' with ApplicationContext: OsgiBundleXmlApplicationContext(bundle=com.centurylink.mdwcameldemovcsfuse, config=osgibundle:/META-INF/spring/\*.xml): startup date [Fri Apr 17 16:42:36 MDT 2015]; root of context hierarchy

[(i)20150417.16:42:51.396 ~14] Registering ServiceHandler:Endpoint[mdw://SOAP/GetEmployee]

[(i)20150417.16:42:51.399 ~14] Registering ServiceHandler:Endpoint[mdw://REST/GetEmployee]

…

[(i)20150417.16:42:51.413 ~14] Registering ServiceHandler:Endpoint[mdw://REST/ActionRequest]

[(i)20150409.13:20:49.348 ~438] Starting MDW process 'FindEmployee v0.13' through Camel route.

[(i)20150409.13:20:49.349 p173252586.10812 m.1428607249348] Process started - FindEmployee/0.13

[(i)20150409.13:20:49.349 p173252586.10812 a1.19147] Activity started - Start

…

[(i)20150409.13:20:53.188 p173252586.10812 a5.19148] Activity completed - completion code 'false'

[(i)20150409.13:20:53.189 p173252586.10812 a3.19149] Activity started - Lookup

sapId: DHO115360

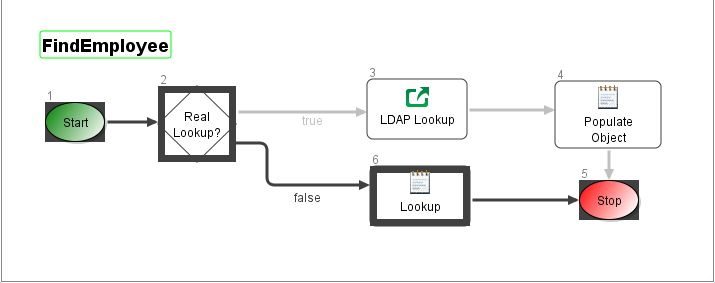
workstationId: null

…

[(i)20150409.13:20:53.501 p173252586.10812 a2.19150] Activity started - Stop

[(i)20150409.13:20:53.501 p173252586.10812 m.1428607249348] Process completed - FindEmployee completion code is null

You will see a process instance was created



## Invoke a Camel Route from a Process

You can also use MDW workflow to launch a Camel route. If you are using Camel Route inside a workflow process then you must define it as a Workflow Asset.  
Example : Look into the "Invoke Camel Route" process in com.centurylink.mdw.demo.camel package This process basically launches Camel Route from MDW process, this goes to the routing and at the end result will populate as part of the response variable.  
In this example the process uses asset driven (MyCamelRoute.camel) activity defined as part of the Camel Route Adapter activity.

<?xml version="1.0" encoding="UTF-8"?>

<route xmlns="http://camel.apache.org/schema/spring"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://camel.apache.org/schema/spring

http://camel.apache.org/schema/spring/camel-spring.xsd">

<from uri="mdw:workflow/this"/>

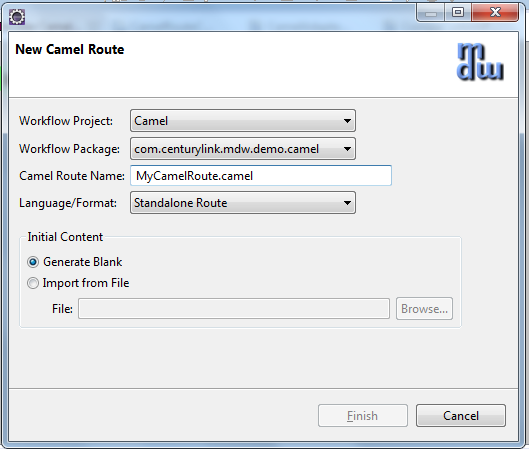
<to uri="file:c:/temp/data/outbox?fileName=mdw-camel-$simple{date:now:yyyyMMdd}.txt"

pattern="InOut" />

</route>

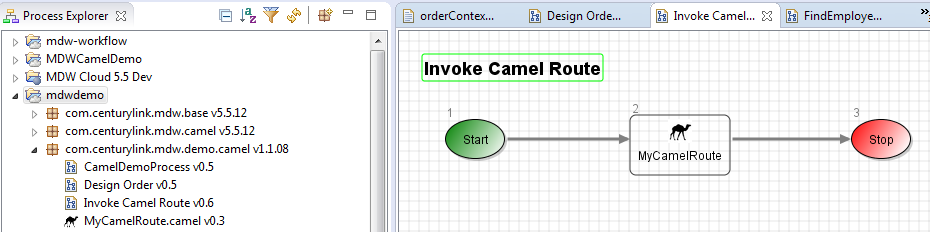
## How to create Camel Route

Right-click on your workflow package in Process Explorer view, and select New > Camel Route



Please note that the syntax "<from uri="mdw:workflow/this"/>" used in MyCamelRoute.camel means the instance of workflow which is running(i.e. instance of Invoke Camel Route process)

Locate “Invoke Camel Route” process and right click and select Run the process

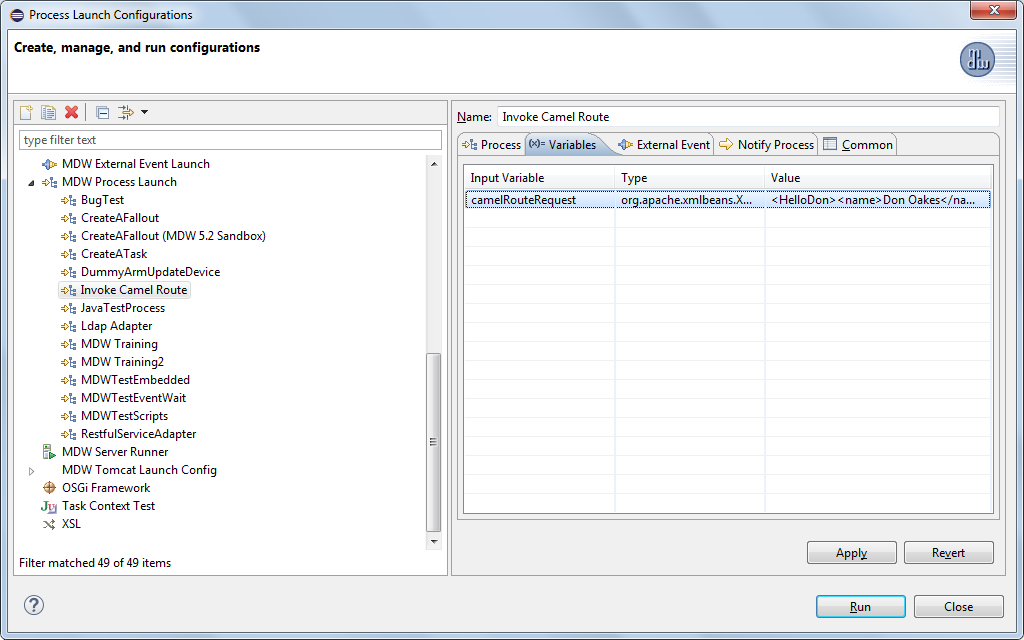
****

Enter following in Variables tab for camelRouteRequest:

<HelloDon>

<name>Don Oakes</name>

</HelloDon>

****

You should see following on console:

[(i)20150417.16:56:18.508 p107165898.10694 m.aa56486~150417-165616] Process started - Invoke Camel Route/0.6

[(i)20150417.16:56:20.038 p107165898.10694 a1.11258] Activity started - Start

[(i)20150417.16:56:20.082 p107165898.10694 a3.11259] Activity started - MyCamelRoute

[(i)20150417.16:56:20.094 ~203] Loading RoutesDefinition RuleSet based on key: com.centurylink.mdw.demo.camel/MyCamelRoute.camel v[0.1,1)~

[(i)20150417.16:56:20.100 ~203] Loading Camel Route 'MyCamelRoute.camel v3' with CamelContext: SpringCamelContext(camel-1) with spring id com.centurylink.mdw.common.spring.MdwOsgiBundleContext@41f8adf9

[(i)20150417.16:56:20.390 ~203] Registering WorkflowHandler: Endpoint[mdw://workflow/com.centurylink.mdw.demo.camel/MyCamelRoute.camel]

[(i)20150417.16:56:20.440 p107165898.10694 a2.11260] Activity started - Stop

[(i)20150417.16:56:20.470 p107165898.10694 m.aa56486~150417-165616] Process completed - Invoke Camel Route completion code is null

MyCamelRoute activity is configured to internally use MyCamelRoute.camel

When process finishes a file named mdw-camel-<date>.txt gets generated under c:/temp/data/outbox with a response from process populated.