



Red Hat Fuse 7.13

Tooling チュートリアル

CodeReady Studio で Fuse Tooling を使用方法の例

Red Hat Fuse 7.13 Tooling チュートリアル

CodeReady Studio で Fuse Tooling を使用方法の例

法律上の通知

Copyright © 2024 Red Hat, Inc.

The text of and illustrations in this document are licensed by Red Hat under a Creative Commons Attribution–Share Alike 3.0 Unported license ("CC-BY-SA"). An explanation of CC-BY-SA is available at

<http://creativecommons.org/licenses/by-sa/3.0/>

. In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

Red Hat, as the licensor of this document, waives the right to enforce, and agrees not to assert, Section 4d of CC-BY-SA to the fullest extent permitted by applicable law.

Red Hat, Red Hat Enterprise Linux, the Shadowman logo, the Red Hat logo, JBoss, OpenShift, Fedora, the Infinity logo, and RHCE are trademarks of Red Hat, Inc., registered in the United States and other countries.

Linux[®] is the registered trademark of Linus Torvalds in the United States and other countries.

Java[®] is a registered trademark of Oracle and/or its affiliates.

XFS[®] is a trademark of Silicon Graphics International Corp. or its subsidiaries in the United States and/or other countries.

MySQL[®] is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js[®] is an official trademark of Joyent. Red Hat is not formally related to or endorsed by the official Joyent Node.js open source or commercial project.

The OpenStack[®] Word Mark and OpenStack logo are either registered trademarks/service marks or trademarks/service marks of the OpenStack Foundation, in the United States and other countries and are used with the OpenStack Foundation's permission. We are not affiliated with, endorsed or sponsored by the OpenStack Foundation, or the OpenStack community.

All other trademarks are the property of their respective owners.

概要

このガイドには、Red Hat Fuse Tooling が提供するツールを使用してアプリケーションを開発およびテストする方法を示す簡単なチュートリアルが多数含まれています。

目次

多様性を受け入れるオープンソースの強化	4
第1章 FUSE TOOLING チュートリアルについて	5
前提条件	5
FUSE TOOLING チュートリアルの概要	5
サンプルアプリケーションについて	6
リソースファイルについて	6
第2章 環境の設定	7
ゴール	7
始める前に	7
FUSE INTEGRATION プロジェクトの作成	7
ID 値を表示するためのコンポーネントラベルの設定	13
プロジェクトのテストメッセージをダウンロードする	14
テストメッセージの表示	15
次のステップ	16
第3章 ルートの定義	18
ゴール	18
作業を開始する前に	18
ソースエンドポイントの設定	18
シンクエンドポイントの設定	19
次のステップ	21
第4章 ルートの実行	22
ゴール	22
前提条件	22
ルートを実行する	22
ルートを確認する	24
次のステップ	24
第5章 コンテンツベースのルーターの追加	25
ゴール	25
前提条件	25
コンテンツベースルーターの追加と設定	25
ロギングの追加と設定	29
メッセージヘッダーの追加と設定	31
有効な注文を処理するためのブランチの追加と設定	34
CBR の検証	40
次のステップ	41
第6章 ルーティングコンテキストに別のルートを追加する	43
ゴール	43
前提条件	43
既存のルートのエンドポイントを再設定する	43
2 番目のルートを追加する	44
米国の注文を処理するための CHOICE ブランチの設定	45
ドイツの注文を処理するための OTHERWISE ブランチの設定	52
2 番目のルートを確認する	56
次のステップ	60
第7章 ルーティングコンテキストのデバッグ	61
ゴール	61

前提条件	61
ブレイクポイントの設定	61
ルーティングコンテキストのステップスルー	63
変数の値を変更する	67
CAMEL デバッガーのフォーカスを絞る	71
メッセージ変数値を変更した場合の影響を確認する	73
次のステップ	74
第8章 ルートを介したメッセージのトレース	75
ゴール	75
前提条件	75
FUSE INTEGRATION パースペクティブの設定	75
メッセージトレースの開始	78
実行中の ZOOORDERAPP プロジェクトにメッセージをドロップする	81
メッセージビューの設定	82
メッセージトレースのステップスルー	84
次のステップ	86
第9章 JUNIT を使用したルートのテスト	87
概要	87
ゴール	87
前提条件	87
SRC/TEST フォルダーの作成	88
JUNIT テストケースの作成	90
BLUEPRINTXMLTEST ファイルの変更	94
POM.XML ファイルの変更	98
JUNIT テストの実行	98
関連資料	99
次のステップ	99
第10章 プロジェクトを RED HAT FUSE に公開する	100
ゴール	100
前提条件	100
RED HAT FUSE SERVER の定義	100
公開オプションの設定	104
ランタイムサーバーへの接続	109
ZOOORDERAPP プロジェクトのアンインストール	110

多様性を受け入れるオープンソースの強化

Red Hat では、コード、ドキュメント、Web プロパティにおける配慮に欠ける用語の置き換えに取り組んでいます。まずは、マスター (master)、スレーブ (slave)、ブラックリスト (blacklist)、ホワイトリスト (whitelist) の 4 つの用語の置き換えから始めます。この取り組みは膨大な作業を要するため、今後の複数のリリースで段階的に用語の置き換えを実施して参ります。詳細は、[CTO である Chris Wright のメッセージ](#) をご覧ください。

第1章 FUSE TOOLING チュートリアルについて

Red Hat Fuse Tooling チュートリアルは、Fuse Tooling を使用して Apache Camel アプリケーションを開発、実行、テスト、およびデプロイするための実践的な紹介を提供します。

前提条件

始める前に、次のソフトウェアに精通している必要があります。

- [Apache Camel](#)
- [Apache Maven](#)

FUSE TOOLING チュートリアルの概要

チュートリアルの概要と、各チュートリアルで達成できることは次のとおりです。

- **2章 環境の設定**
Fuse Integration プロジェクトを作成し、チュートリアルリソースファイル (メッセージの例とルーティングコンテキストファイル) を設定します。プロジェクトを作成すると、ルーティングコンテキストと事前ルートが自動作成されます。
- **3章 ルートの定義**
フォルダーからメッセージを取得して別のフォルダーにコピーする単純なルートのエンドポイントを定義します。
- **4章 ルートの実行**
テストメッセージを表示します。ルートを実行し、テストメッセージがソースフォルダーからターゲットフォルダーにコピーされたことを確認して、ルートが機能することを確認します。
- **5章 コンテンツベースのルーターの追加**
メッセージをフィルタリングし、メッセージのコンテンツに基づいて別のターゲットフォルダーにコピーするコンテンツベースのルーターを追加します。
- **6章 ルーティングコンテキストに別のルートを追加する**
メッセージをさらにフィルタリングし、メッセージのコンテンツに基づいて別のターゲットフォルダーにコピーする別のルートを追加します。
- **7章 ルーティングコンテキストのデバッグ**
Camel デバッガーを使用してブレークポイントを設定してから、ルートをステップスルーしてルート変数とメッセージ変数を調べます。
- **8章 ルートを介したメッセージのトレース**
メッセージをルートにドロップし、すべてのルートノードを介して追跡します。
- **9章 JUnit を使用したルートのテスト**
ルートの JUnit テストケースを作成してから、ルートをテストします。
- **10章 プロジェクトを Red Hat Fuse に公開する**
Apache Camel プロジェクトを Red Hat Fuse に公開するプロセスをウォークスルーします。ローカルサーバーを定義し、公開オプションを設定し、サーバーを起動し、プロジェクトを公開し、サーバーに接続して、プロジェクトが正常にビルドおよび公開されたことを確認します。

Fuse Tooling 機能の詳細は、[Tooling ユーザーガイド](#) を参照してください。

サンプルアプリケーションについて

Fuse Tooling チュートリアルでビルドしたサンプルアプリケーションは、動物園が動物を注文するための簡単な注文アプリケーションをシミュレートします。サンプルの XML メッセージが提供されます。各 XML メッセージには、お客様情報 (動物園の名前、都市、国) と注文情報 (要求された動物の種類と数、および許可される動物の最大数) が含まれます。

Fuse Tooling を使用して、受信サンプルメッセージを取得し、コンテンツ (有効な注文と無効な注文) に基づいてフィルタリングし、動物園の場所 (国) で有効な注文をさらに並べ替えるブループリントプロジェクトを作成します。後のチュートリアルでは、サンプルアプリケーションを使用して、ルーティングコンテキストをデバッグし、ルートを紹介してメッセージをトレースし、JUnit でルートをテストし、最後に Fuse プロジェクトを公開します。

リソースファイルについて

各チュートリアルは、前のチュートリアルの上にビルドされています。あるチュートリアルで生成されたコードは、次のチュートリアルの開始点であり、チュートリアルを順番に完了することができます。または、最初のチュートリアルを完了した後、提供されているコンテキストファイルの1つを開始点として使用して、他のチュートリアルを順番どおりに実行することもできます。

チュートリアルは、[ここ](#)にある **Fuse-tooling-tutorials-jbds-10.3.zip** ファイルにあるリソースファイルに依存します。この zip ファイルには2つのフォルダーが含まれています。

メッセージ

このフォルダーには、**message1.xml**、**message2.xml**、...、**message6.xml** という名前の6つのメッセージファイルが含まれます。最初のチュートリアルでは、[2章環境の設定](#)、これらのメッセージファイルを保存するディレクトリを作成し、そのコンテンツも表示します。これらのメッセージファイルは、すべてのチュートリアルに必要です。

blueprintContexts

このフォルダーには、次の3つのルーティングコンテキストファイルが含まれています。

- **Blueprint1.xml**: これは、[3章ルートの定義](#)のチュートリアルを完了することで得られるソリューションルーティングコンテキストです。次のチュートリアルの開始点として使用できます。
 - [4章ルートの実行](#)
 - [5章コンテンツベースのルーターの追加](#)
- **Blueprint2.xml**: これは、[5章コンテンツベースのルーターの追加](#)チュートリアルのソリューションコンテキストファイルです。**blueprint2.xml** を、[6章ルーティングコンテキストに別のルートを追加する](#)チュートリアルの開始点として使用できます。
- **Blueprint3.xml**: これは、[6章ルーティングコンテキストに別のルートを追加する](#)チュートリアルのソリューションコンテキストファイルです。**blueprint3.xml** を、以下のチュートリアルの開始点として使用できます。
 - [7章ルーティングコンテキストのデバッグ](#)
 - [8章ルートを紹介したメッセージのトレース](#)
 - [9章JUnit を使用したルートのテスト](#)
 - [10章プロジェクトをRed Hat Fuse に公開する](#)

第2章 環境の設定

このチュートリアルでは、Fuse Integration プロジェクトを作成するプロセスについて説明します。プロジェクトには、初期ルートとデフォルトの CamelContext が含まれています。ルートは、メッセージが通過するプロセッサのチェーンです。CamelContext は、ルートを設定するためのコンテキストを定義し、エンドポイント (メッセージソースとターゲット) 間のメッセージエクスチェンジに使用するポリシーを指定する単一のルーティングルールベースです。

他のチュートリアルに従う前に、このチュートリアルを完了する必要があります。

ゴール

このチュートリアルでは、次のタスクを完了します。

- Fuse Integration プロジェクトを作成する
- プロジェクトのテストメッセージ (XML ファイル) をダウンロードします
- テストメッセージを表示する

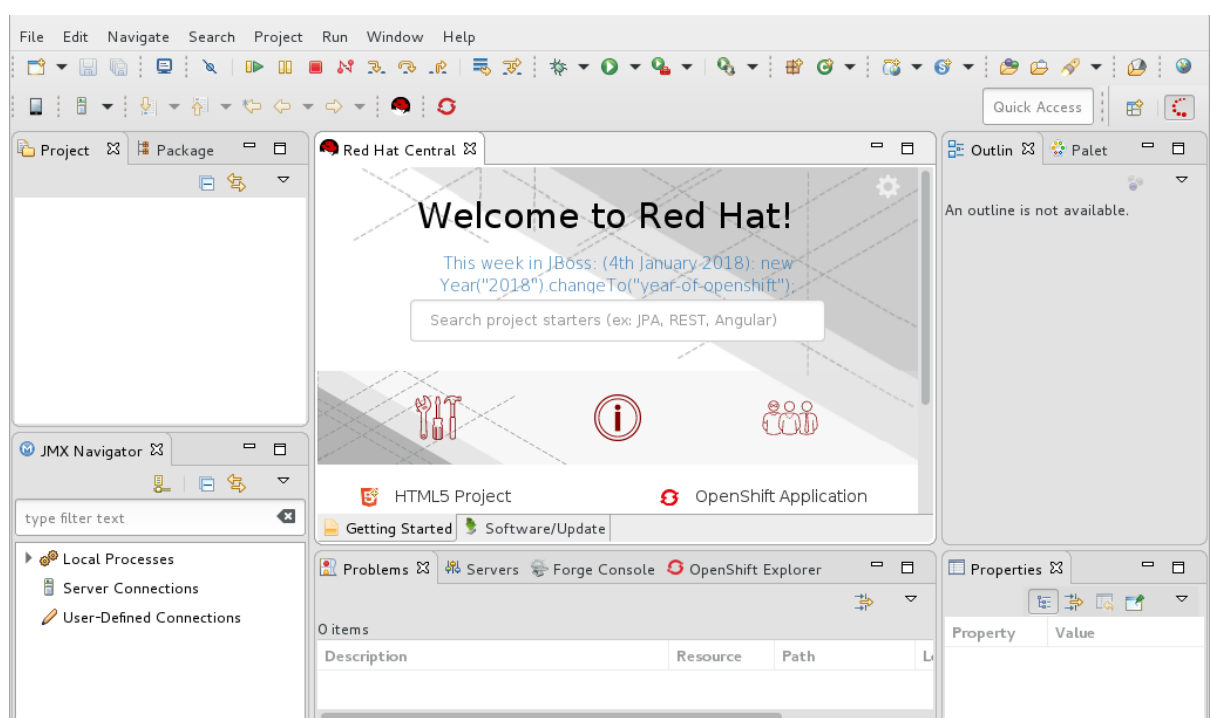
始める前に

Fuse Integration プロジェクトを設定する前に、Fuse Tooling を使用して Red Hat CodeReady Studio をインストールする必要があります。CodeReady Studio のインストール方法は、[Red Hat customer portal](#) にアクセスして、プラットフォームのインストールガイドを参照してください。

次の手順を実行する前に[10章 プロジェクトを Red Hat Fuse に公開する](#)チュートリアルでは、Java8 をインストールする必要があります。

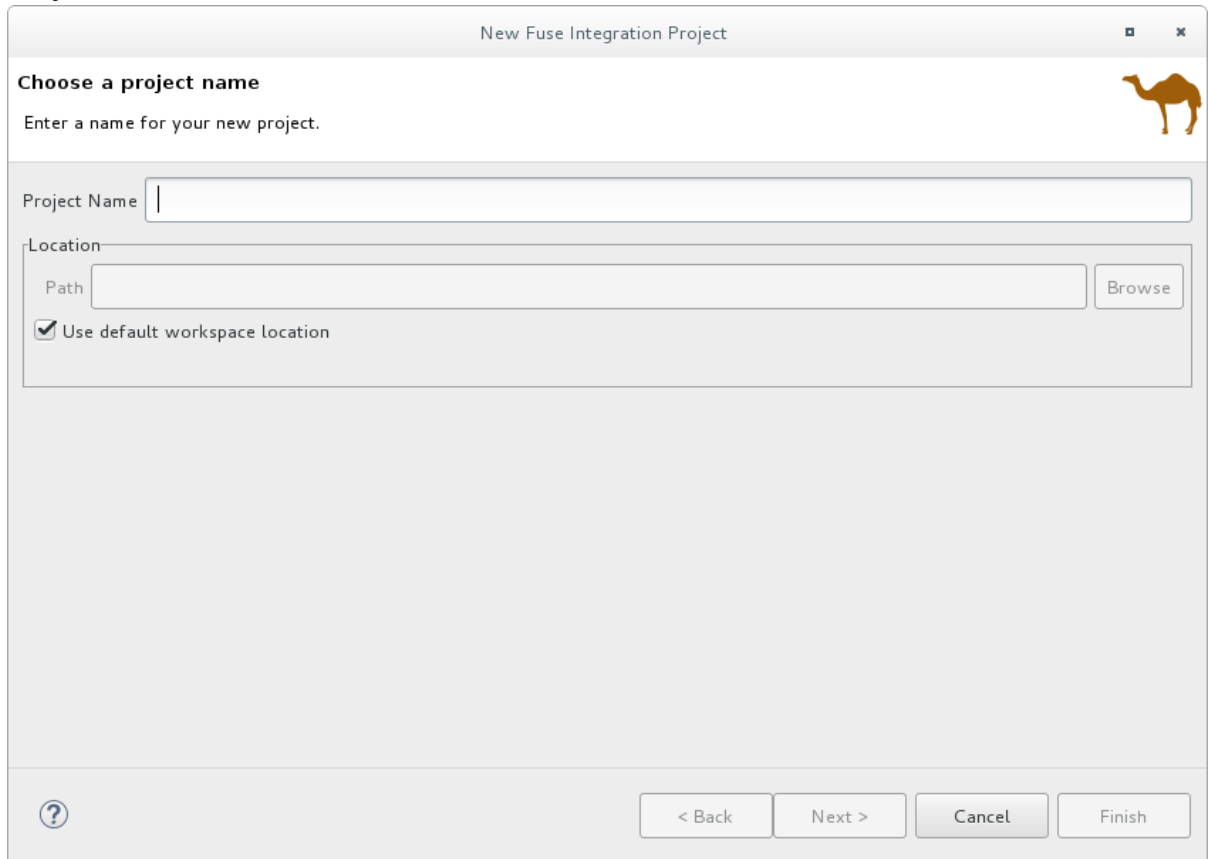
FUSE INTEGRATION プロジェクトの作成

1. Red Hat CodeReady Studio を開きます。
CodeReady Studio を初めて起動すると、JBoss パースペクティブで開きます。



それ以外の場合は、前の CodeReady Studio セッションで使用していたパースペクティブで開きます。

2. メニューから、**File → New → Fuse Integration Project** を選択して、**New Fuse Integration Project** ウィザードを開きます。



3. **Project Name** フィールドに **ZooOrderApp** を入力します。
Use default workspace location オプションをオンのままにします。
4. **Next** をクリックして、**Select a Target Runtime** ページを開きます。

New Fuse Integration Project

Select a Target Environment

Select a target environment for deploying your new project.

Choose the deployment platform

Kubernetes/OpenShift

Standalone

Choose the runtime environment

Spring Boot

Karaf/Fuse on Karaf

Runtime (optional) None selected

Wildfly/Fuse on EAP

Runtime (optional) None selected

Select the Camel version

5. デプロイメントプラットフォームとして**Standalone** を選択します。
6. **Karaf/Fuse on Karaf** を選択し、ランタイムに選択されている**None selected** を受け入れません。



注記

後でランタイムを追加します。[10章 プロジェクトを Red Hat Fuse に公開するチュートリアル](#)。

7. デフォルトの Apache **Camel version** を受け入れます。

New Fuse Integration Project

Select a Target Environment

Select a target environment for deploying your new project.

Choose the deployment platform

Kubernetes/OpenShift

Standalone

Choose the runtime environment

Spring Boot

Karaf/Fuse on Karaf

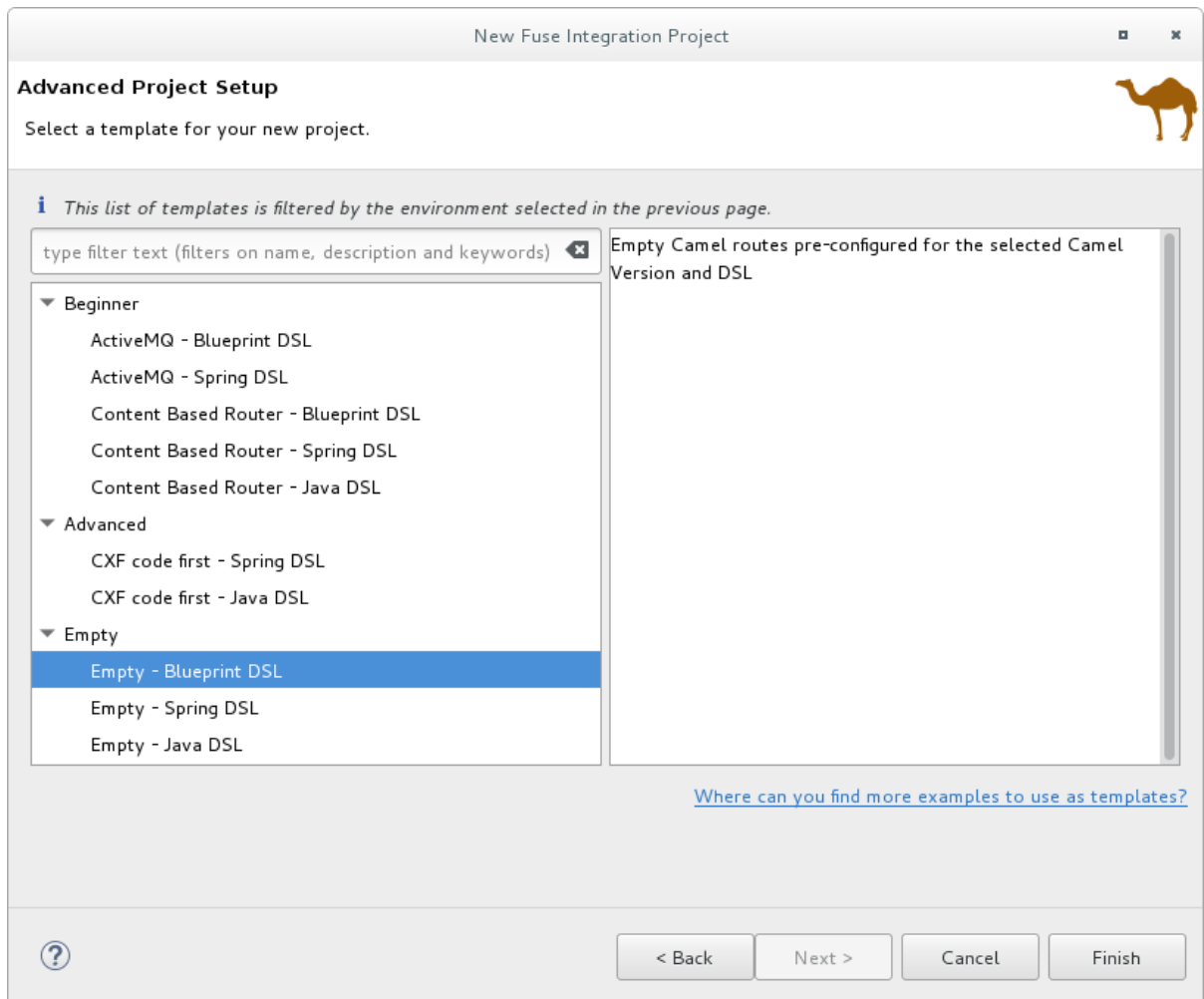
Runtime (optional) None selected

Wildfly/Fuse on EAP

Runtime (optional) None selected

Select the Camel version

8. **Next** をクリックして **Advanced Project Setup** ページを開き、**Empty - Blueprint DSL** テンプレートを選択します。



9. **Finish** をクリックします。

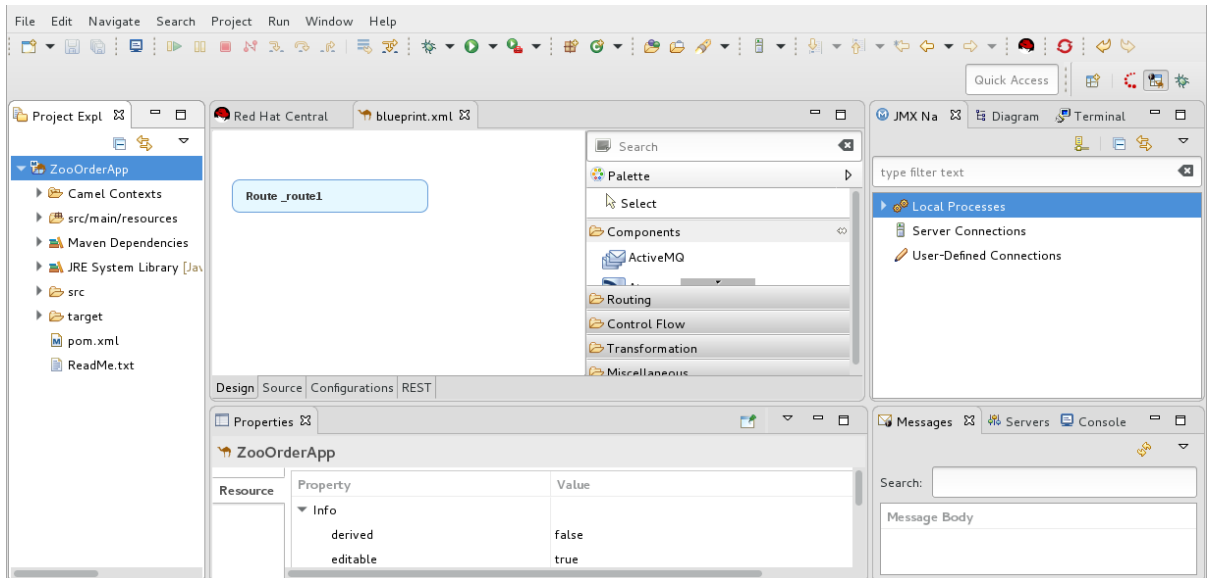
Fuse Tooling は、Maven リポジトリから、プロジェクトのビルドに必要なすべてのファイルのダウンロードを開始し、新しいプロジェクトを **Project Explorer** ビューに追加します。

CodeReady Studio がまだ **Fuse Integration** パースペクティブを表示していない場合は、今すぐ切り替えるかどうかを尋ねられます。



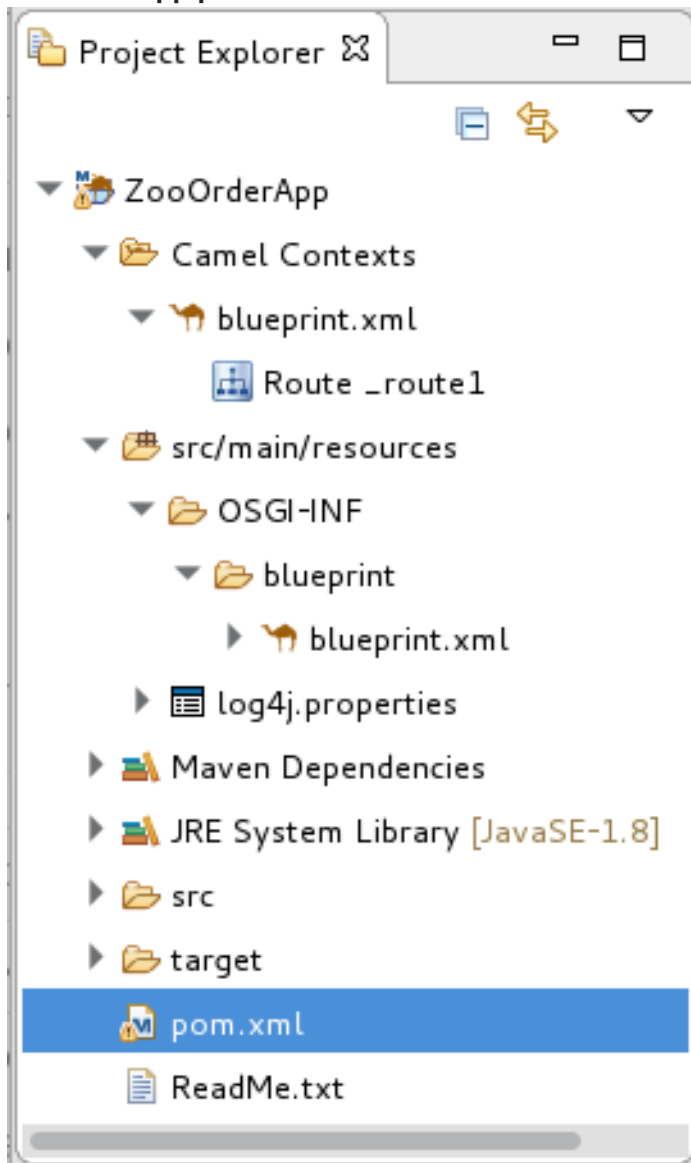
10. **Yes** をクリックします。

新しい **ZooOrderApp** プロジェクトが **Fuse Integration** パースペクティブで開きます。

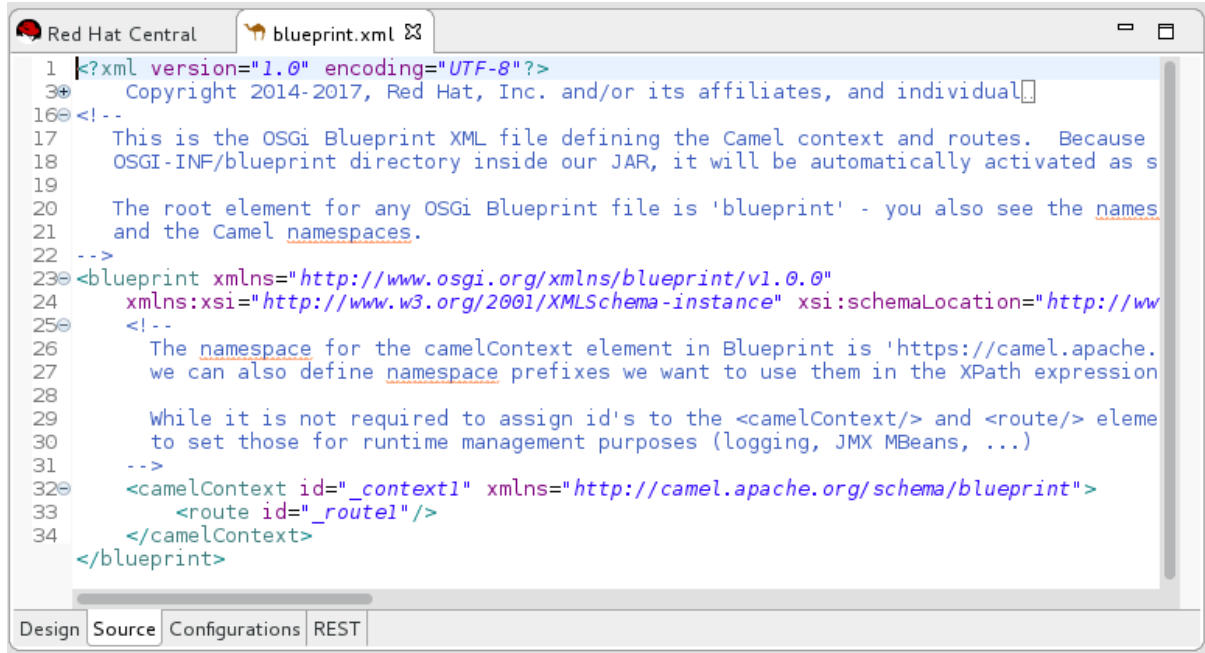


ZooOrderApp プロジェクトには、ルートを作成して実行するために必要なすべてのファイルが含まれています。

- **ZooOrderApp/pom.xml** – Maven プロジェクトファイル。



- **ZooOrderApp/src/main/resources/OSGI-INF/blueprint/blueprint.xml** – Camel ルーティングコンテキストと初期の空のルートが含まれる Blueprint XML ファイル。
11. メインのルーティングコンテキストを表示するには、Editor ビューで **blueprint.xml** ファイルを開き、**Source** タブをクリックします。

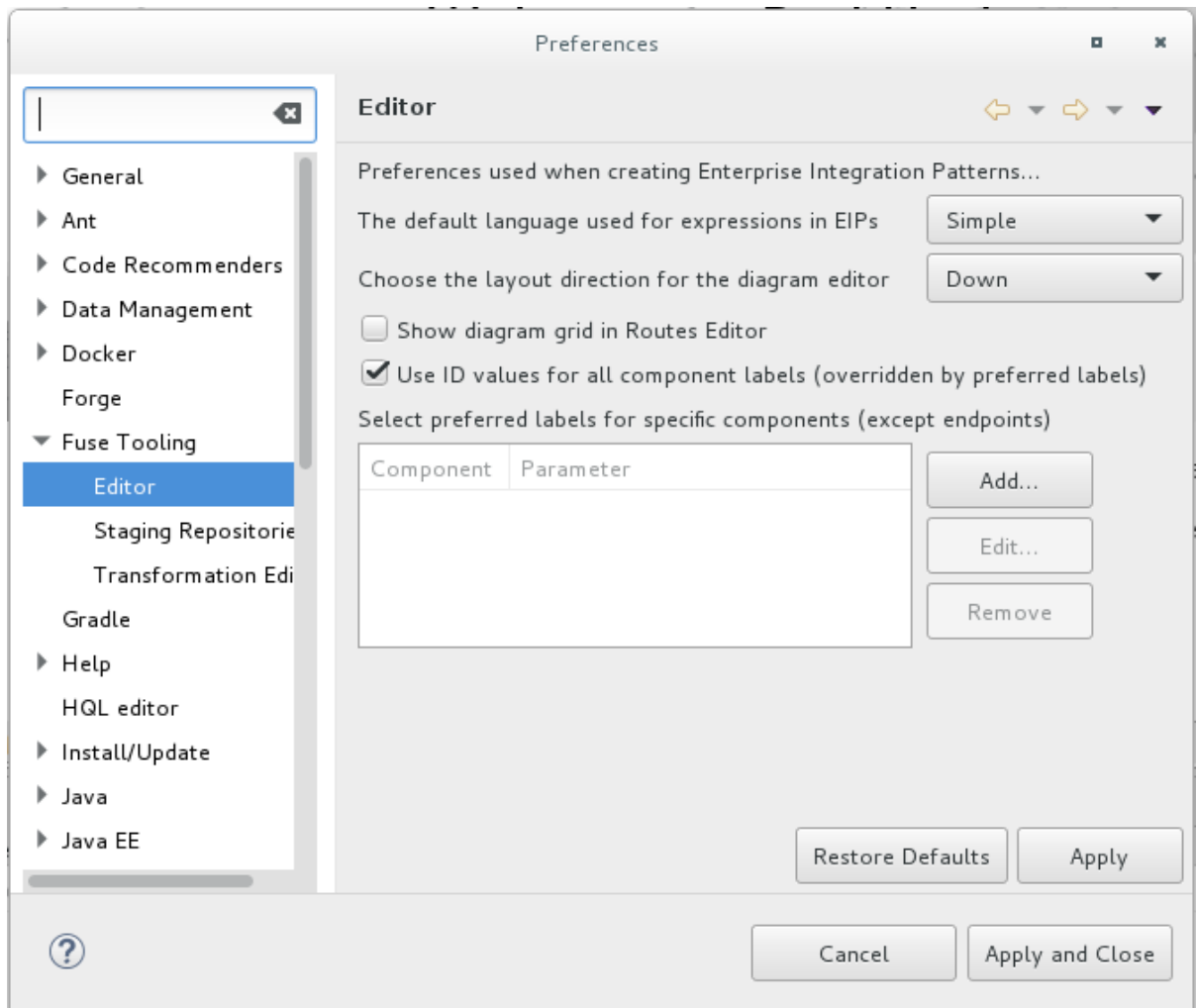


```
1 <?xml version="1.0" encoding="UTF-8"?>
2 Copyright 2014-2017, Red Hat, Inc. and/or its affiliates, and individual
16 <!--
17 This is the OSGi Blueprint XML file defining the Camel context and routes. Because
18 OSGI-INF/blueprint directory inside our JAR, it will be automatically activated as s
19
20 The root element for any OSGi Blueprint file is 'blueprint' - you also see the names
21 and the Camel namespaces.
22 -->
23 <blueprint xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0"
24 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://ww
25 <!--
26 The namespace for the camelContext element in Blueprint is 'https://camel.apache.
27 we can also define namespace prefixes we want to use them in the XPath expression
28
29 While it is not required to assign id's to the <camelContext/> and <route/> eleme
30 to set those for runtime management purposes (logging, JMX MBeans, ...)
31 -->
32 <camelContext id="_context1" xmlns="http://camel.apache.org/schema/blueprint">
33 <route id="_route1"/>
34 </camelContext>
</blueprint>
```

ID 値を表示するためのコンポーネントラベルの設定

デザインキャンバスに配置するパターンとコンポーネントのラベルが、Tooling チュートリアルに示されているラベルと同じであることを確認するには、次の手順に従います。

1. エディター設定ページを開きます。
 - Linux および Windows マシンでは、**Windows → Preferences → Fuse Tooling → Editor** を選択します。
 - OS X では、**CodeReady Studio → Preferences → Fuse Tooling → Editor** を選択します。
2. **Use ID values for all component labels** オプションをオンにします。



3. **Apply and Close** をクリックします。

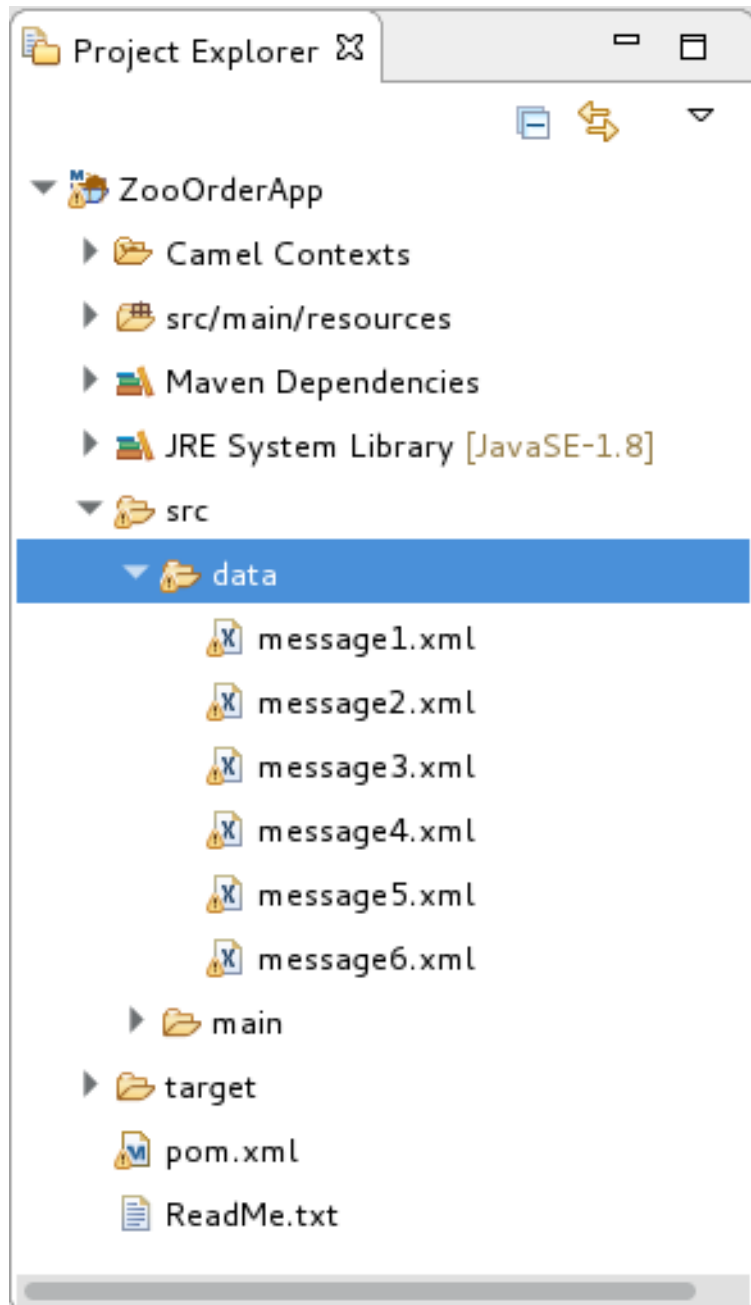
プロジェクトのテストメッセージをダウンロードする

サンプルの XML メッセージファイルが提供されているので、Tooling チュートリアルを実行しながら ZooOrderApp プロジェクトをテストできます。メッセージには、動物園の動物の注文情報が含まれています。たとえば、シカゴ動物園の 5 頭のウォンバットの注文。


提供されたテストメッセージ (XML ファイル) をダウンロードしてプロジェクトにコピーするには:

1. CodeReady Studio **Project Explorer** ビューで、テストメッセージを含むフォルダーを作成します。
 - a. **ZooOrderApp/src** フォルダーを右クリックし、**New → Folder** を選択します。New Folder ウィザードが開きます。
 - b. **Folder name** に **data** と入力します。
 - c. **Finish** をクリックします。
2. [こちら](#) をクリックして、提供される Tooling チュートリアルのリソース **Fuse-tooling-tutorials-jbds-10.3.zip** ファイルの場所に対して Web ブラウザーを開きます。**Fuse-tooling-tutorials-jbds-10.3.zip** ファイルを、ZooOrderApp プロジェクトのワークスペースの外部にある便利な場所にダウンロードし、デプロイメントします。[1章 Fuse Tooling チュートリアルについて](#)で説明されているように、2つのフォルダーが含まれています。

3. `messages` フォルダから、6つのXMLファイルを `ZooOrderApp` プロジェクトの `src/data` フォルダにコピーします。



注記

XML ファイルの  は無視しても問題はありません。

テストメッセージの表示

各XMLメッセージファイルには、動物園(お客様)からの大量の動物の注文が含まれています。たとえば、`message1.xml` ファイルには、ブルックリン動物園からの12頭のウォンバットの注文が含まれています。

Editor ビューで任意のメッセージXMLファイルを開いて、コンテンツを調べることができます。

1. **Project Explorer** ビューで、メッセージファイルを右クリックします。
2. ポップアップメニューから **Open** を選択します。

3. **Source** タブをクリックします。
XML ファイルが **Editor** ビューで開きます。


たとえば、**message1.xml** ファイルの内容には、Brooklyn 動物園からの 12 頭のウォンバットに関する注文が表示されます。

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

<order>
  <customer>
    <name>Bronx Zoo</name>
    <city>Bronx NY</city>
    <country>USA</country>
  </customer>
  <orderline>
    <animal>wombat</animal>
    <quantity>12</quantity>
  </orderline>
</order>
```



注記

新しく作成した **message1.xml** ファイルの最初の行の  を無視しても問題はありません。ドキュメントが参照する文法の制約 (DTD または XML スキーマ) がないことを示しています。

次の表に、6 つのメッセージファイルすべての内容の概要を示します。

表2.1 提供されたテストメッセージ

msg#	<name>	<city>	<country>	<animal>	<quantity>
1	ブロンクス動物園	ブロンクス NY	USA	ウォンバット	12
2	サンディエゴ動物園	カリフォルニア州サンディエゴ	USA	キリン	3
3	シーライフセンター	ミュンヘン	ドイツ	ペンギン	15
4	Berlin Zoo	ベルリン	ドイツ	エミュー	6
5	フィラデルフィア動物園	フィラデルフィア PA	USA	キリン	2
6	セントルイス動物園	セントルイス MO	USA	ペンギン	10

次のステップ

CodeReady Studio プロジェクトを設定したので、次の手順 [3章ルート](#)の定義XML メッセージを処理するルートを定義するチュートリアルに進むことができます。

第3章 ルートの定義

このチュートリアルでは、エンドポイントをルートに追加および設定する方法について説明します。エンドポイントは、ルートを通過するメッセージのソースとシンクを定義します。**ZooOrderApp** プロジェクトでは、開始(ソース)エンドポイントはXMLメッセージファイルを含むフォルダーです。シンク(終了)エンドポイントは、プロジェクトで指定する別のフォルダーです。

ゴール

このチュートリアルでは、次のタスクを完了します。

- ルートにソースエンドポイントとシンクエンドポイントを追加します
- エンドポイントを設定します
- エンドポイントを接続します


作業を開始する前に

このチュートリアルを開始する前に:

1. [2章 環境の設定](#) チュートリアルで説明されているように、ワークスペース環境を設定する必要があります。
2. CodeReady Studio で、**Editor** ビューで **ZooOrderApp** プロジェクトの `/src/main/resources/OSGI-INF/blueprint/blueprint.xml` ファイルを開きます。
3. 必要に応じて、**Editor** ビューの下部にある **Design** タブをクリックして、**Route_route1** というラベルが付けられた初期ルートのグラフを表示します。

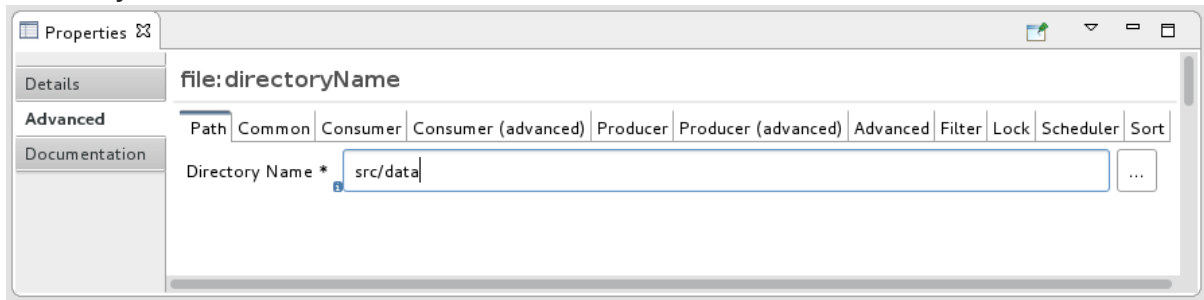
ソースエンドポイントの設定

以下の手順に従って、**src/data** フォルダーをルートのソースエンドポイントとして設定します。

1. **File** コンポーネント () を **Palette** の **Components** ドロワーからキャンバスにドラッグし、それを **Route_route1** コンテナノードにドロップします。
File コンポーネントは、**Route_route1** コンテナノード内の **From_from1** ノードに変更されます。
2. キャンバスで **From_from1** ノードを選択します。
キャンバスの下にある **Properties** ビューには、編集用のノードのプロパティフィールドが表示されます。
3. メッセージファイルのソースディレクトリを指定するには、**Properties** ビューで **Advanced** タブをクリックします。

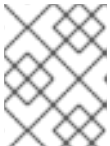
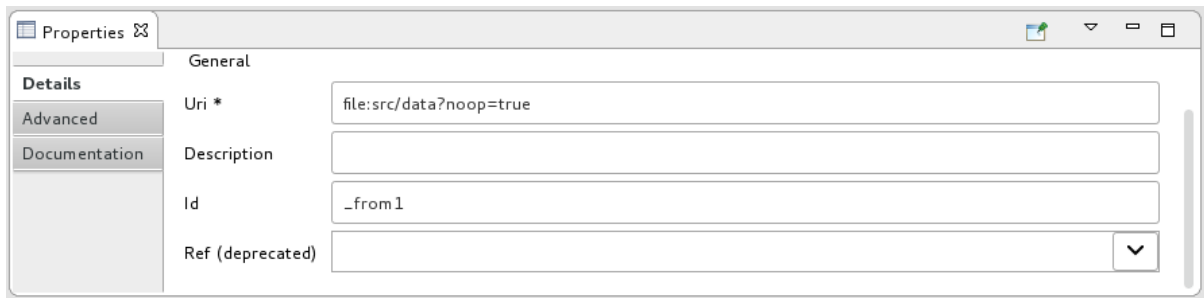


4. **Directory Name** フィールドに **src/data** を入力します。



src/data は、プロジェクトのディレクトリーからの相対パスです。

5. **Consumer** タブで、チェックボックスをクリックして **Noop** オプションを有効にします。
Noop オプションにより、**message#.xml** ファイルが **src/data** フォルダーから削除されなくなり、べき等を有効にして各 **message#.xml** ファイルが1度だけ消費されるようにします。
6. **Details** タブを選択して、ファイルノードの **Details** ページを開きます。
 ツールは、**Advanced** 設定タブで設定した **Directory Name** と **Noop** プロパティを **Uri** フィールドに自動的に入力することに注意してください。また、**Id** フィールドに自動生成される ID (**_from1**) も設定します。



注記

ツールは、自動生成された ID 値の前にアンダースコア () を付けます。オプションで ID 値を変更できます。アンダースコア接頭辞は必須ではありません。

自動生成された **ID** はそのままにします。

7. **File** → **Save** を選択して、ルートを保存します。

シンクエンドポイントの設定

ルートのシンク (ターゲット) エンドポイントを追加して設定するには、次の手順に従います。

- 別の **File** コンポーネントを **Palette** の **Components** ドロワーからドラッグし、それを **Route_route1** コンテナノードにドロップします。
File コンポーネントは、**Route_route1** コンテナノード内の **To_to1** ノードに変更されます。
- キャンバスで **To_to1** ノードを選択します。
 キャンバスの下にある **Properties** ビューには、編集用のノードのプロパティフィールドが表示されます。
- Details** タブ:
 - Uri** フィールドに **file:target/messages/received** と入力します。


- b. Id フィールドに **_Received** と入力します。

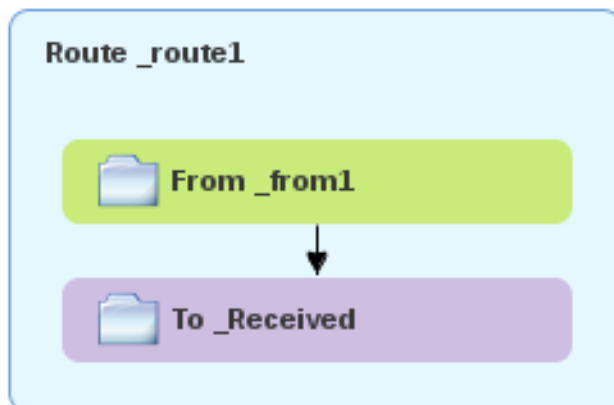
The screenshot shows the 'Properties' window for a component named 'To_to1'. The 'Id' field is filled with the text '_Received'. Other fields include 'Uri *' (file:target/messages/received), 'Description', and 'Pattern'.



注記

このツールは、実行時に **target/messages/received** フォルダを作成します。

4. **Route_route1** コンテナで **From_from1** ノードを選択し、そのコネクタ矢印 () を **To_Received** ノードにドラッグしてからリリースします。



注記

2つのファイルノードは、ルートエディターのレイアウト方向設定に従って、キャンバス上で接続および整列されます。選択肢は、**Down** (デフォルト) と **Right** です。

ルートエディターのレイアウト設定オプションにアクセスするには:

- Linux および Windows マシンでは、**Windows → Preferences → Fuse Tooling → Editor → Choose the layout direction for the diagram editor** を選択します。
- OSX では、**CodeReady Studio → Preferences → Fuse Tooling → Editor → Choose the layout direction for the diagram editor** を選択します。



注記

プロジェクトを閉じる前にノードを接続しない場合、プロジェクトを再度開くと、ツールによってノードが自動的に接続されます。

5. ルートを **Save** します。
6. キャンバスの下部にある **Source** タブをクリックして、ルートの XML を表示します。

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

<blueprint xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.osgi.org/xmlns/blueprint/v1.0.0
    https://www.osgi.org/xmlns/blueprint/v1.0.0/blueprint.xsd
    http://camel.apache.org/schema/blueprint
    http://camel.apache.org/schema/blueprint/camel-blueprint.xsd">

  <camelContext id="_context1" xmlns="http://camel.apache.org/schema/blueprint">
    <route id="_route1">
      <from id="_from1" uri="file:src/data?noop=true"/>
      <to id="_Received" uri="file:target/messages/received"/>
    </route>
  </camelContext>
</blueprint>
```

次のステップ

ルートにエンドポイントを追加して設定したので、次の [4章ルートの実行](#) チュートリアル説明に従ってルートを実行できます。

第4章 ルートの実行

このチュートリアルでは、ルートを実行して、ルートがソースエンドポイントからシンクエンドポイントにメッセージを正しく転送することを確認するプロセスについて説明します。

ゴール

このチュートリアルでは、次のタスクを完了します。

- ルートをローカル Camel コンテキストとして実行します (まだテストを設定していないため、テストなしで)
- ルートを介してメッセージを送信します。
- シンクエンドポイントが受信したメッセージを調べて、ルートがテストメッセージを正しく処理したことを確認します

前提条件

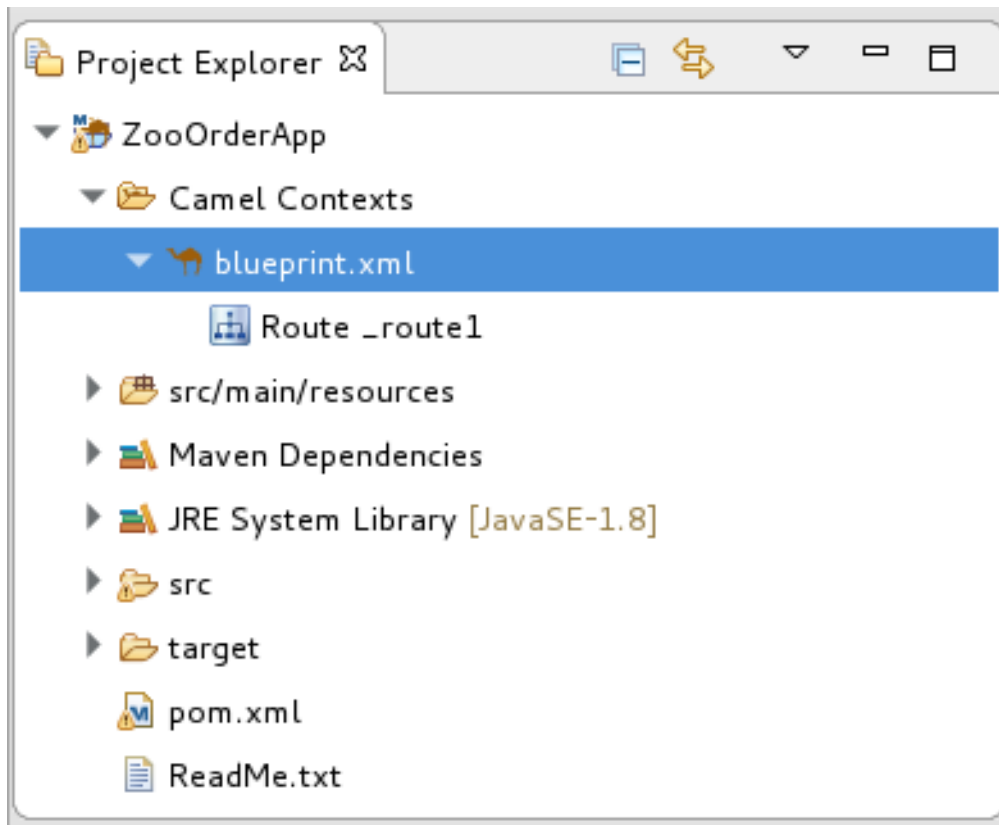
このチュートリアルを開始するには、次の結果の **ZooOrderApp** プロジェクトが必要です。

1. [2章環境の設定](#) チュートリアルを終了します。
2. 次のいずれかになります。
 - [3章ルートの定義](#) チュートリアルを終了します。
または
 - 「[リソースファイルについて](#)」で説明されているように、プロジェクトの **blueprint.xml** ファイルを提供される **blueprintContexts/blueprint1.xml** ファイルに置き換える。

ルートを実行する

ルートを実行するには:

1. **ZooOrderApp** プロジェクトを開きます。
2. Project Explorer で **ZooOrderApp/Camel Contexts/blueprint.xml** を選択します。



3. **blueprint.xml** を右クリックし、**Run As** → **Local Camel Context (without tests)** を選択します。



注記

代わりに **Local Camel Context** を選択すると、ツールは提供された JUnit テストに対してルーティングコンテキストを自動的に実行しようとしています。JUnit テストが存在しないため、ツールはテストなしでルーティングコンテキストの実行に戻ります。[9章 JUnit を使用したルートのテストチュートリアル](#)では、**ZooOrderApp** プロジェクトをテストするための JUnit テストケースを作成します。

Console パネルが開き、プロジェクトの実行の進行状況を反映するログメッセージが表示されます。最初に、Maven はローカルの Maven リポジトリを更新するために必要なリソースをダウンロードします。Maven のダウンロードプロセスには数分かかる場合があります。

4. 出力の最後にメッセージ (以下と同様) が表示されるのを待ちます。これらのメッセージは、ルートが正常に実行されたことを示しています。

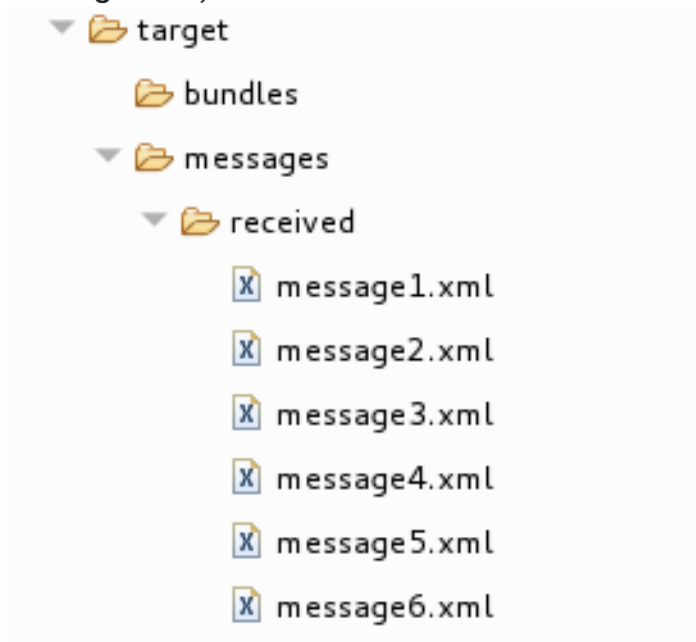
```
...
[Blueprint Event Dispatcher: 1] BlueprintCamelContext INFO Route: _route1 started and
consuming from:Endpoint[file://src/data?noop=true]
[Blueprint Event Dispatcher: 1] BlueprintCamelContext INFO Total 1 routes, of which 1 are
started.
[Blueprint Event Dispatcher: 1]BlueprintCamelContext INFO Apache Camel 2.21.0.redhat-3
(CamelContext: ...) started in 0.163 seconds
[Blueprint Event Dispatcher: 1] BlueprintCamelContext INFO Apache Camel 2.21.0.redhat-3
(CamelContext: ...) started in 0.918 seconds
```

5. ルートをシャットダウンするには、**Console** ビューの上部にある  をクリックします。

ルートを確認する

ルートが適切に実行されたことを確認するには、メッセージ XML ファイルがソースフォルダー (**src/data**) からターゲットフォルダー (**target/messages/received**) にコピーされたかどうかを確認します。

1. Project Explorer で **ZooOrderApp** を選択します。
2. 右クリックして、**Refresh** を選択します。
3. Project Explorer で **target/messages/** フォルダーを見つけ、そのフォルダーをデプロイメントし、**target/messages/received** フォルダーに6つのメッセージファイル (**message1.xml** から **message6.xml**) が含まれていることを確認します。



4. **message1.xml** をダブルクリックして、ルートエディターの **Design** タブでそれを開き、**Source** タブを選択して XML コードを表示します。

```
<?xml version="1.0" encoding="UTF-8"?>
<order>
  <customer>
    <name>Bronx Zoo</name>
    <city>Bronx NY</city>
    <country>USA</country>
  </customer>
  <orderline>
    <animal>wombat</animal>
    <quantity>12</quantity>
  </orderline>
</order>
```

次のステップ

5章 [コンテンツベースのルーターの追加](#) チュートリアルでは、メッセージのコンテンツを使用して宛先を決定するコンテンツベースのルーターを追加します。

第5章 コンテンツベースのルーターの追加

このチュートリアルでは、Content-Based Router (CBR) を追加してルートにログを記録する方法を示します。

CBR は、そのコンテンツに基づいてメッセージを宛先にルーティングします。このチュートリアルでは、作成する CBR は、各メッセージの数量フィールドの値 (注文された動物の数) に基づいて、メッセージをさまざまなフォルダー (有効または無効) にルーティングします。各注文の動物の最大値は 10 です。CBR は、数量が 10 より大きいかどうかに応じて、メッセージをさまざまなフォルダーにルーティングします。例えば、ある動物園が 5 頭のシマウマを注文し、3 頭のシマウマしか入手できなかった場合、注文は無効な注文先フォルダーにコピーされます。

ゴール

このチュートリアルでは、次のタスクを完了します。

- ルートにコンテンツベースのルーターを追加します
- コンテンツベースのルーターを設定します。
 - コンテンツベースルーターの各出力ブランチにログエンドポイントを追加します
 - 各ログエンドポイントの後に Set Header EIP を追加します
 - それ以外の場合は、コンテンツベースのルーターにブランチを追加します


前提条件

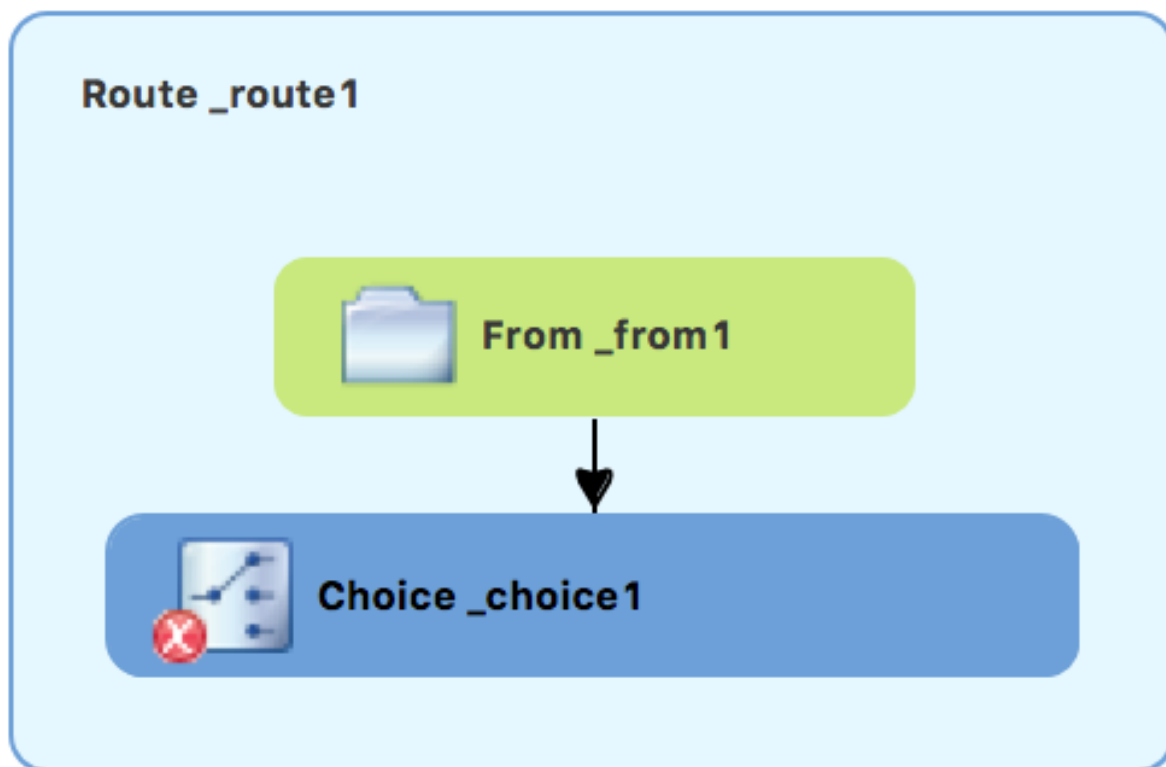
このチュートリアルを開始するには、次のいずれかの結果である ZooOrderApp プロジェクトが必要です。

- [4章 ルートの実行](#) チュートリアルを終了します。
または
- [2章 環境の設定](#) チュートリアルを完了し、「[リソースファイルについて](#)」に記載されているように、プロジェクトの `blueprint.xml` ファイルを、提供される `blueprintContexts/blueprint1.xml` ファイルに置き換える。


コンテンツベースルーターの追加と設定

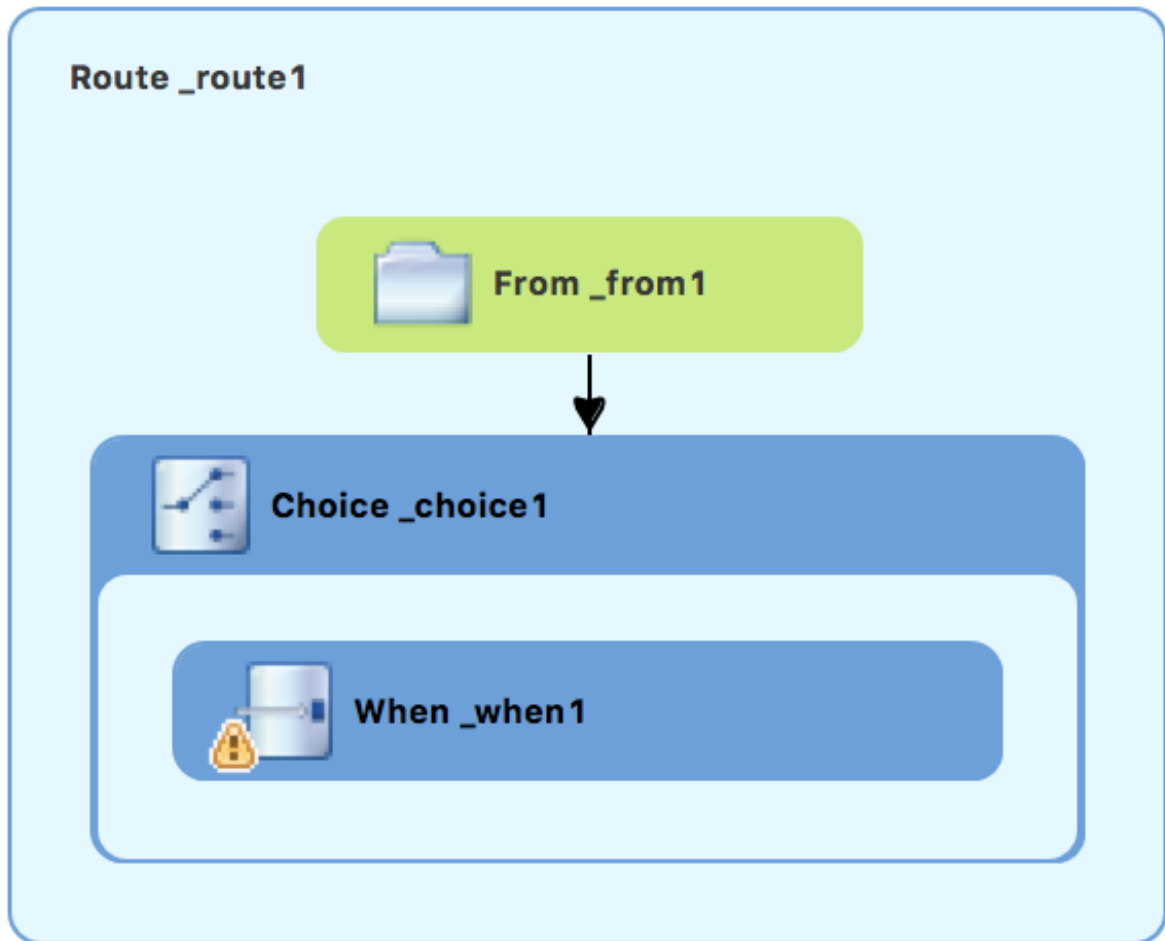
ルートにコンテンツベースのルーターを追加して設定するには、次の手順に従います。


1. Project Explorer で `ZooOrderApp/src/main/resources/OSGI-INF/blueprint/blueprint.xml` をダブルクリックして Editor ビューで開きます。
2. Design キャンバスで `To_Received` ノードを選択し、ごみ箱アイコンを選択して削除します。
3. Palette で Routing ドロワーを開き、Choice () パターンをクリックし、(Design キャンバスで) `From_from1` ノードをクリックします。

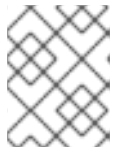


Route_route1 コンテナが拡張し、**Choice_choice1** ノードが追加されます。エラーアイコンは、**Choice_choice1** ノードに子ノードが必要なことを示しています。子ノードは、次のステップで追加します。

4. Routing ドロワーから、When () パターンをクリックし、キャンバスで **Choice_choice1** ノードをクリックします。
Choice_choice1 コンテナが拡張し、**When_when1** ノードが追加されます。



When_when1 ノードに付随する  は、1つまたは複数の必要なプロパティ値を設定する必要があることを示しています。




注記

このツールは、ルートコンテナの無効なドロップポイントにパターンを追加することを防ぎます。

- キャンバスで **When_when1** ノードを選択し、**Properties** ビューでそのプロパティを開きます。



- Expression** フィールドの  ボタンをクリックして、使用可能なオプションのリストを開きます。

7. テストメッセージは XML で記述されているため、**xpath** (XML クエリ言語の場合) を選択します。



注記

Expression を選択すると、**Properties** ビューに、**Expression** フィールドのすぐ下のインデントされたリストにそのプロパティが表示されます。このインデントされたリストの **Id** プロパティは、式の ID を設定します。**Description** フィールドに続く **Id** プロパティによって **When** ノードの ID が設定されます。

8. インデントされた **Expression** フィールドに、**/order/orderline/quantity/text() > 10** と入力します。
この式は、**quantity** フィールドの値が 10 より大きいメッセージのみが、ルート内のこのパスを (**invalidOrders** フォルダに) 移動することを指定します。
9. 残りの各プロパティはそのままにしておきます。



注記

Trim オプション (デフォルトで有効) は、メッセージから先頭または末尾の空白と改行を削除します。

The screenshot shows the 'Properties' window for a 'When' node named '_when1'. The 'Expression' is set to 'xpath' and the expression text is '/order/orderline/quantity/text() > 10'. The 'Result Type' is 'NODESET' and the 'Trim' checkbox is checked. The 'Id' field is set to '_when1'.

When _when1	
General	
Expression *	xpath
Expression *	/order/orderline/quantity/text() > 10
Document Type	
Factory Ref	
Header Name	
Id	
Log Namespaces	<input type="checkbox"/>
Object Model	
Result Type	NODESET
Saxon	<input type="checkbox"/>
Thread Safety	<input type="checkbox"/>
Trim	<input checked="" type="checkbox"/>
Description	
Id	_when1

10. ルーティングコンテキストファイルを **Save** します。
11. **Source** タブをクリックして、ルートの XML を表示します。


```


19  this is the OSGi Blueprint XML file defining the Camel context and routes.
25  <blueprint xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0"
26    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="h
27  <!--
28    The namespace for the camelContext element in Blueprint is 'https://camel
29    we can also define namespace prefixes we want to use them in the XPath ex
30
31    While it is not required to assign id's to the <camelContext/> and <route
32    to set those for runtime management purposes (logging, JMX MBeans, ...)
33  -->
34  <camelContext id="_context1" xmlns="http://camel.apache.org/schema/blueprin
35  <route id="route1" shutdownRoute="Default">
36    <from id="_from1" uri="file:src/data?noop=true" />
37    <choice id="_choice1">
38    <when id="when1">
39      <xpath>/order/orderline/quantity/text() &gt; 10</xpath>
40    </when>
41    </choice>
42  </route>
43  </camelContext>
</blueprint>

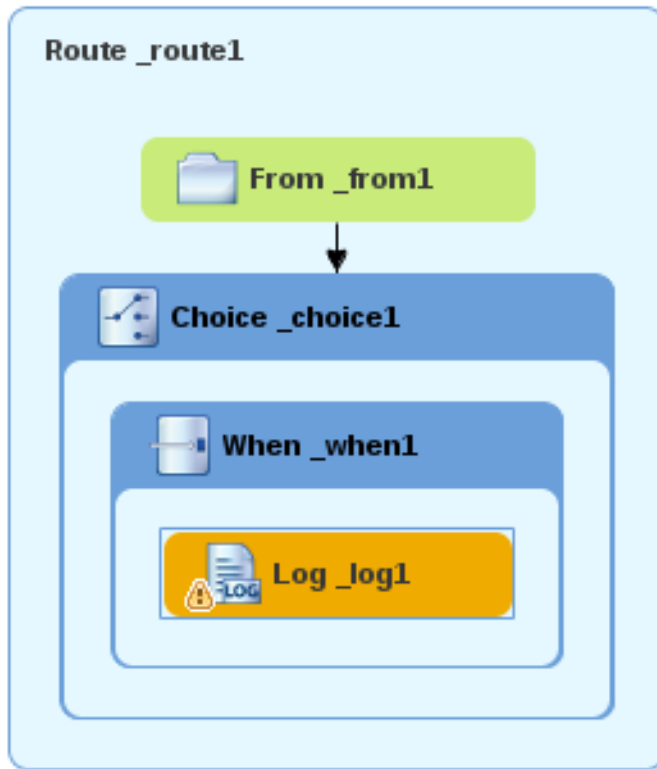
```

ロギングの追加と設定

ZooOrder アプリケーションの例では、ログメッセージを追加して、ルートを通過するときに XML メッセージを追跡できるようにします。ルートを実行すると、ログメッセージが **Console** ビューに表示されます。

次の手順に従って、CBR ルートにロギングを追加します。

1. **Design** タブの **Palette** で、**Components** ドロワーを開き、**Log** コンポーネント() をクリックします。
2. キャンバスで **When_when1** ノードをクリックします。
When_when1 コンテナが拡張し、**Log_log1** ノードが追加されます。



3. キャンバスで **Log_log1** ノードを選択し、**Properties** ビューでそのプロパティを開きます。
4. **Message** フィールドで、**The quantity requested exceeds the maximum allowed - contact customer.**と入力します。

Log_log1	
Message *	The quantity requested exceeds the maximum allowed - contact customer.
Description	
Id	_log1
Log Name	
Logger Ref	
Logging Level	INFO
Marker	

残りのプロパティはそのまましておきます。

+



注記


ツールは、ログノード id 値を自動生成します。Fuse Integration パースペクティブの Messages ビューで、ルートでトレースが有効になっている場合、ツールはメッセージインスタンスの Trace NodeId 列にログノードの Id フィールドのコンテンツを挿入します (8章 [ルートを介したメッセージのトレース](#) チュートリアルを参照してください)。Console では、ルートが実行されるたびに、ログノードの Message フィールドのコンテンツがログデータに追加されます。

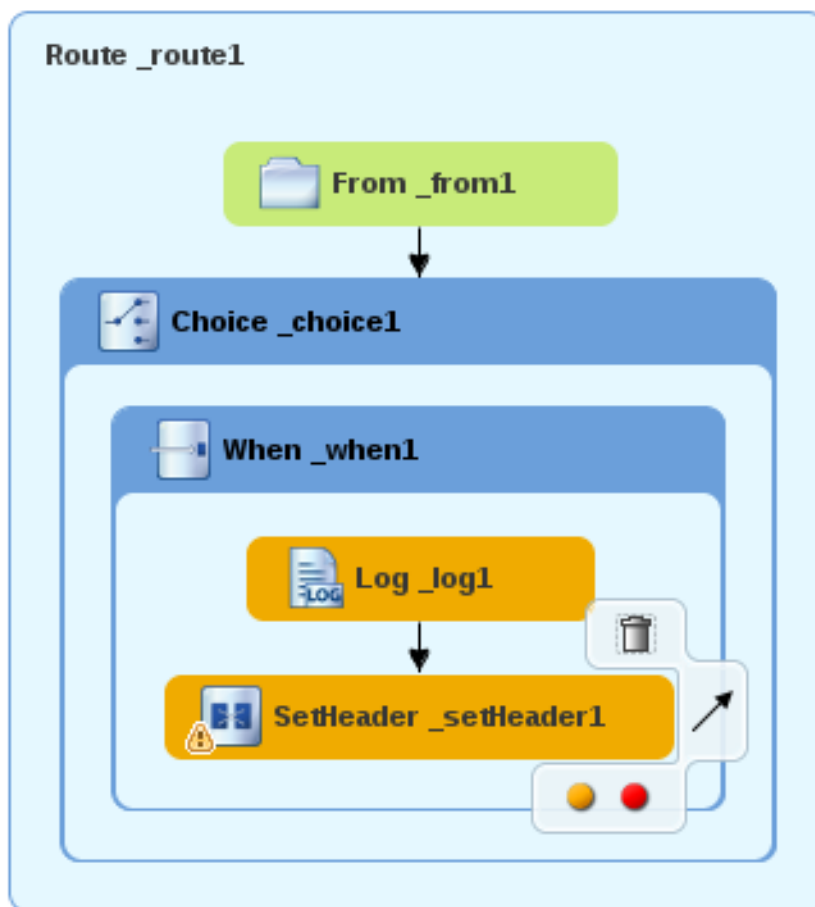
1. ルーティングコンテキストファイルを Save します。

メッセージヘッダーの追加と設定

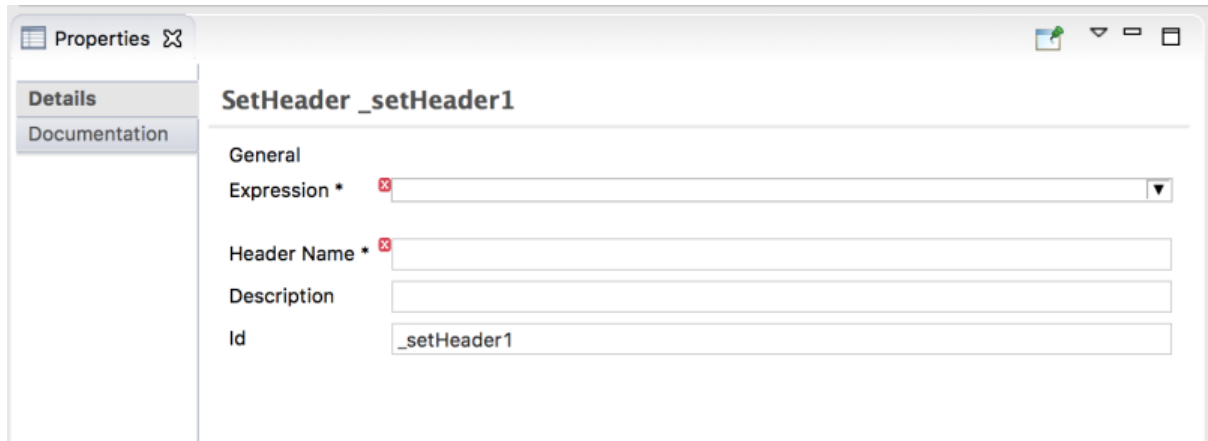
メッセージヘッダーには、メッセージを処理するための情報が含まれています。


メッセージヘッダーを追加および設定するには:

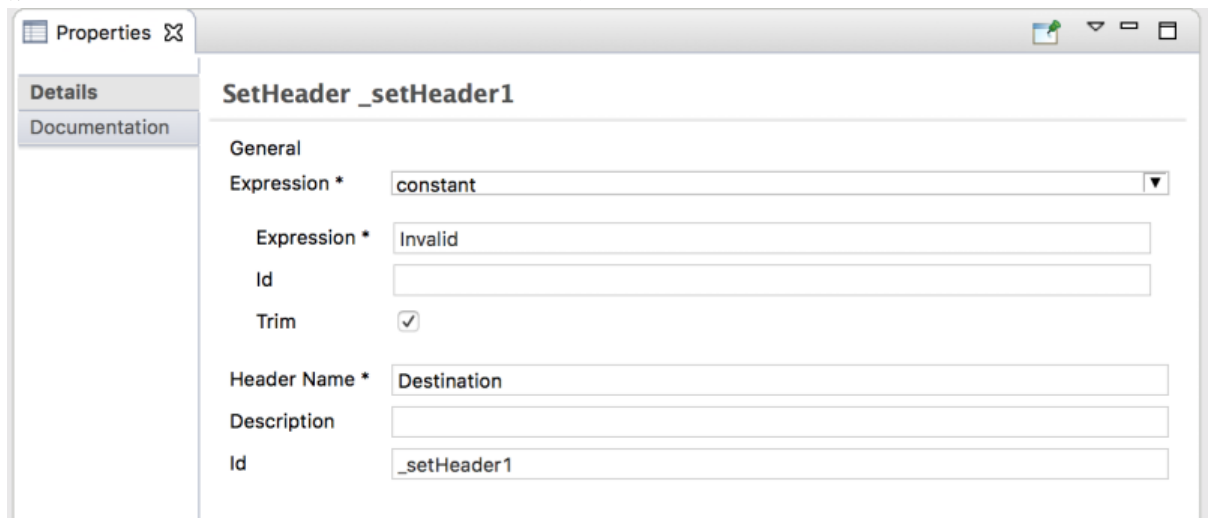
1. Palette で、Transformation ドロワーを開き、Set Header () パターンをクリックします。
2. キャンバスで Log_log1 ノードをクリックします。
When_when1 コンテナが拡張し、SetHeader_setHeader1 ノードが追加されます。




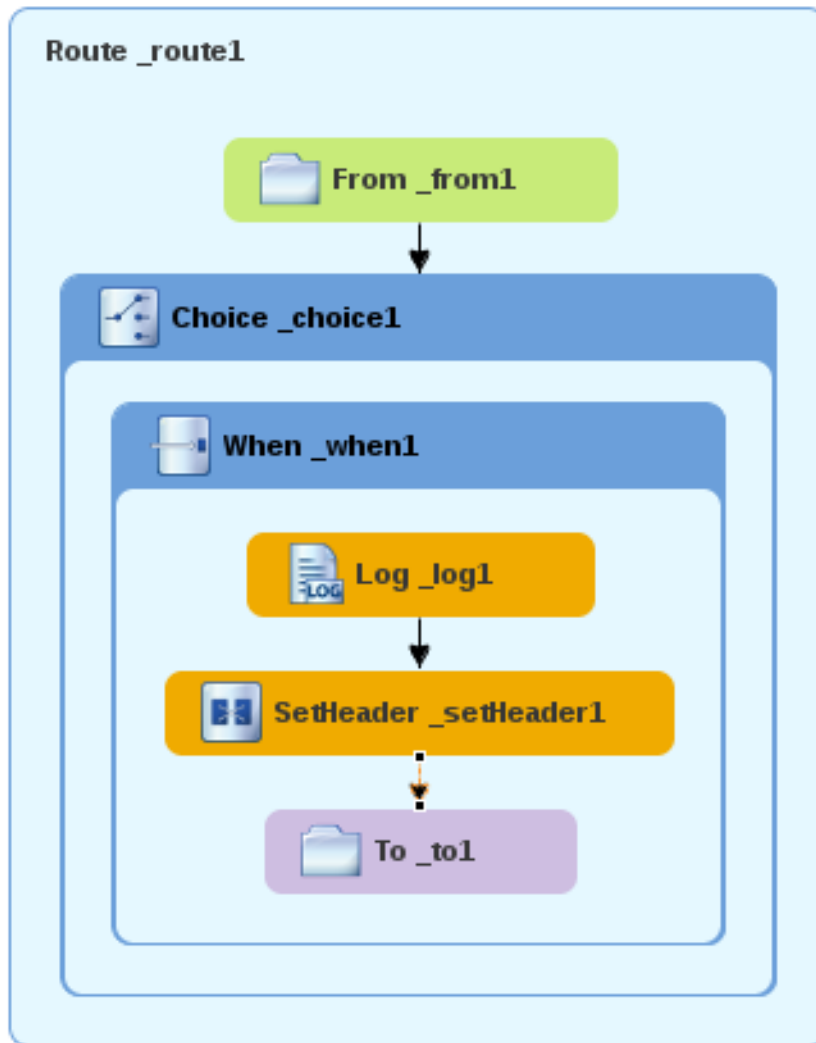
3. キャンバスで SetHeader_setHeader1 ノードを選択し、Properties ビューでそのプロパティを開きます。



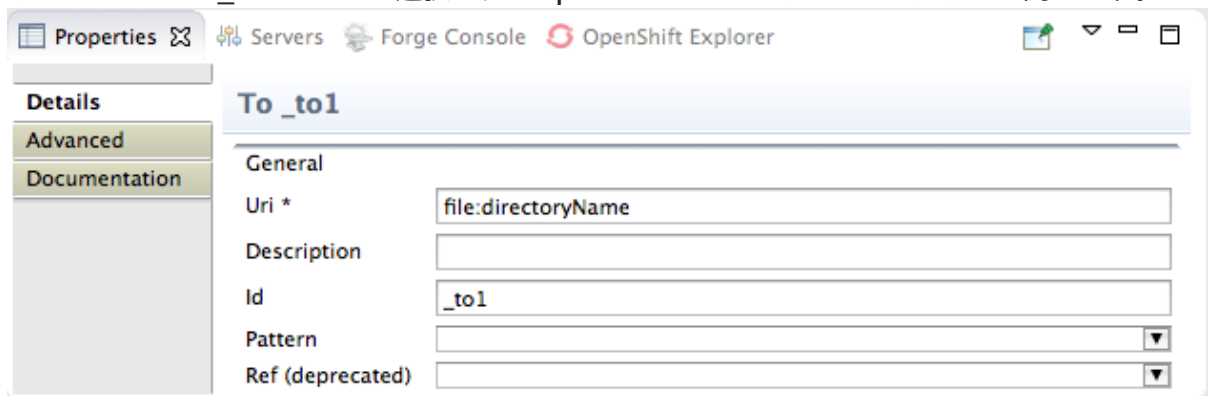
4.  **Expression** フィールドのボタンをクリックして使用可能な言語のリストを開き、**constant** を選択します。
5. インデントされた **Expression** フィールドに **Invalid** と入力します。
6. **Header Name** フィールドに **Destination** と入力します。
7. 残りのプロパティはそのままにしておきます。



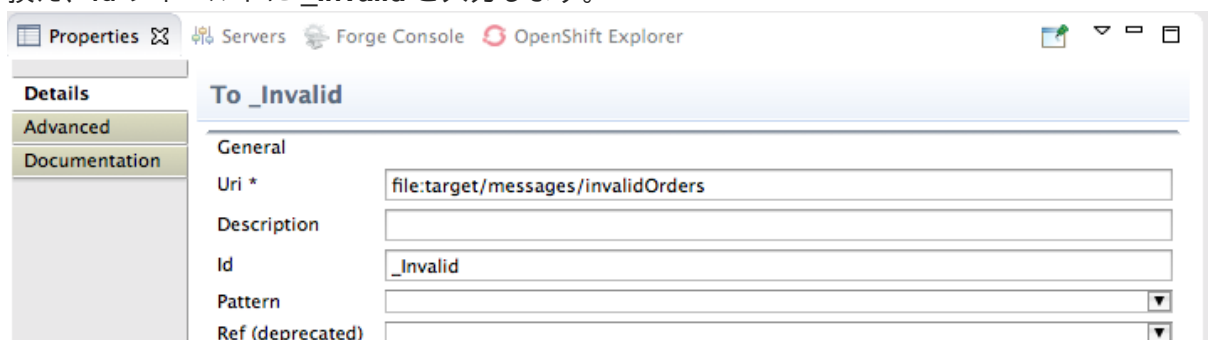
8. **Palette** で、**Components** ドロワーを開き、**File** () コンポーネントをクリックします。
9. キャンバスで **SetHeader_setHeader1** ノードをクリックします。
When_when1 コンテナが拡張し、**To_to1** ノードが追加されます。



10. キャンバスで **To_to1** ノードを選択し、Properties ビューでそのプロパティを開きます。



11. Details タブで、directoryName を Uri フィールドの **target/messages/invalidOrders** に置き換え、Id フィールドに **_Invalid** と入力します。



12. ルーティングコンテキストファイルを **Save** します。
13. **Source** タブをクリックして、ルート XML を表示します。

```


34 <camelContext id="_context1" xmlns="http://camel.apache.org/schema/blueprint">
35 <route id="route1" shutdownRoute="Default">
36 <from id="from1" uri="file:src/data?noop=true"/>
37 <choice id="_choice1">
38 <when id="_when1">
39 <xpath>/order/orderline/quantity/text() > 10</xpath>
40 <log id="_log1" message="quantity requested exceeds maximum allowed - contact customer"/>
41 <setHeader headerName="Destination" id="_setHeader1">
42 <constant>Invalid</constant>
43 </setHeader>
44 <to id="_Invalid" uri="file:target/messages/invalidOrders"/>
45 </when>
46 </route>
47 </camelContext>
</blueprint>

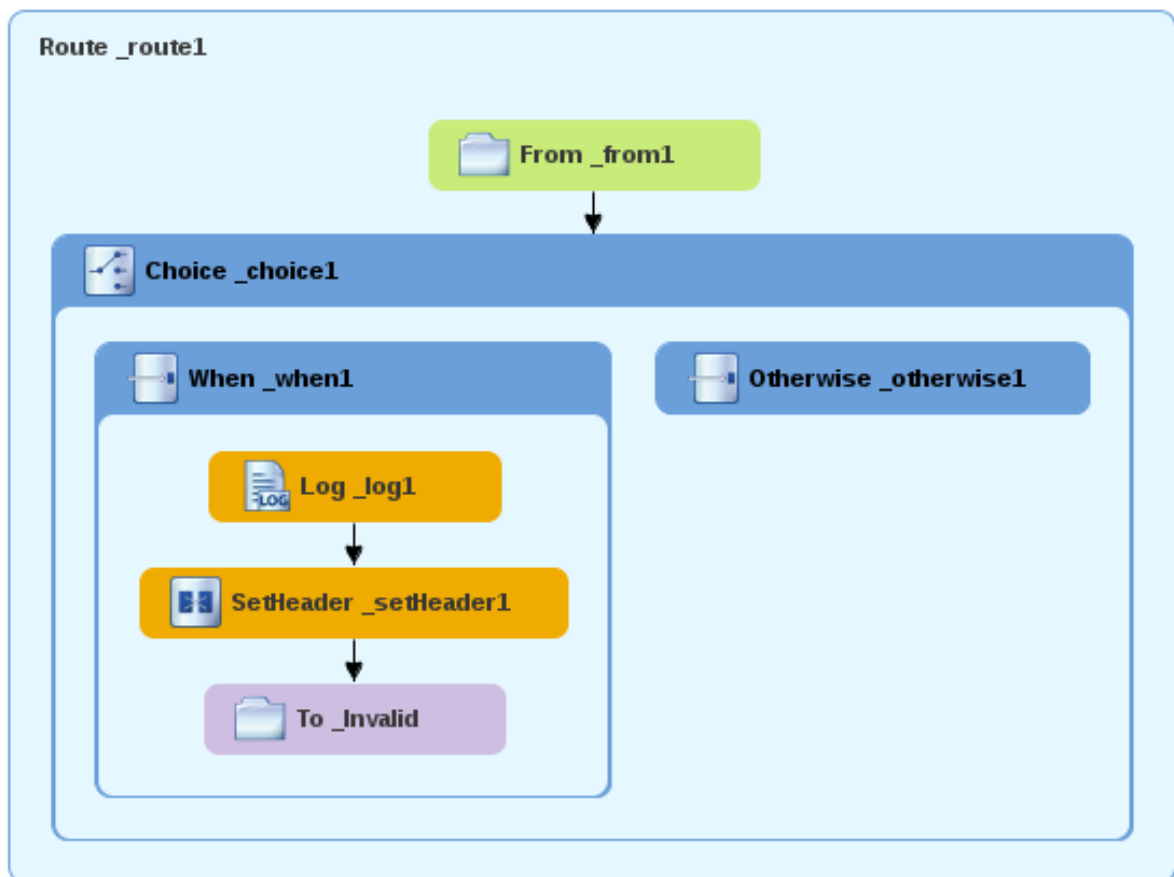
```

有効な注文を処理するためのブランチの追加と設定

これまでのところ、CBR は無効な注文 (数量の値が 10 より大きい注文) を含むメッセージを処理しません。

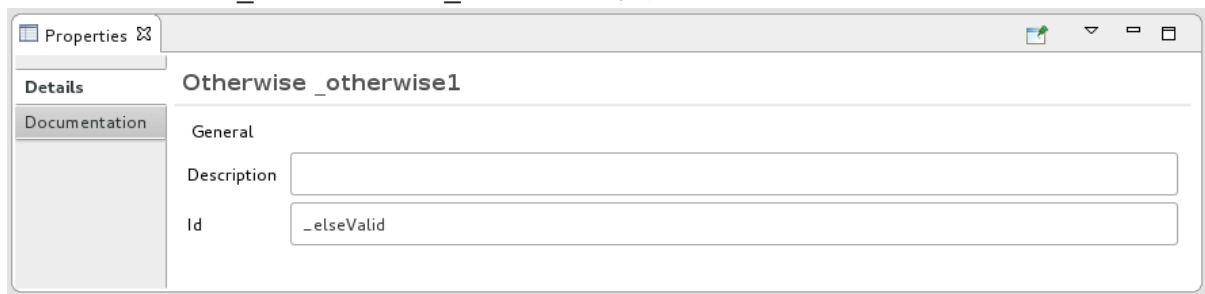
有効な注文 (つまり、**When_when1** ノードに設定された XPath 式と一致しない XML メッセージ) を処理するようにルートのブランチを追加および設定するには、次のようにします。

1. **Palette** で、**Routing** ドロワーを開き、**Otherwise** () パターンをクリックします。
2. キャンバスで **Choice_choice1** コンテナをクリックします。



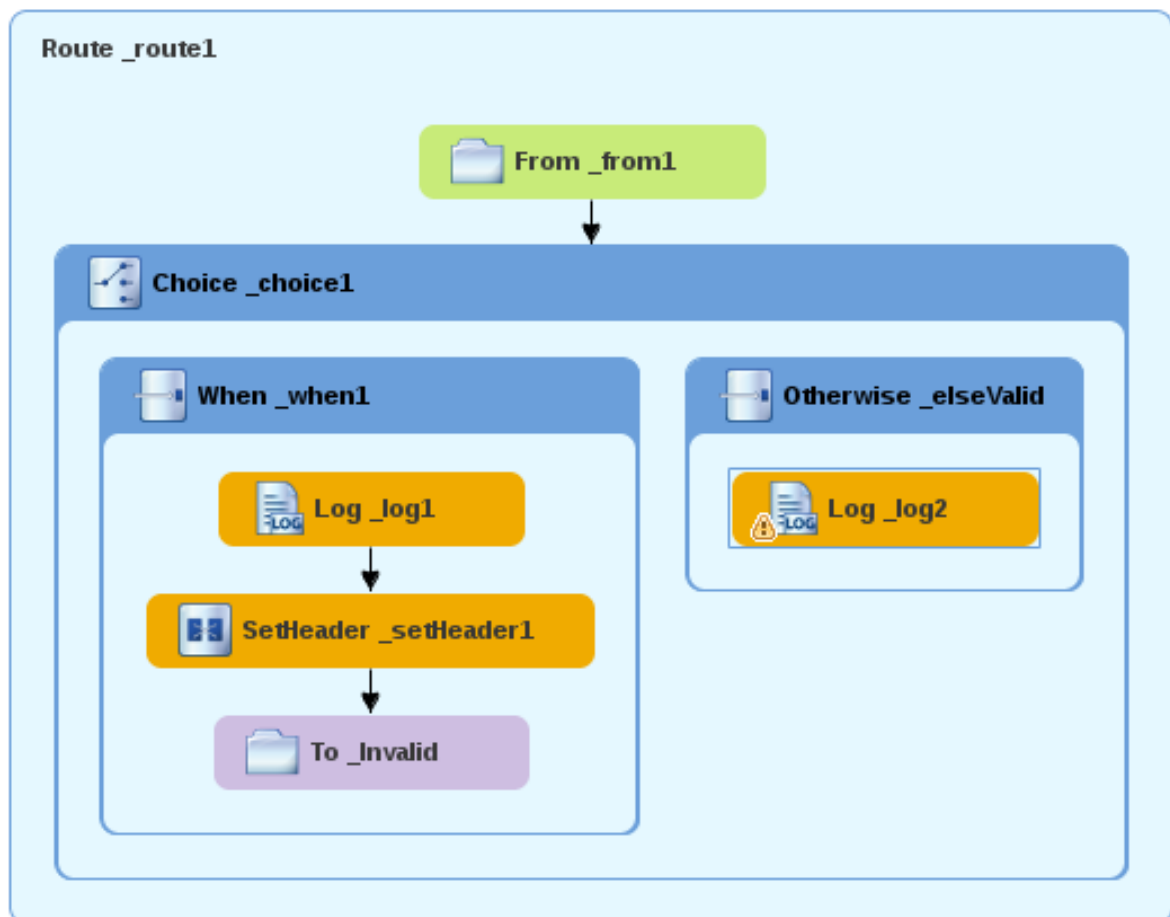
Choice_choice1 コンテナが拡張し、**Otherwise_otherwise1** ノードが追加されます。

- キャンバスで **Otherwise_otherwise1** ノードを選択し、Properties ビューでそのプロパティを開きます。
- Id フィールドで、**_otherwise1** を **_elseValid** に変更します。

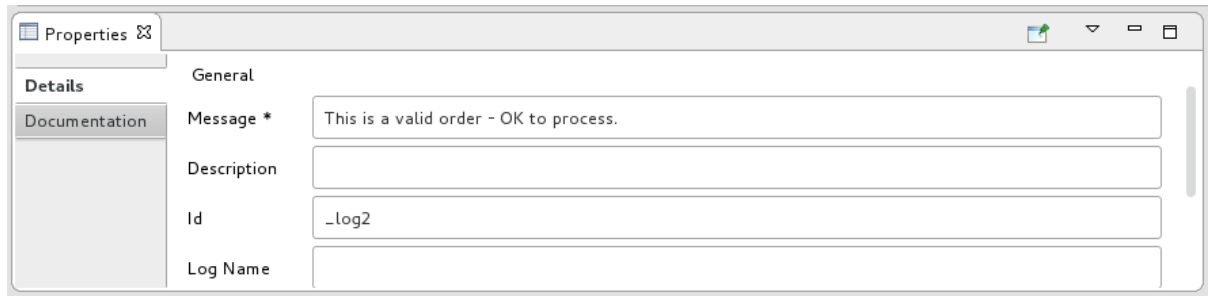


それ以外のブランチのロギングを設定するには、次のようにします。

- Palette で、Components ドロワーを開き、Log () コンポーネントをクリックします。
- キャンバスで **Otherwise_elseValid** ノードをクリックします。
Otherwise-elseValid コンテナが拡張し、**Log_log2** ノードが追加されます。



- キャンバスで **Log_log2** ノードを選択し、Properties ビューでそのプロパティを開きます。
- Message フィールドに、**This is a valid order - OK to process.** と入力します。

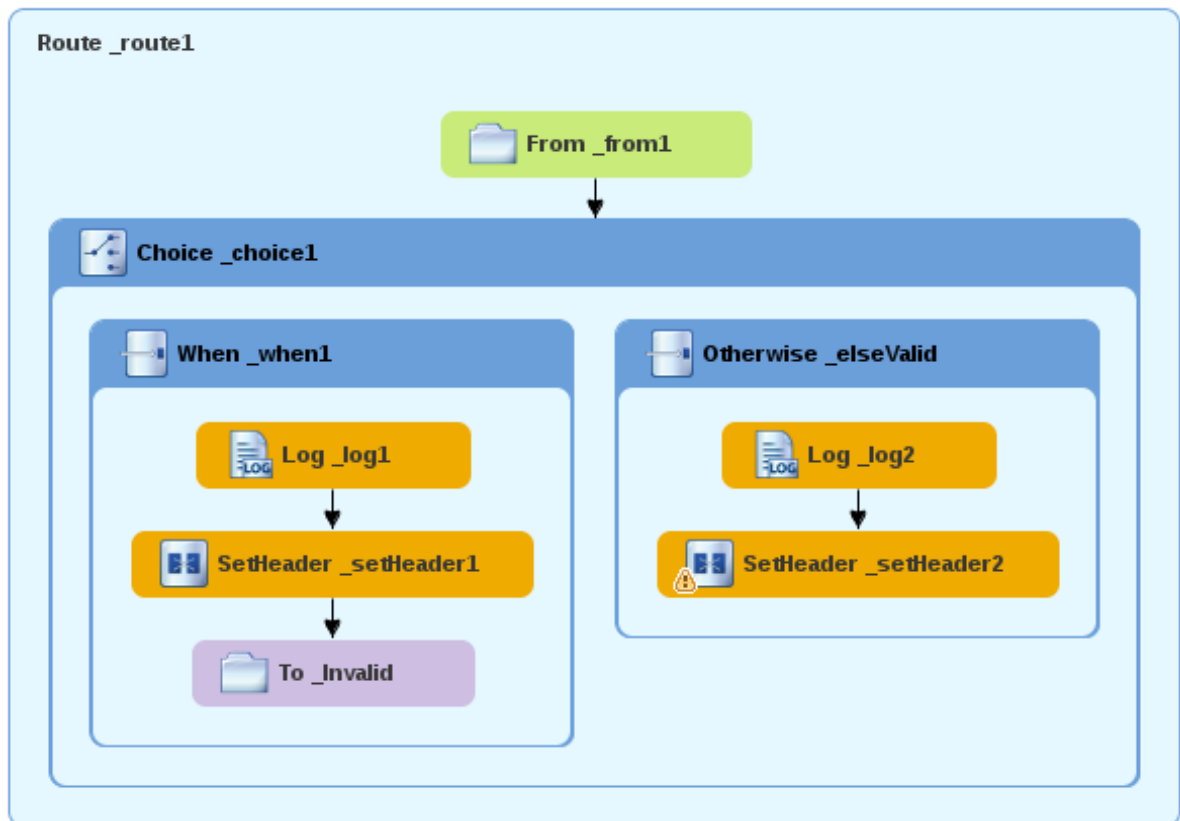


残りのプロパティはそのままにしておきます。


5. ルートを **Save** します。

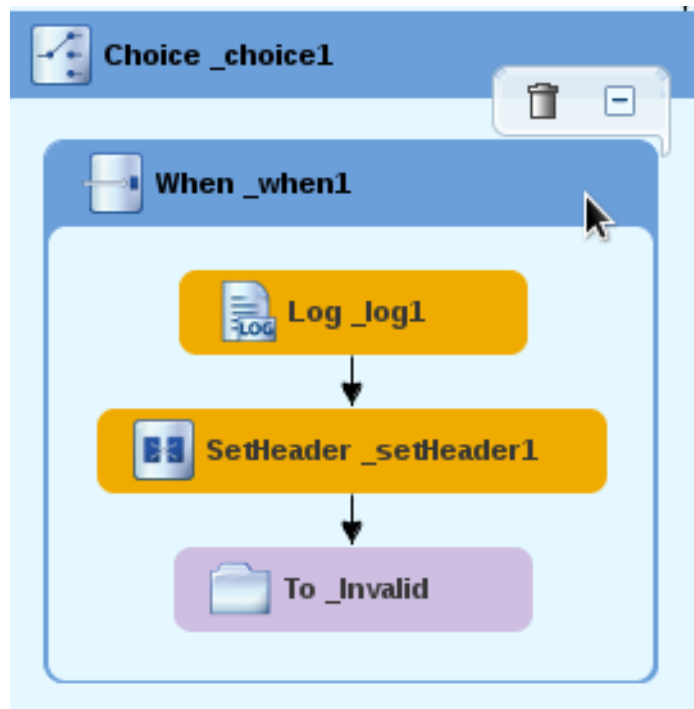
それ以外のブランチのメッセージヘッダーを設定するには、次の手順に従います。


1. **Palette** で、**Transformation** ドロワーを開き、**Set Header** パターンをクリックします。
2. キャンバスで **Log_log2** ノードをクリックします。
Otherwise_elseValid コンテナが拡張し、**SetHeader_setHeader2** ノードが追加されます。

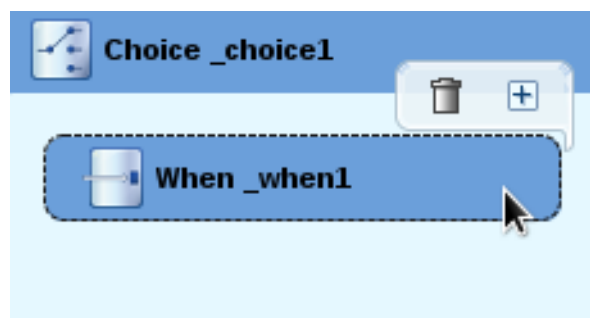


注記


ダイアグラムが混雑したときに、コンテナを閉じてスペースを解放できます。これを行うには、閉じるコンテナを選択し、そのコンテナの  ボタンをクリックします。

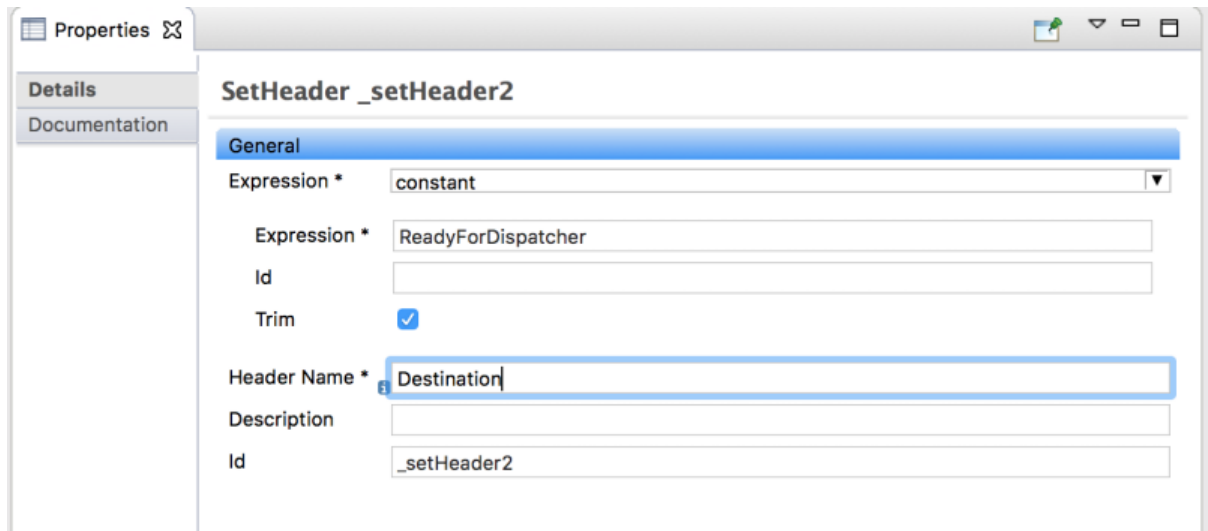


コンテナを再度開くには、コンテナを選択して、  ボタンをクリックします。




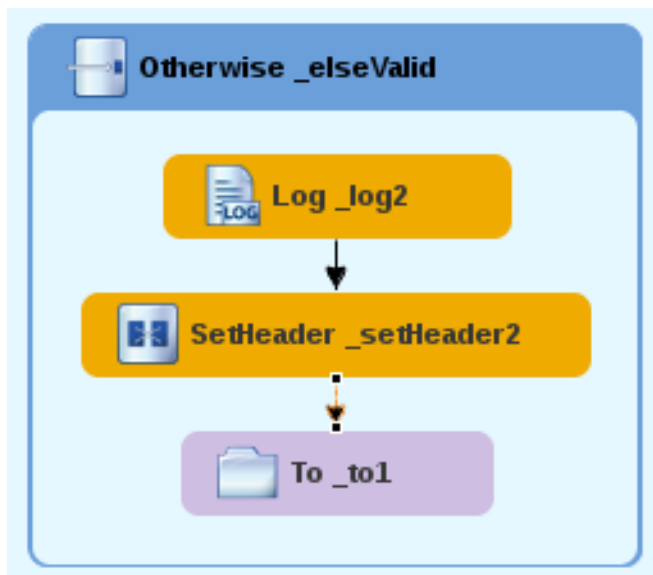
Design タブでコンテナを閉じたり開いたりしても、ルーティングコンテキストファイルには影響しません。変更はありません。

3. キャンバスで **SetHeader_setHeader2** ノードを選択し、**Properties** ビューでそのプロパティを開きます。
4. **Expression** フィールドの  ボタンをクリックして、使用可能な言語のリストを開き、**constant** を選択します。
5. インデントされた **Expression** フィールドに **ReadyForDispatcher** と入力します。
6. **Header Name** フィールドに **Destination** と入力します。
7. 残りのプロパティはそのままにしておきます。

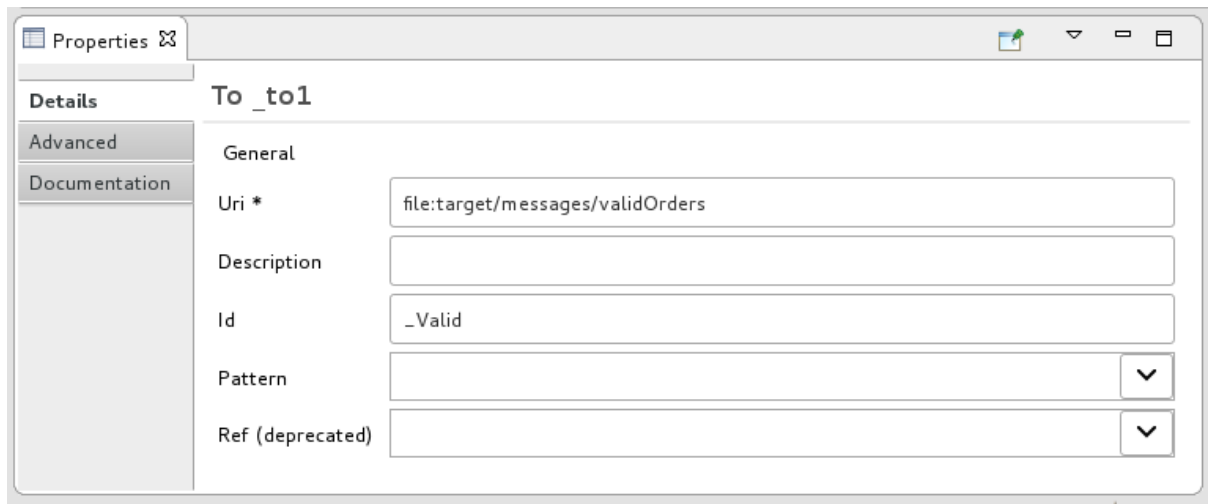


有効なメッセージのターゲットフォルダーを指定するには、次の手順に従います。

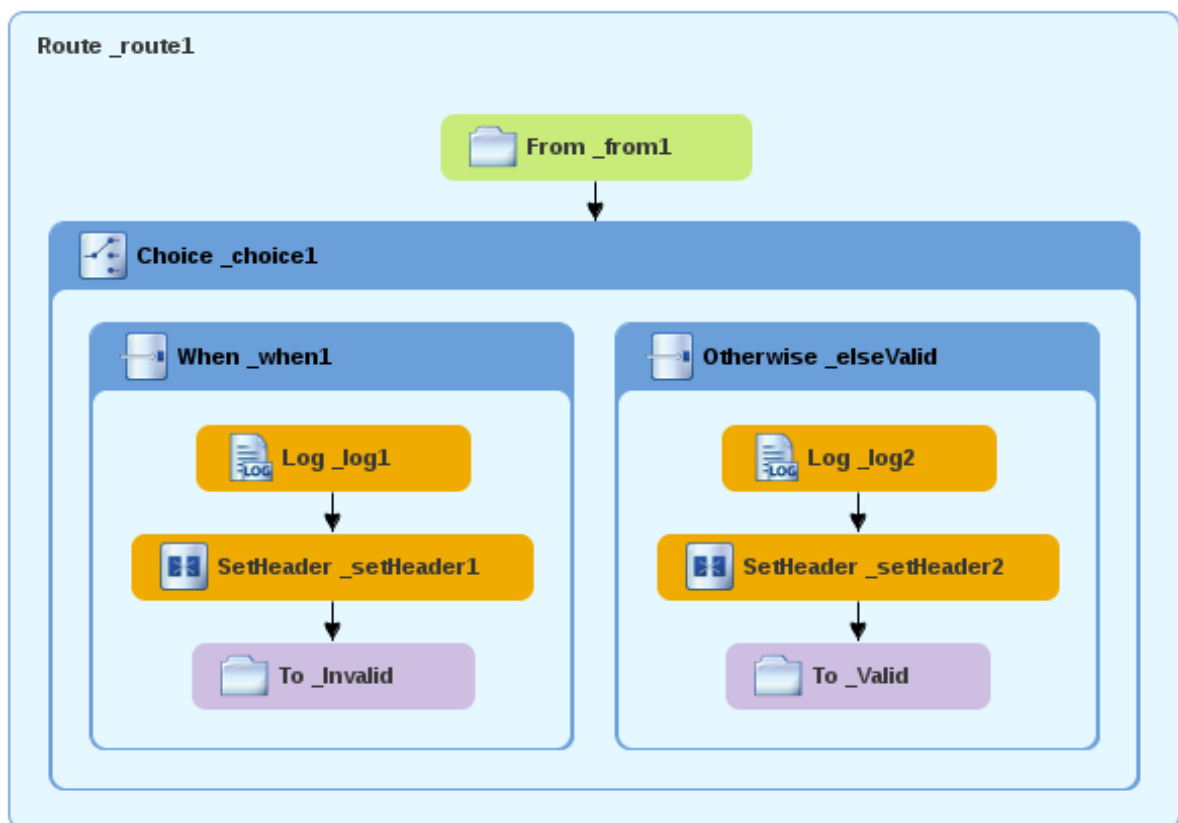
1. Palette で、Components ドロワーを開き、File () コンポーネントを選択します。
2. キャンバスで **SetHeader_setHeader2** ノードをクリックします。
Otherwise_elseValid コンテナが拡張し、**To_to1** ノードが追加されます。



3. キャンバスで **To_to1** ノードを選択し、Properties ビューでそのプロパティを開きます。
4. URI フィールドの `directoryName` を **target/messages/validOrders** に置き換え、Id フィールドに **_Valid** と入力します。



5. ルーティングコンテキストファイルを **Save** します。
完成したコンテンツベースのルーターは次のようになります。



6. キャンバスの左下にある **Source** タブをクリックして、ルート XML を表示します。

```
<?xml version="1.0" encoding="UTF-8"?>
<blueprint xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.osgi.org/xmlns/blueprint/v1.0.0
    https://www.osgi.org/xmlns/blueprint/v1.0.0/blueprint.xsd
    http://camel.apache.org/schema/blueprint
    http://camel.apache.org/schema/blueprint/camel-blueprint.xsd">

<camelContext id="_context1" xmlns="http://camel.apache.org/schema/blueprint">
  <route id="_route1">
    <from id="_from1" uri="file:src/data?noop=true"/>
```

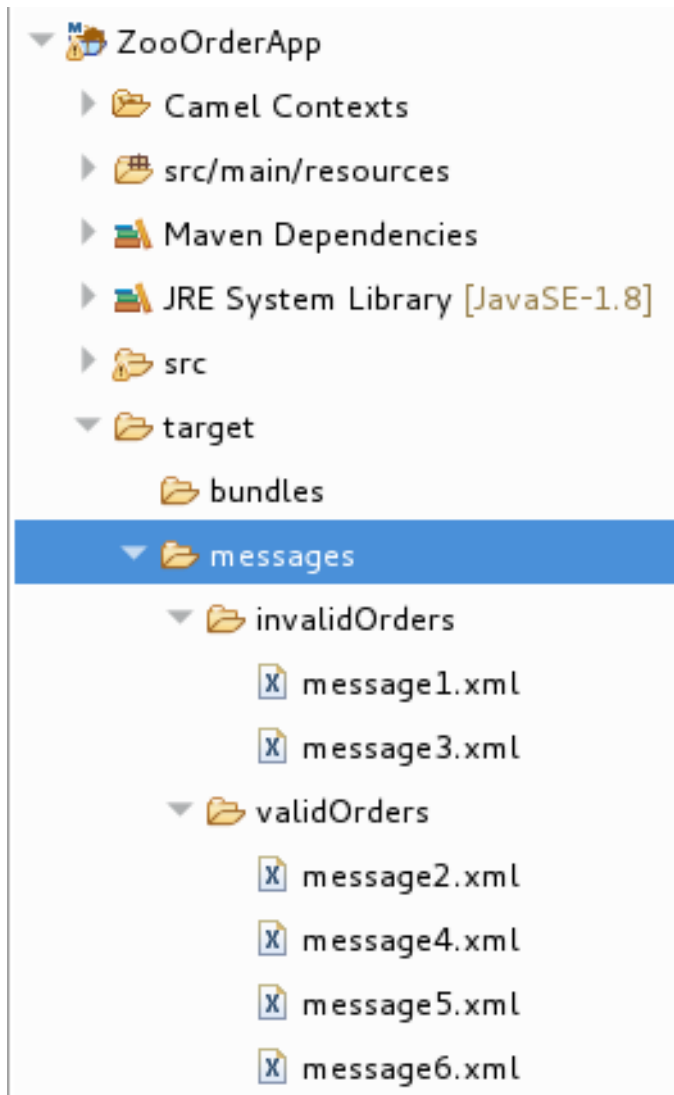
```
<choice id="_choice1">
  <when id="_when1">
    <xpath>/order/orderline/quantity/text() > 10</xpath>
    <log id="_log1" message="The quantity requested exceeds the maximum
allowed - contact customer."/>
    <setHeader headerName="Destination" id="_setHeader1">
      <constant>Invalid</constant>
    </setHeader>
    <to id="_Invalid" uri="file:target/messages/invalidOrders"/>
  </when>
  <otherwise id="_elseValid">
    <log id="_log2" message="This is a valid order - OK to process."/>
    <setHeader headerName="Destination" id="_setHeader2">
      <constant>ReadyForDispatcher</constant>
    </setHeader>
    <to id="_Valid" uri="file:target/messages/validOrders"/>
  </otherwise>
</choice>
</route>
</camelContext>
</blueprint>
```

CBR の検証

「[ルートを実行する](#)」チュートリアルで説明したように新しいルートを実行し、**Console** ビューでログメッセージを確認することができます。

実行後、ルートが正しく実行されたかどうかを確認するには、**Project Explorer** でターゲットの宛先フォルダーを確認します。

1. **ZooOrderApp** を選択します。
2. それを右クリックしてコンテキストメニューを開き、**Refresh** を選択します。
3. プロジェクトの root ノード (**ZooOrderApp**) で **target/messages/** フォルダーを見つけ、デブ
ロイメントします。



4. **target/messages/invalidOrders** フォルダに **message1.xml** および **message3.xml** が含まれることを確認します。
これらのメッセージでは、**quantity** 要素の値は 10 を超えます。
5. **target/messages/validOrders** フォルダに有効な注文が含まれる 4 つのメッセージファイルが含まれていることを確認します。
 - **message2.xml**
 - **message4.xml**
 - **message5.xml**
 - **message6.xml**
これらのメッセージでは、**quantity** 要素の値は 10 以下です。



注記

メッセージの内容を表示するには、各メッセージをダブルクリックして、ルートエディターの XML エディターで開きます。

次のステップ

次のチュートリアル [6章 ルーティングコンテキストに別のルートを追加する](#) では、有効な注文メッセージをさらに処理する 2 番目のルートを追加します。

第6章 ルーティングコンテキストに別のルートを追加する

このチュートリアルでは、**ZooOrderApp** プロジェクトの **blueprint.xml** ファイルの Camel コンテキストに 2 番目のルートを追加する方法を説明します。2 番目のルート:

- 最初のルートの **otherwise** の場合は分岐の終端から直接メッセージ (有効な注文) を受け取ります。
- お客様の国に従って有効なメッセージを並べ替えます。
- 各メッセージを **ZooOrderApp/target/messages** フォルダの対応する **国** フォルダに送信します。たとえば、シカゴ動物園からの注文は **USA** フォルダにコピーされます。

ゴール

このチュートリアルでは、次のタスクを完了します。

- 2 番目のルートに直接接続するために既存のルートを再設定します
- Camel コンテキストに 2 番目のルートを追加します
- 最初のルートのブランチから直接メッセージを受信するように 2 番目のルートを設定します
- コンテンツベースのルーターを 2 番目のルートに追加します
- 2 番目のルートのコンテンツベースルーターの各出力ブランチに、メッセージヘッダー、ロギング、およびターゲット宛先を追加して設定します

前提条件

このチュートリアルを開始するには、次のいずれかの結果である **ZooOrderApp** プロジェクトが必要です。

- [5章コンテンツベースのルーターの追加チュートリアル](#)を完了します。
または
- [2章環境の設定](#)チュートリアルを完了し、「[リソースファイルについて](#)」に記載されているように、プロジェクトの **blueprint.xml** ファイルを、提供される **blueprintContexts/blueprint2.xml** ファイルに置き換える。

既存のルートのエンドポイントを再設定する

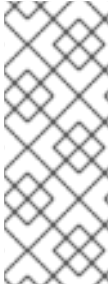
既存のルートは、すべての有効な注文を **target/messages/validOrders** フォルダに送信します。

このセクションでは、既存のルートの **Otherwise_elseValid** ブランチのエンドポイントを再設定して、代わりに 2 番目のルート (次のセクションで作成します) に接続します。

2 番目のルートと直接接続するように既存のルートを設定するには:

1. ルートエディターで **ZooOrderApp/src/main/resources/OSGI-INF/blueprint/blueprint.xml** を開きます。
2. キャンバスで、**Route_route1** コンテナを選択して、**Properties** ビューでそのプロパティを開きます。

3. **Shutdown Route** プロパティまで下にスクロールして、**Default** を選択します。
4. キャンバスで、ターミナルファイルノード **To_Valid** を選択して、そのプロパティを **Properties** ビューに表示します。
5. **Uri** フィールドで既存のテキストを削除して、**direct:OrderFulfillment** を入力します。
6. **Id** フィールドに **_Fulfill** を入力します。




注記

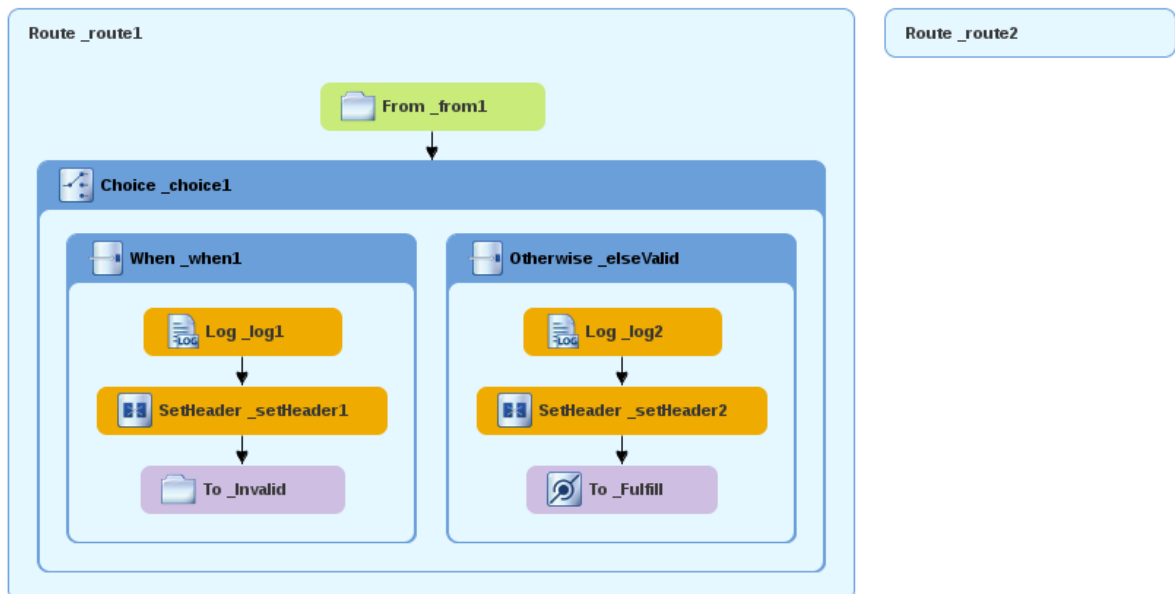
既存の **To_Valid** 端末ファイルノードを再使用する代わりに、**Components** → **Direct** コンポーネントに移動してこれを置き換えることができます (再使用する **To_Valid** ノードと同じプロパティ値で設定する)。

Direct コンポーネントの詳細は、[Apache Camel コンポーネントリファレンス](#) を参照してください。

2 番目のルートを追加する

ルーティングコンテキストに別のルートを追加するには:

1. **Palette** で、**Routing** ドロワーを開き、**Route** () パターンをクリックします。
2. キャンバスで **Route_route1** コンテナの右をクリックします。

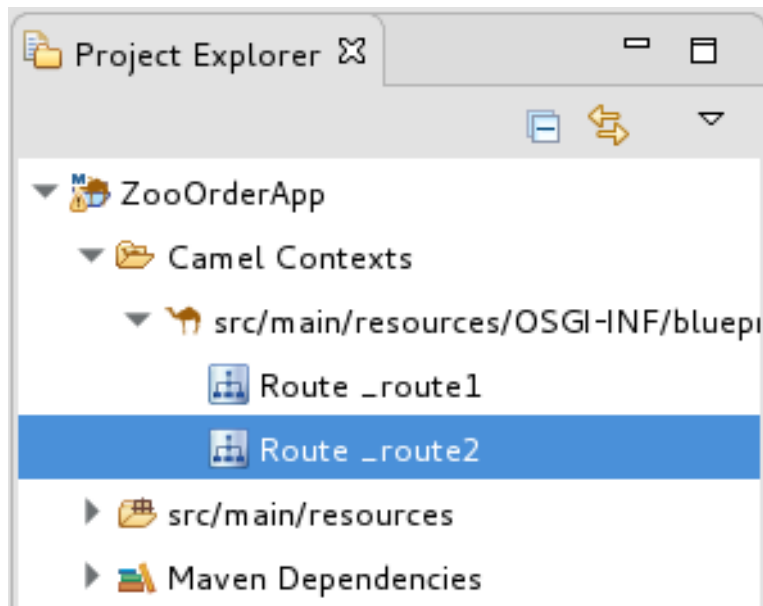


キャンバスで **Route** パターンが **Route_route2** コンテナノードになります。

3. **Route_route2** コンテナノードをクリックして、**Properties** ビューでそのプロパティを表示します。プロパティはそのままにしておきます。
4. ファイルを**保存**します。

注記


ルーティングコンテキストが複雑になるにつれて、作業中はルートエディターを個々のルートに集中させることができます。これを実行するには、**Project Explorer** で、ルートエディターがキャンバスで表示するルートをクリックします (例: **Route_route2**)。

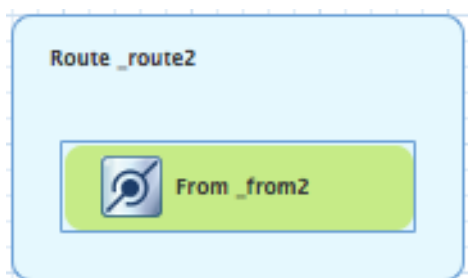


キャンバスにルーティングコンテキストにあるすべてのルートを表示するには、**Camel Contexts** フォルダの上部にあるプロジェクトの **.xml** コンテキストファイルエントリ (**src/main/resources/OSGI-INF/...**) をダブルクリックします。

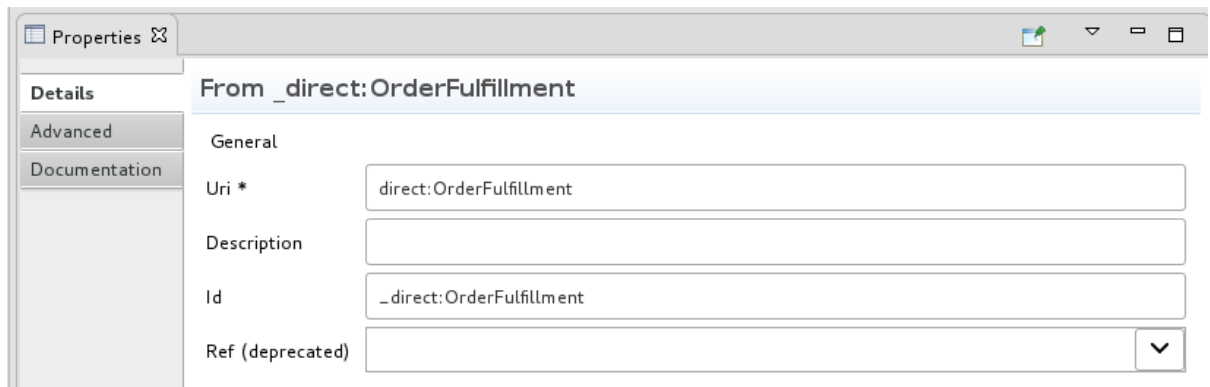
米国の注文を処理するための CHOICE ブランチの設定


このセクションでは、ルートに Choice ブランチを追加し、新しい **target/messages/validOrders/USA** フォルダに USA からの注文を送信するようにルートを設定します。また、メッセージヘッダーとログファイルコンポーネントを設定します。

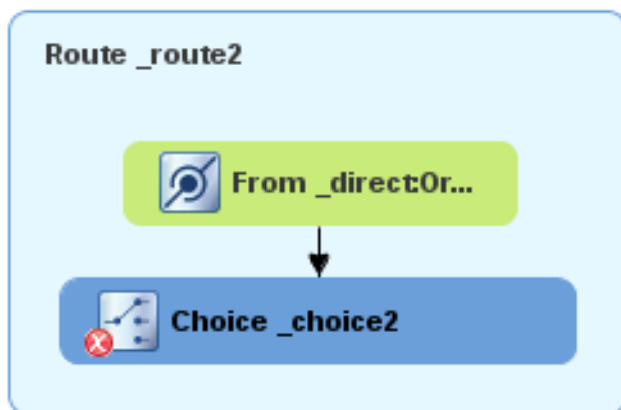
1. **Palette** で、**Components** ドロワーを開き、**Direct** コンポーネントを選択します ()。
2. キャンバスで **Route_route2** コンテナをクリックします。
Route_route2 コンテナが拡張し、**Direct** コンポーネント (**From_from2** ノード) が追加されます。




3. キャンバスで **From_from2** ノードをクリックし、**Properties** ビューでそのプロパティを開きます。
4. **Uri** フィールドで、**name** (**direct:** の後) を **OrderFulfillment** に置き換え、**Id** フィールドで、**_direct:OrderFulfillment** と入力します。

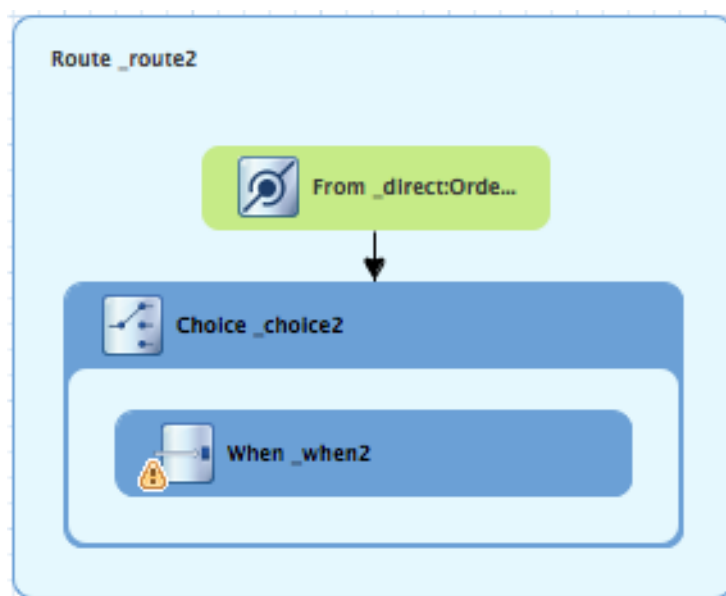


5. Palette で、Routing ドロワーを開き、Choice () パターンを選択します。
6. キャンバスで **From_direct:OrderFulfillment** ノードをクリックします。
Route_route2 コンテナが拡張し、**Choice_choice2** ノードが追加されます。

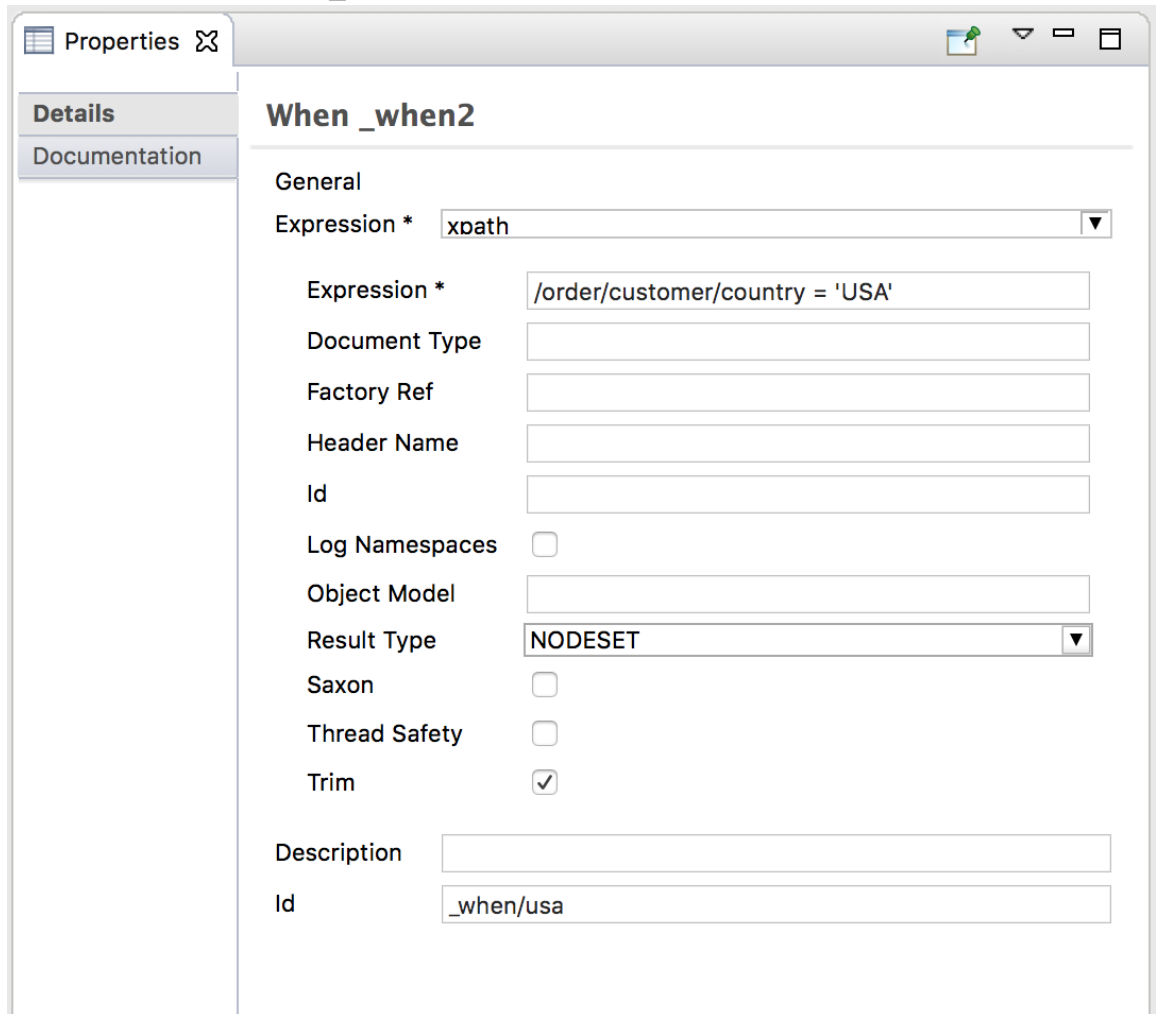


Properties ビューで、**Choice_choice2** ノードのプロパティをそのままにしておきます。


7. パレットで、Routing ドロワーを開き、When () パターンを選択します。
8. キャンバスで **Choice_choice2** ノードをクリックします。
Choice_choice2 コンテナが拡張し、**When_when2** ノードが追加されます。

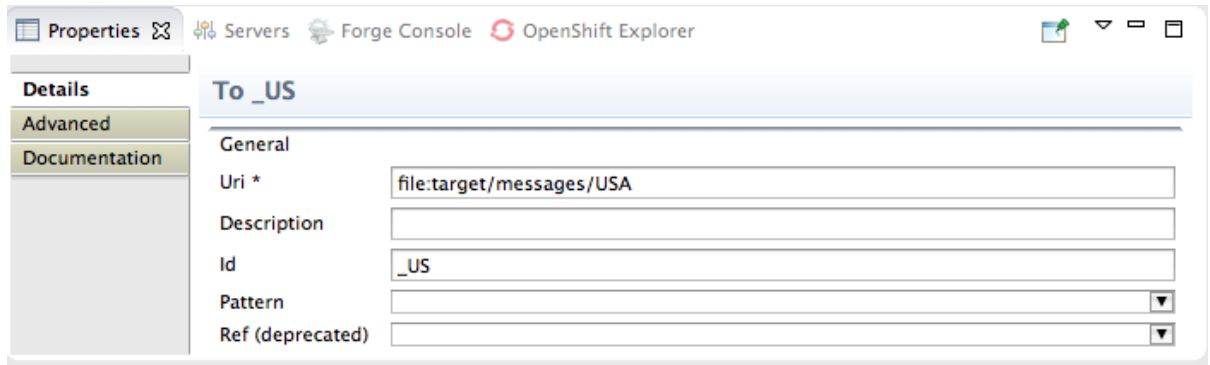


9. キャンバスで **When_when2** ノードを選択し、**Properties** ビューでそのプロパティを開きます。
10. **When_when2** ノードのプロパティを以下のように設定します。
 - **Expression** ドロップダウンリストから **xpath** を選択します。
 - インデントされた **Expression** フィールドに、**/order/customer/country = 'USA'** と入力します。
 - **Trim** を有効のままにします。
 - 2 番目の **Id** フィールドに **_when/usa** と入力します。



The screenshot shows the 'Properties' window for a 'When_when2' node. The 'Details' tab is active. Under the 'General' section, the 'Expression *' dropdown is set to 'xpath'. Below it, the 'Expression *' text field contains the XPath expression '/order/customer/country = 'USA''. Other fields like 'Document Type', 'Factory Ref', 'Header Name', and 'Id' are empty. The 'Log Namespaces', 'Object Model', 'Saxon', and 'Thread Safety' checkboxes are unchecked. The 'Trim' checkbox is checked. The 'Result Type' dropdown is set to 'NODESET'. At the bottom, the 'Description' field is empty, and the 'Id' field contains '_when/usa'.

11. **Palette** で、**Components** ドロワーを開き、**File** () コンポーネントを選択します。
12. キャンバスで **When_when/usa** コンテナをクリックします。
When_when/usa コンテナが拡張し、**To_to1** ノードが追加されます。
13. **Properties** ビューで:

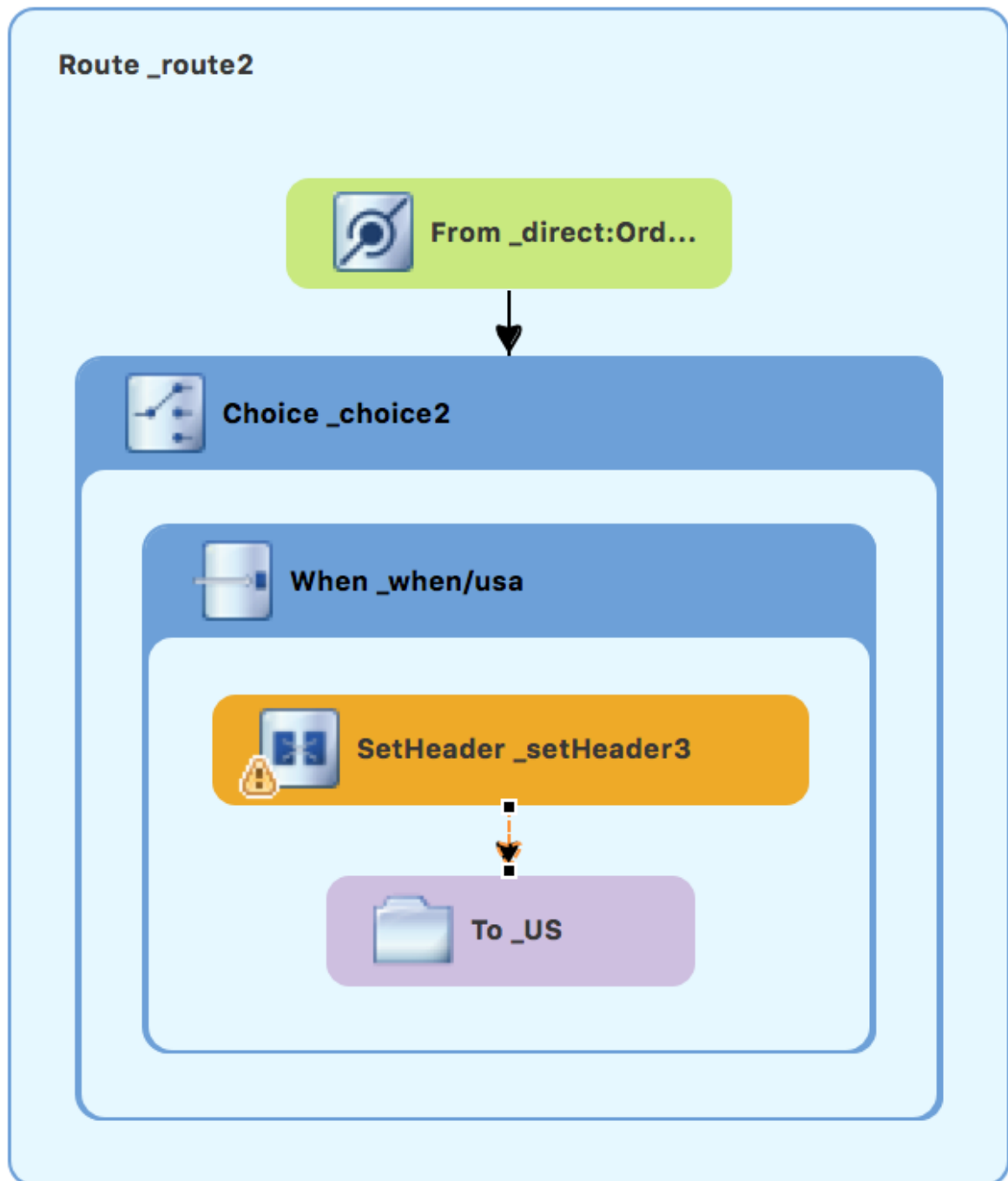


- Uri フィールドで、**directoryName** を **target/messages/validOrders/USA** に置き換えます。
- Id フィールドに **_US** と入力します。

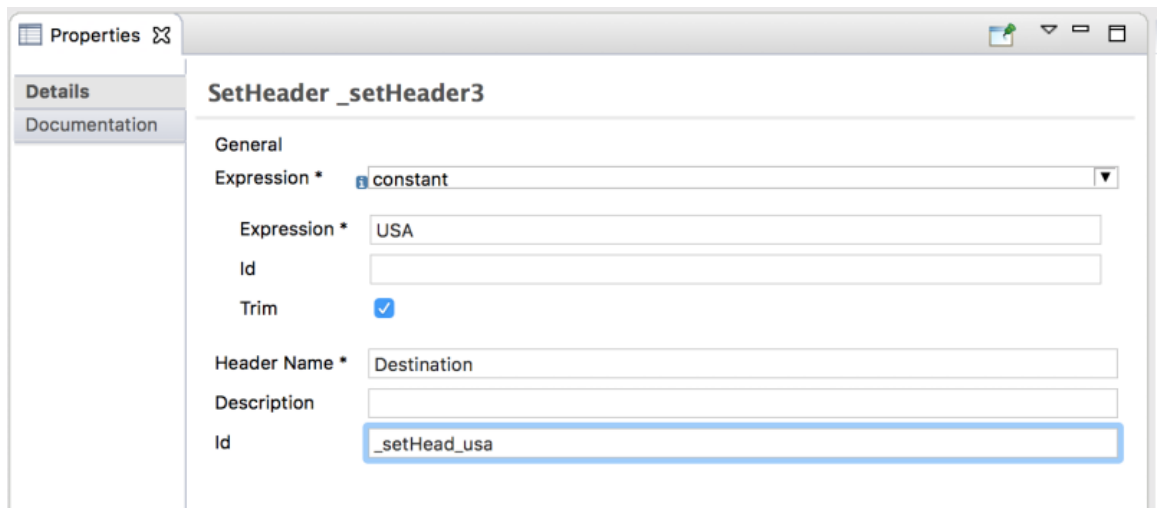
14. ファイルを**保存**します。

メッセージヘッダーを設定し、ログコンポーネントを追加するには:

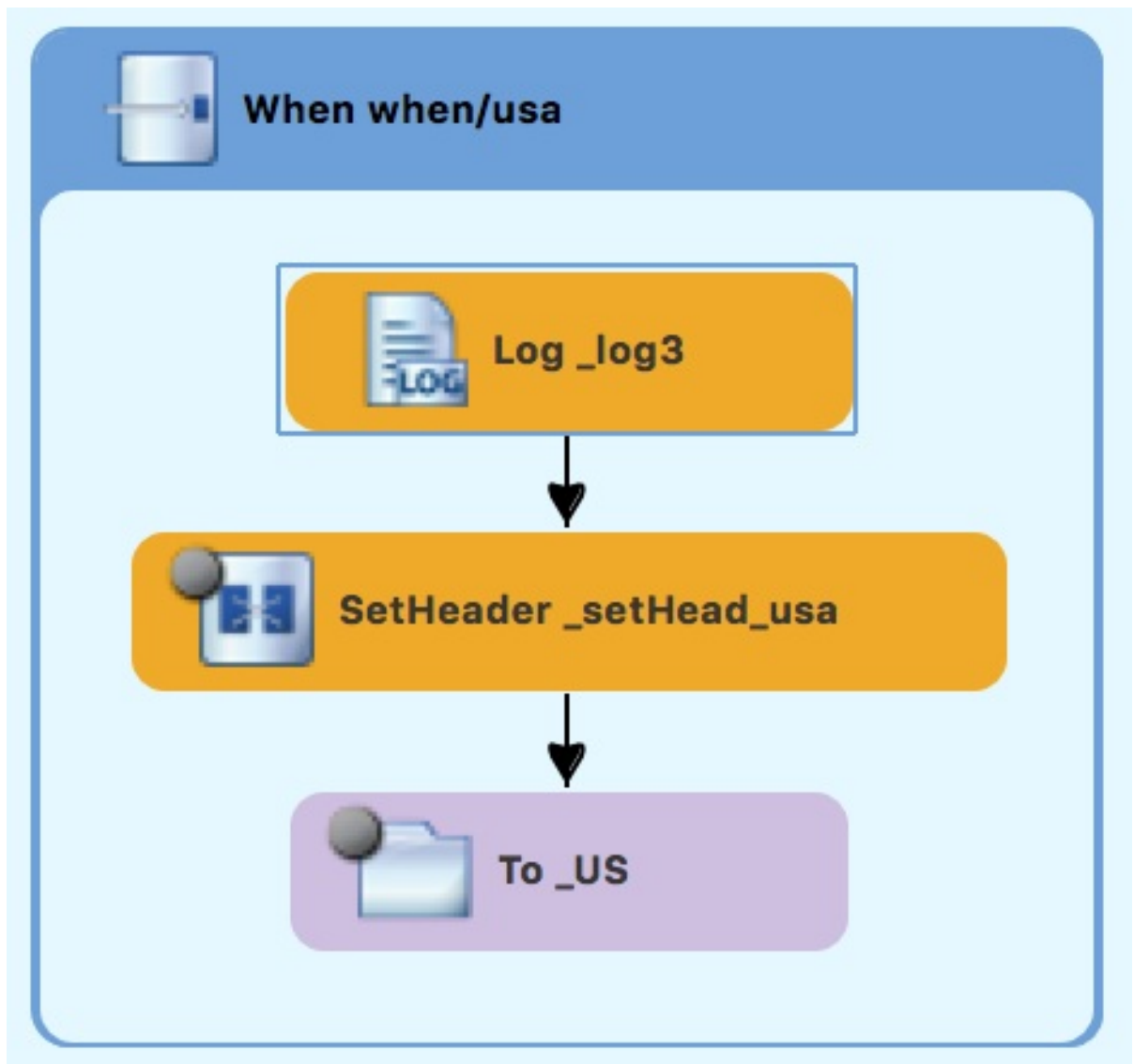
1. **Palette** で、**Transformation** ドロワーを開き、**Set Header** を選択します。
2. キャンバスで **When_when/usa** ノードをクリックします。
When_when/usa コンテナが拡張し、**SetHeader_setHeader3** ノードが追加されます。



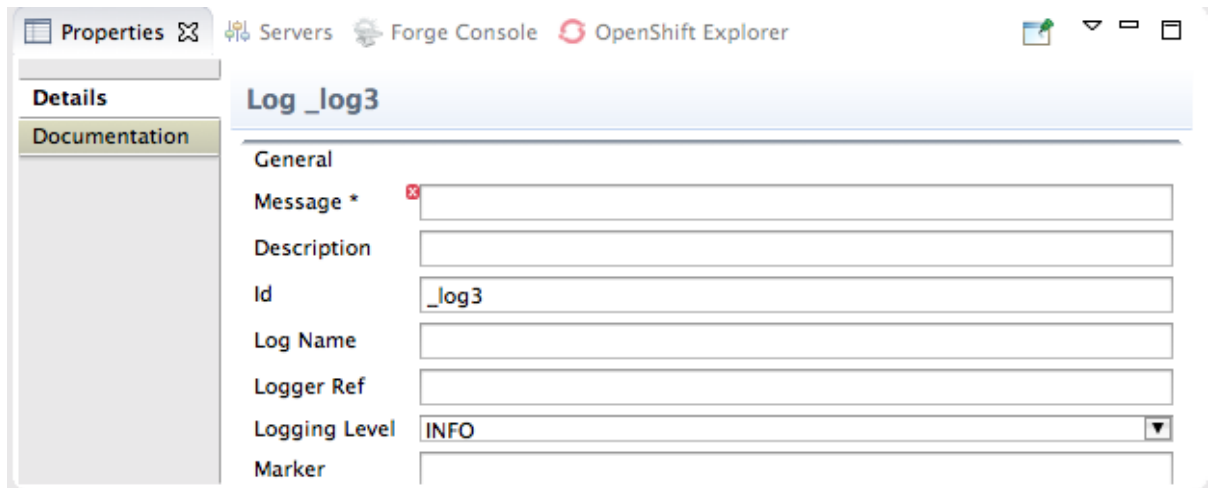
3. キャンバスで **SetHeader_setHeader3** ノードを選択し、**Properties** ビューでそのプロパティを開きます。
4. ノードのプロパティを次のように設定します。
 - **Expression** ドロップダウンメニューから、**constant** を選択します。
 - インデントされた **Expression** フィールドに **USA** と入力します。
 - **Trim** を有効のままにします。
 - **Header Name** フィールドに **Destination** と入力します。
 - 2 番目の **Id** フィールドに **_setHead_usa** と入力します。



5. Palette で、Components ドロワーを開き、Log コンポーネント () を選択します。
6. キャンバスで **SetHeader** ノードの上をクリックします。
When_when/usa コンテナが拡張し、**Log_log3** ノードが追加されます。



7. キャンバスで **Log_log3** ノードを選択し、Properties ビューでそのプロパティを開きます。

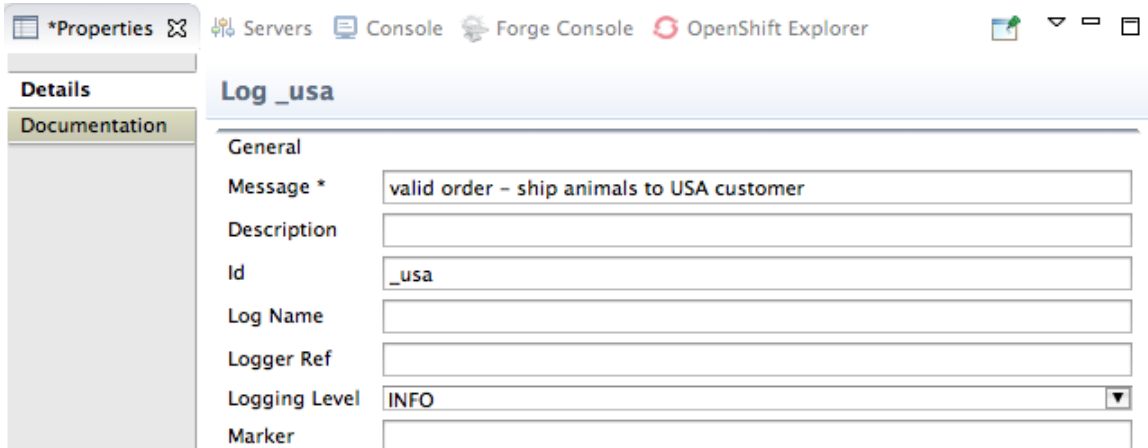


The screenshot shows the 'Properties' view for a log configuration named 'Log_log3'. The 'General' section contains the following fields:

Field	Value
Message *	
Description	
Id	_log3
Log Name	
Logger Ref	
Logging Level	INFO
Marker	

8. Properties ビューで:

- Message フィールドに **Valid order - ship animals to USA customer** と入力します。
- Id フィールドに **_usa** と入力します。
- Logging Level はそのままにします。

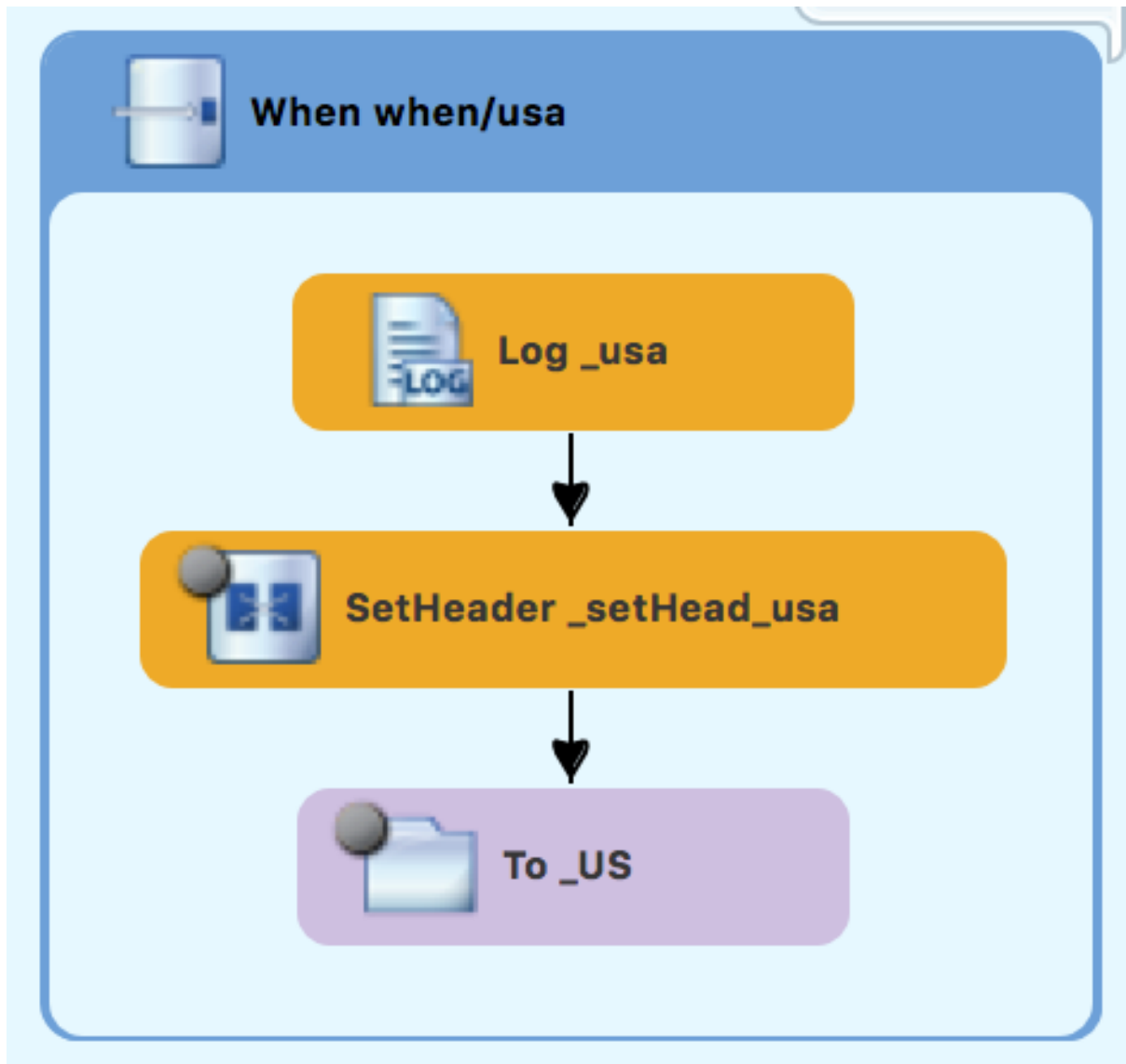


The screenshot shows the 'Properties' view for a log configuration named 'Log_usa'. The 'General' section contains the following fields:

Field	Value
Message *	valid order - ship animals to USA customer
Description	
Id	_usa
Log Name	
Logger Ref	
Logging Level	INFO
Marker	


9. ファイルを**保存**します。

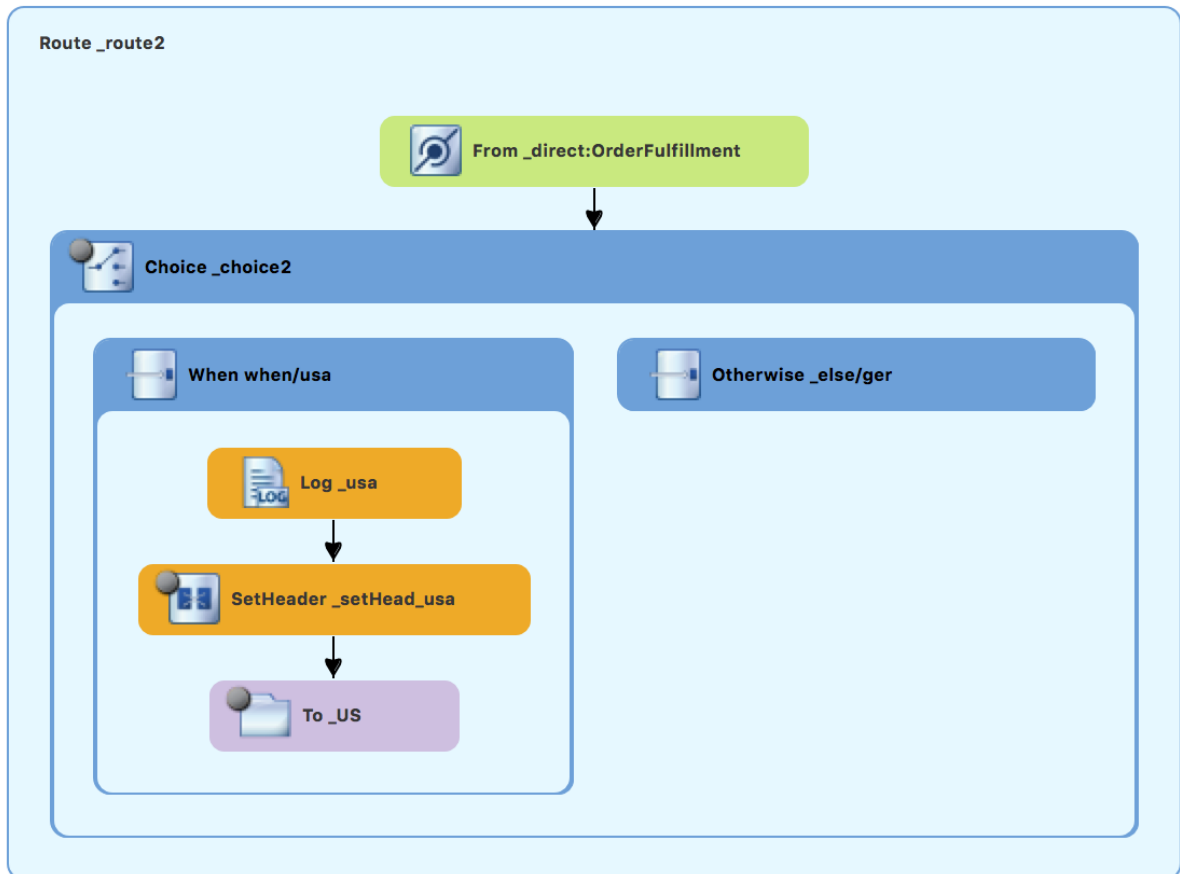
Route_route2 の USA ブランチは以下ようになります。




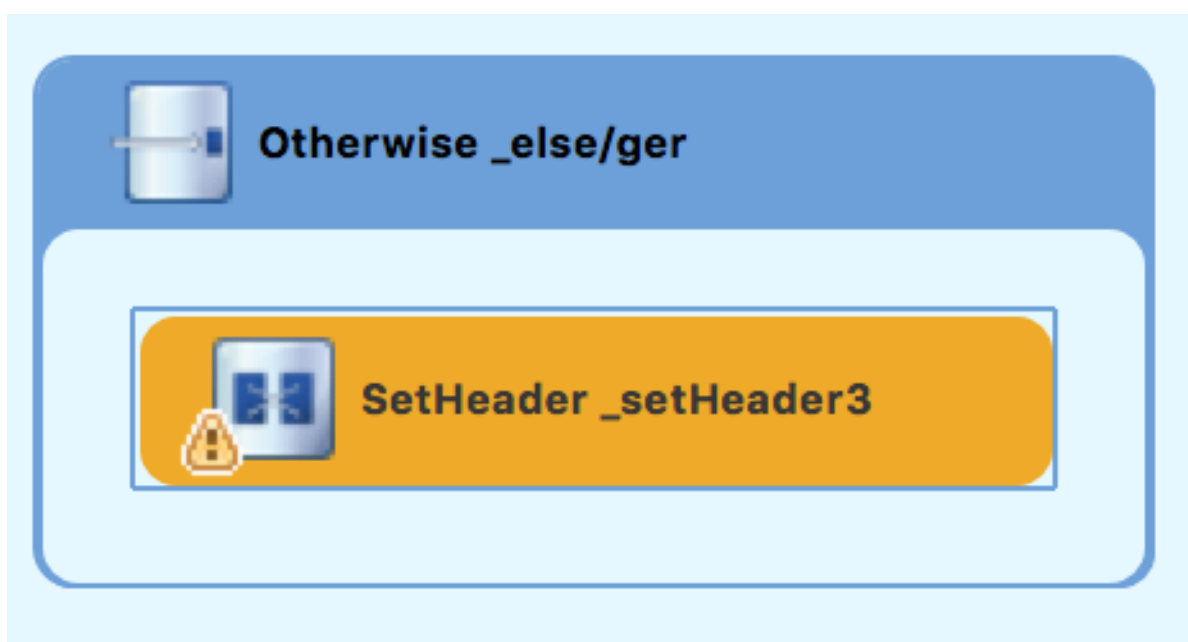
ドイツの注文を処理するための OTHERWISE ブランチの設定


Route_route2 がキャンバスに表示される場合:

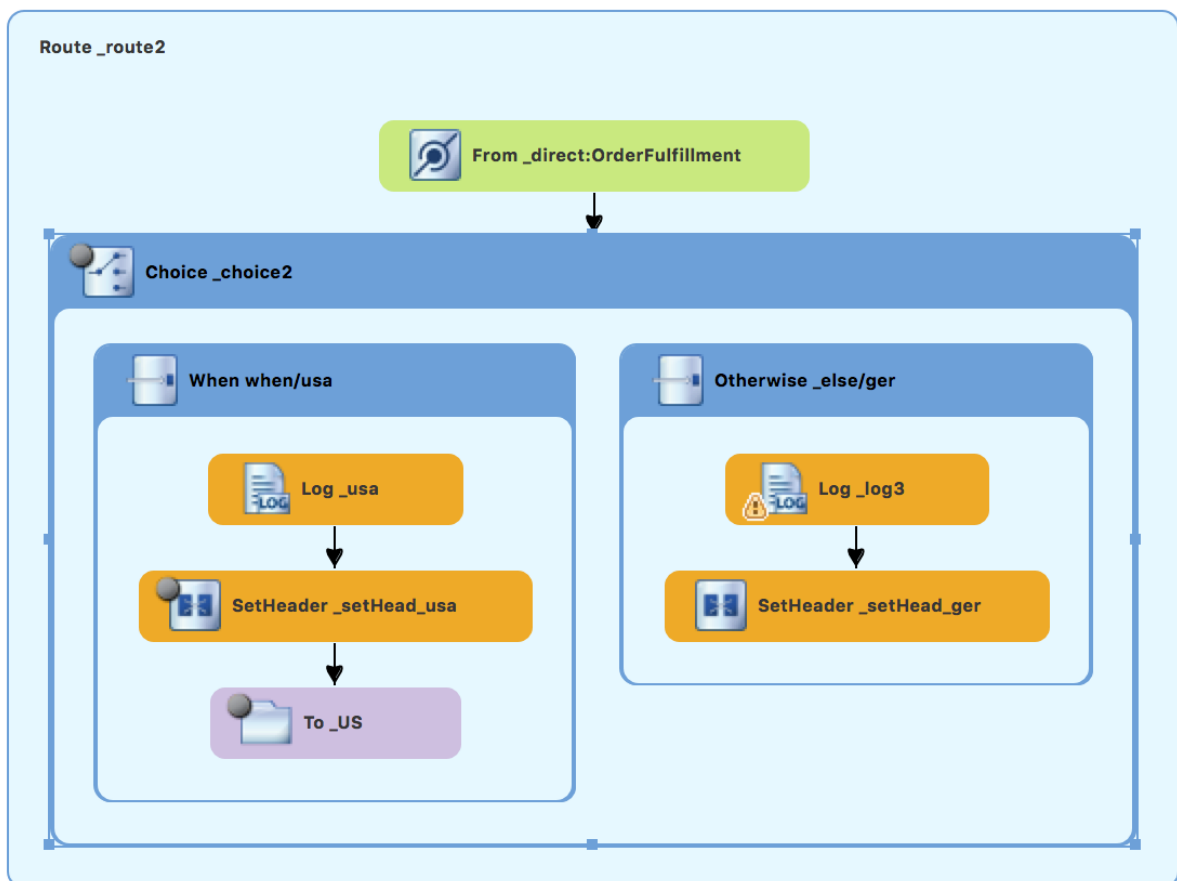
1. Palette で、Routing ドロワーを開き、Otherwise のパターン () を選択します。
2. キャンバスで **Choice_choice2** コンテナをクリックします。
Choice_choice2 コンテナが拡張し、**Otherwise_otherwise1** ノードが追加されます。



3. **Otherwise_otherwise1** ノードを選択し、**Properties** ビューでそのプロパティを開きます。
4. **Properties** ビューで、**Id** フィールドに **_else/ger** と入力します。
5. **Palette** で、**Transformation** ドロワーを開き、**Set Header** パターン () を選択します。
6. キャンバスで **Otherwise_else/ger** ノードをクリックします。
Otherwise_else/ger コンテナが拡張し、**SetHeader_setHeader3** ノードが追加されます。




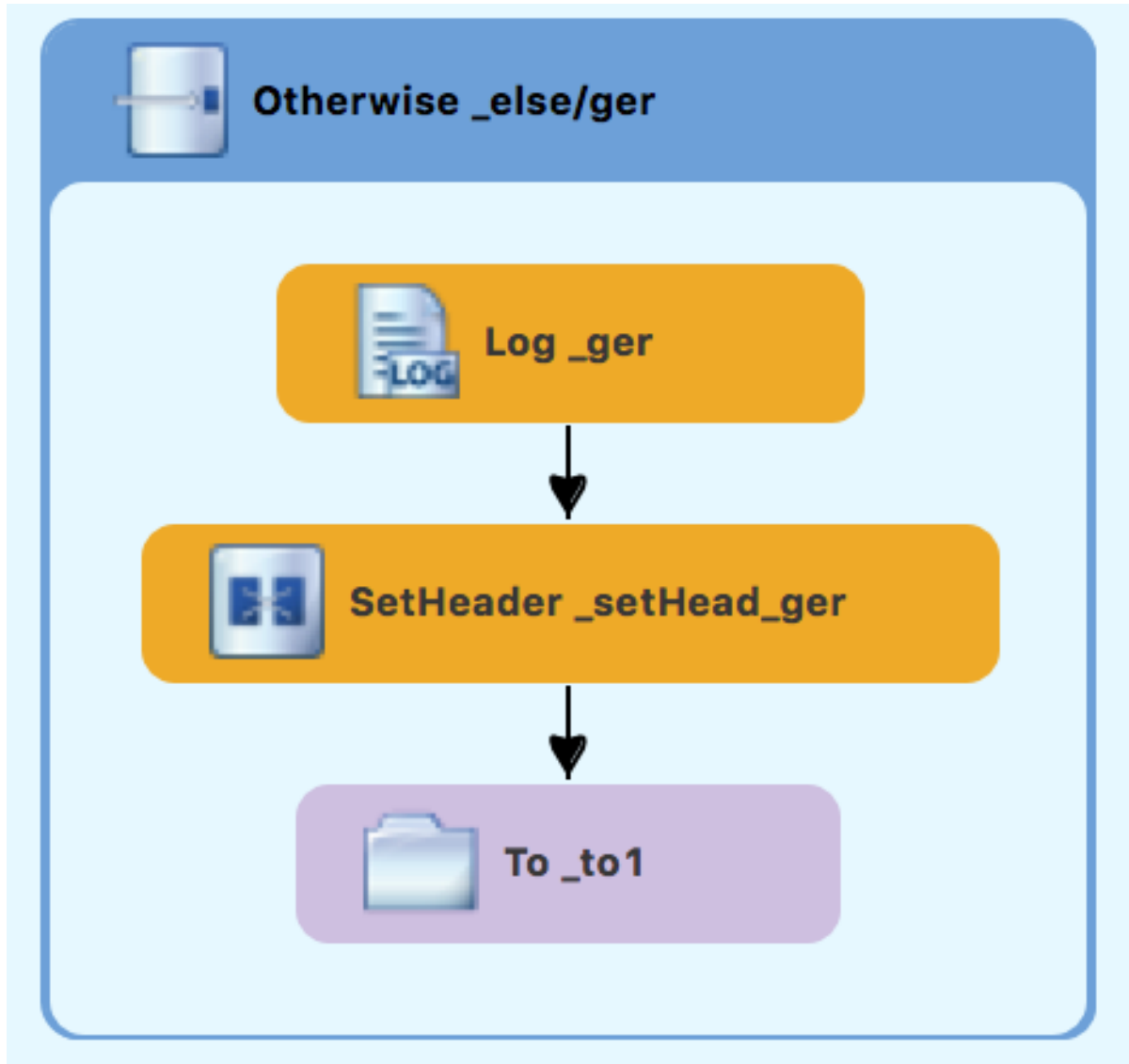
7. キャンバスで **SetHeader_setHeader3** ノードを選択し、**Properties** ビューでそのプロパティを開きます。
8. **Properties** ビューで:
 - **Expression** ドロップダウンリストから、**constant** を選択します。
 - 2 番目の **Expression** フィールドに **Germany** と入力します。
 - **Trim** はそのままにしておきます。
 - **Header Name** フィールドに **Destination** と入力します。
 - 2 番目の **Id** フィールドに **_setHead_ger** と入力します。
9. **Palette** で、**Components** ドロワーを開き、**Log** パターン () を選択します。
10. キャンバスで、**SetHeader_setHead_ger** ノードの下をクリックします。**Otherwise_else/ger** コンテナが拡張し、**Log_log3** ノードが追加されます。必要に応じて、コネクタエラーを **Log_log3** ノードから **SetHeader_setHead_ger** ノードにドラッグします。



11. キャンバスで **Log_log3** ノードを選択し、**Properties** ビューでそのプロパティを開きます。
12. **Properties** ビューで:
 - **Message** フィールドに **Valid order - ship animals to Germany customer** と入力します。
 - **Id** フィールドに **_ger** と入力します。

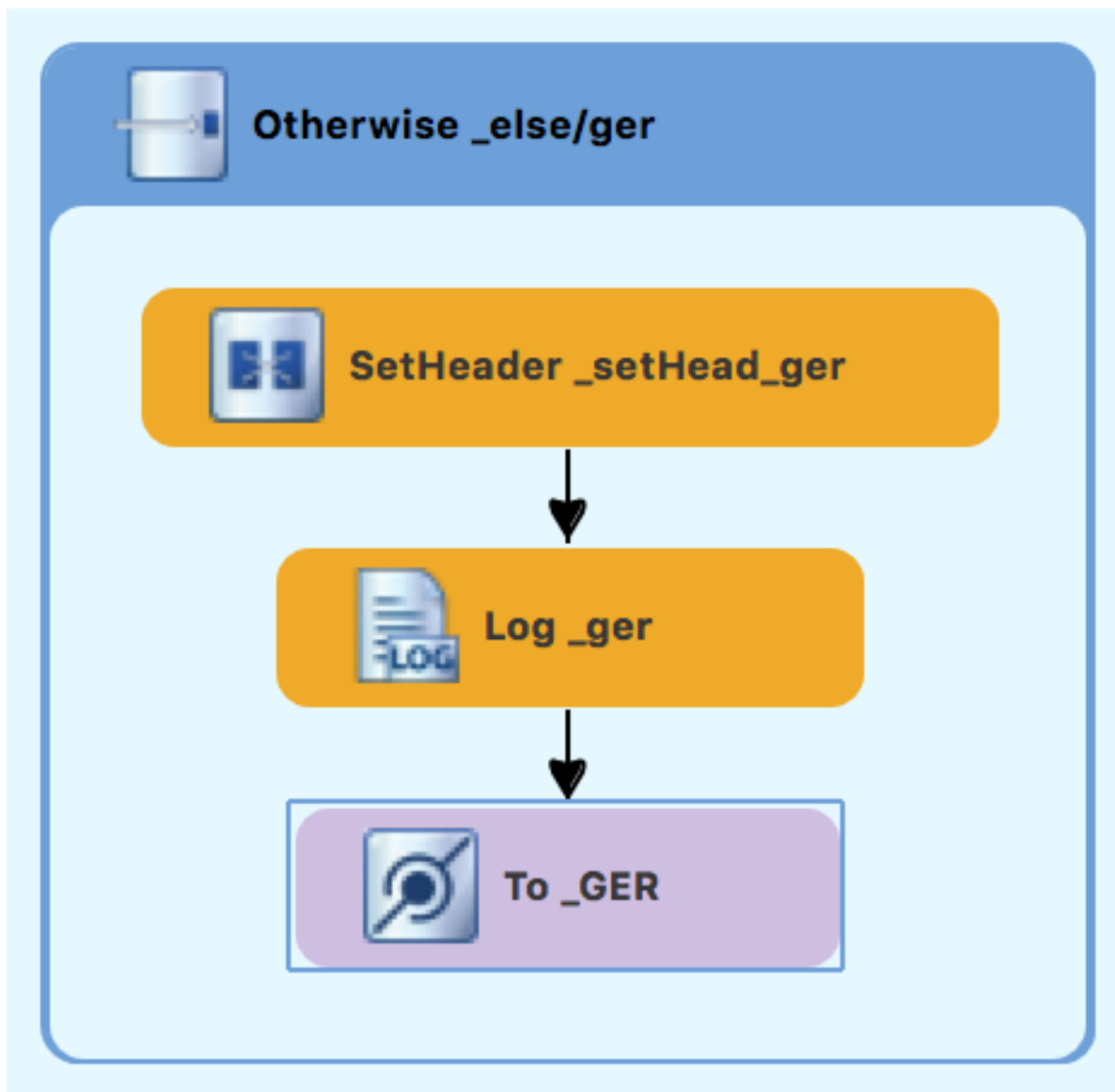
- Logging Level はそのままにしておきます。

13. Components ドロワーで File パターン () を選択し、Log_ger ノードの下をクリックします。
- Otherwise_else/ger コンテナが拡張し、To_to1 ノードが追加されます。必要に応じて、コネクタエラーを SetHeader_setHead_ger ノードから To_to1 ノードにドラッグします。



14. キャンバスで To_to1 ノードを選択し、Properties ビューでそのプロパティーを開きます。
15. Properties ビューで:
- Uri フィールドで、directoryName を target/messages/validOrders/Germany に置き換えます。
 - Id フィールドに _GER と入力します。
16. ファイルを保存します。

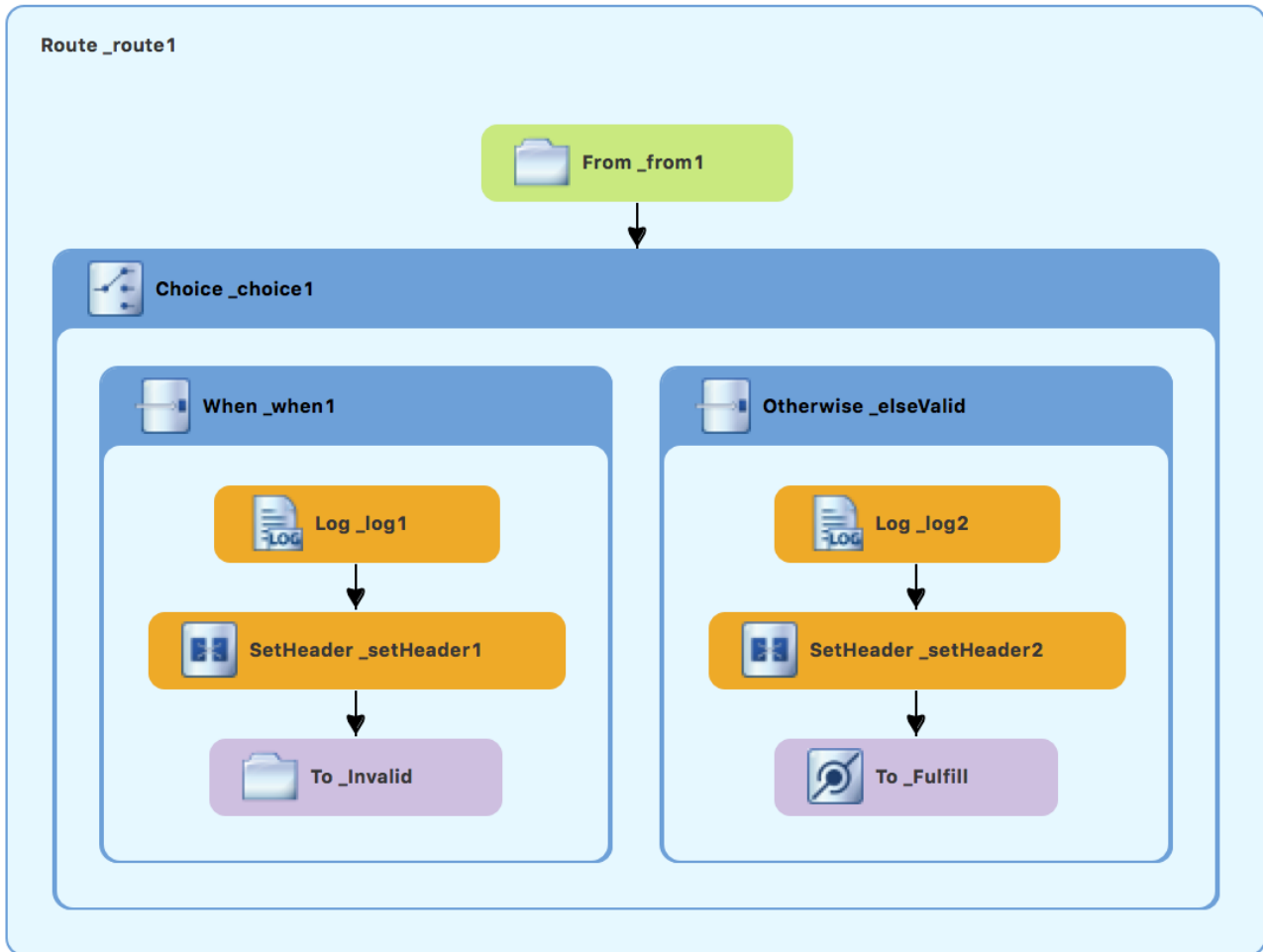
Route_route2 のドイツブランチは以下のようになります。



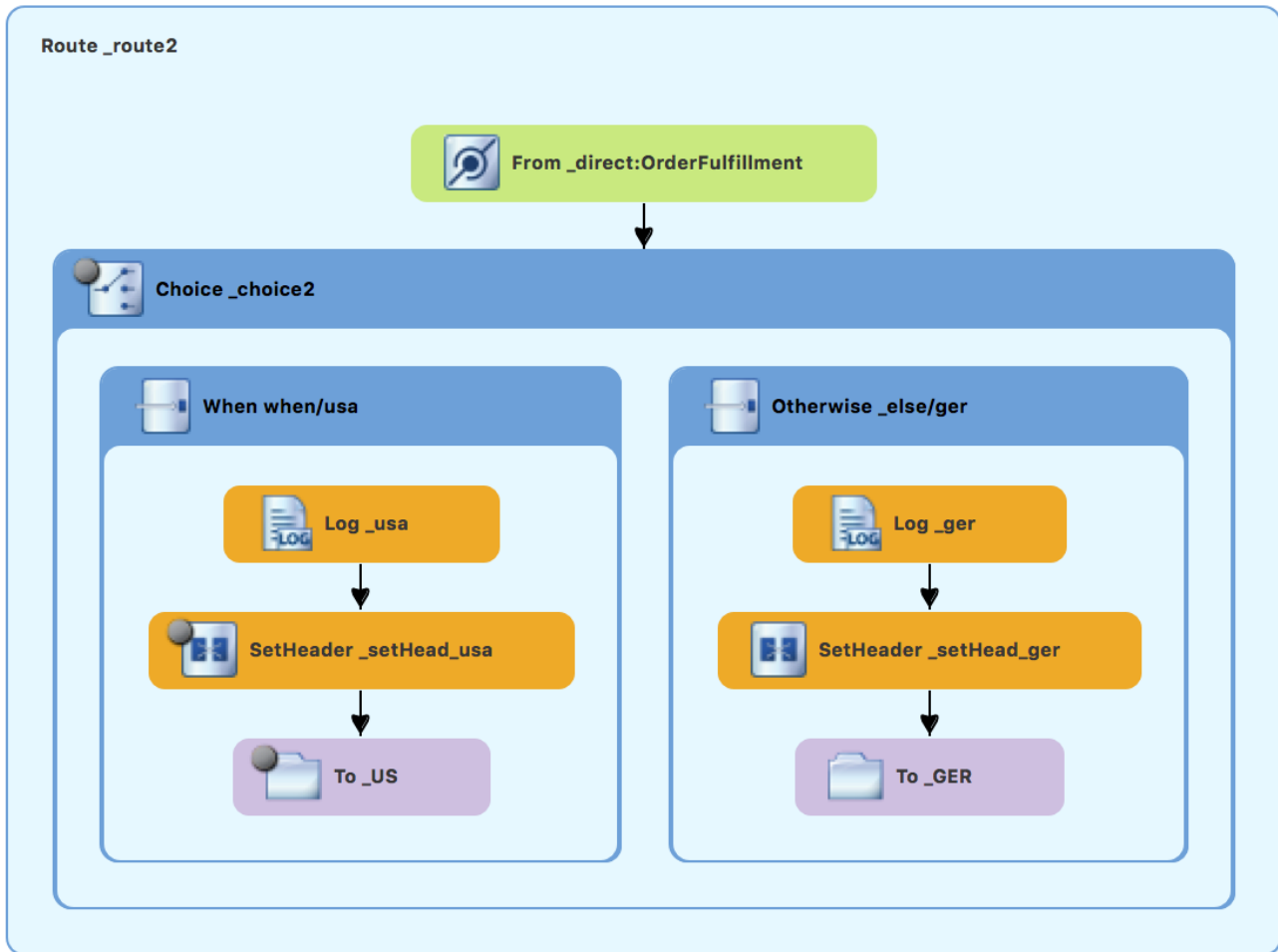
2 番目のルートを確認する

キャンバス上のルートは次のようになります。

完成したルート 1



完成したルート 2



キャンバスの下部にある **Source** タブで、camelContext 要素の XML は次のようになります。例 6.1「デュアルルートコンテンツベースルーターの XML」:

例6.1デュアルルートコンテンツベースルーターの XML

```

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

<blueprint xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.osgi.org/xmlns/blueprint/v1.0.0
    https://www.osgi.org/xmlns/blueprint/v1.0.0/blueprint.xsd
    http://camel.apache.org/schema/blueprint
    http://camel.apache.org/schema/blueprint/camel-blueprint.xsd">

  <camelContext id="_context1" xmlns="http://camel.apache.org/schema/blueprint">
    <route id="_route1" shutdownRoute="Default">
      <from id="_from1" uri="file:src/data?noop=true"/>
      <choice id="_choice1">
        <when id="_when1">
          <xpath>/order/orderline/quantity/text() > 10</xpath>
          <log id="_log1" message="The quantity requested exceeds the maximum allowed -
            contact customer."/>
          <setHeader headerName="Destination" id="_setHeader1">
            <constant>Invalid</constant>
          </setHeader>
          <to id="_Invalid" uri="file:target/messages/invalidOrders"/>
        </when>
      </choice>
    </route>
  </camelContext>

```

```

<otherwise id="_elseValid">
  <log id="_log2" message="This is a valid order - OK to process."/>
  <setHeader headerName="Destination" id="_setHeader2">
    <constant>ReadyForDispatcher</constant>
  </setHeader>
  <to id="_Fulfill" uri="direct:OrderFulfillment"/>
</otherwise>
</choice>
</route>
<route id="_route2">
  <from id="_direct:OrderFulfillment" uri="direct:OrderFulfillment"/>
  <choice id="_choice2">
    <when id="when/usa">
      <xpath>/order/customer/country = 'USA'</xpath>
      <log id="_usa" message="Valid order - ship animals to USA customer"/>
      <setHeader headerName="Destination" id="_setHead_usa">
        <constant>USA</constant>
      </setHeader>
      <to id="_US" uri="file:target/messages/validOrders/USA"/>
    </when>
    <otherwise id="_else/ger">
      <log id="_ger" message="Valid order - ship animals to Germany customer"/>
      <setHeader headerName="Destination" id="_setHead_ger">
        <constant>Germany</constant>
      </setHeader>
      <to id="_GER" uri="file:target/messages/validOrders/Germany"/>
    </otherwise>
  </choice>
</route>
</camelContext>
</blueprint>

```

重要

ツールが `shutdownRoute=""` 属性を 2 番目のルート要素 (`<route id="route2">`) に追加した場合は、その属性を削除します。そうしないと、**ZooOrderApp** プロジェクトの実行に失敗する可能性があります。

更新されたプロジェクトが期待どおりに機能することを確認するには、次の手順に従います。

1. **ZooOrderApp/Camel Contexts/blueprint.xml** をローカルの Camel コンテキスト (テストなし) として実行します。
2. コンソールの出力の終わりを確認します。次の行が表示されます。

```

[ Blueprint Event Dispatcher: 1 ] BlueprintCamelContext      INFO Route: _route1 started and consuming from: file://src/data?noop=true
[ Blueprint Event Dispatcher: 1 ] BlueprintCamelContext      INFO Route: _route2 started and consuming from: direct://OrderFulfillment
[ Blueprint Event Dispatcher: 1 ] BlueprintCamelContext      INFO Total 2 routes, of which 2 are started
[ Blueprint Event Dispatcher: 1 ] BlueprintCamelContext      INFO Apache Camel 2.21.0.fuse-000112-redhat-3 (CamelContext: _context1) started in 0.318
[2, redhat.com:1099/jmxrmi/camel] DefaultManagementAgent      INFO JMX Connector thread started and listening at: service:jmx:rmi:///jndi/rmi://ovpn-1
(1) thread #4 - file://src/data] _route1                      INFO This is a valid order - OK to process.
(1) thread #4 - file://src/data] _route2                      INFO Valid order - ship animals to USA customer
(1) thread #4 - file://src/data] _route1                      INFO This is a valid order - OK to process.
(1) thread #4 - file://src/data] _route2                      INFO Valid order - ship animals to Germany customer
(1) thread #4 - file://src/data] _route1                      INFO This is a valid order - OK to process.
(1) thread #4 - file://src/data] _route2                      INFO Valid order - ship animals to USA customer
(1) thread #4 - file://src/data] _route1                      INFO The quantity requested exceeds the maximum allowed - contact customer.
(1) thread #4 - file://src/data] _route1                      INFO This is a valid order - OK to process.
(1) thread #4 - file://src/data] _route2                      INFO Valid order - ship animals to USA customer
(1) thread #4 - file://src/data] _route1                      INFO The quantity requested exceeds the maximum allowed - contact customer.

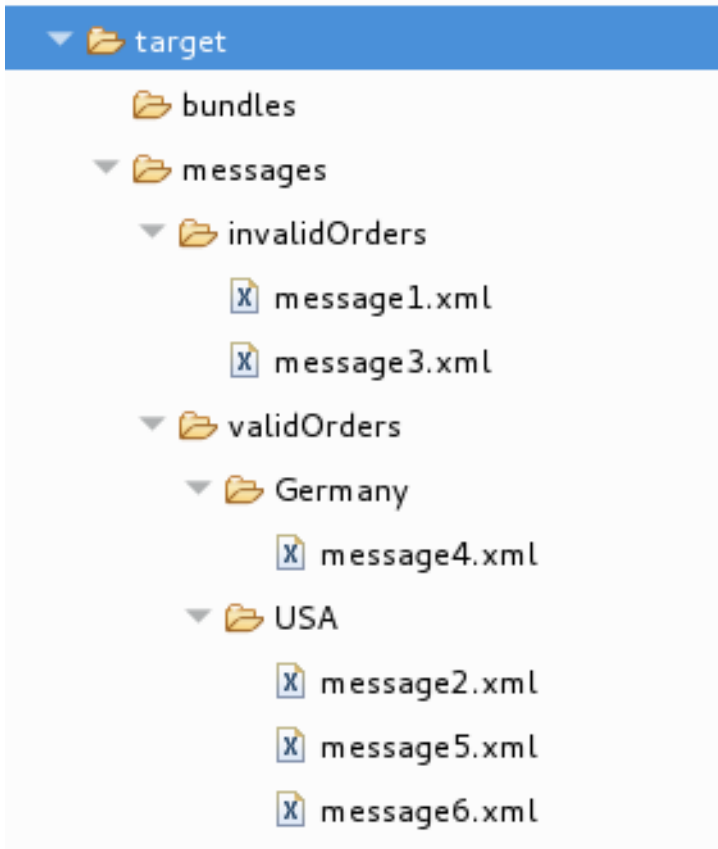
```

3. ターゲットの宛先フォルダーを確認して、ルートが正しく実行されたことを確認します。

Project Explorer を ZooOrderApp を右クリックし、Default を選択します。

- a. Project Explorer で **zooOrderApp** を右クリックし、**Refresh** を選択します。
- b. **target/messages/** フォルダをデプロイメントします。
message*.xml ファイルが以下のように宛先フォルダ内で分散されるはずですが。

図6.1 プロジェクトエクスプローラーでのターゲットメッセージの宛先



次のステップ

次のチュートリアル [7章ルーティングコンテキストのデバッグ](#)では、Fuse Tooling デバッガーの使用方を学習します。

第7章 ルーティングコンテキストのデバッグ

このチュートリアルでは、Camel デバッガーを使用して、ローカルで実行されているルーティングコンテキストの論理エラーを見つける方法を示します。

ゴール

このチュートリアルでは、次のタスクを完了します。

- 2つの経路の対象ノードにブレークポイントを設定します
- デバッグパースペクティブで、経路をステップスルーし、メッセージ変数の値を調べます
- メッセージ変数の値を変更し、効果を観察しながら、経路をもう一度ステップスルーします

前提条件

このチュートリアルを開始するには、次のいずれかの結果である ZooOrderApp プロジェクトが必要です。

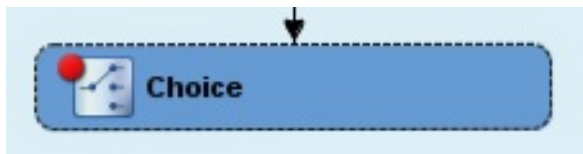
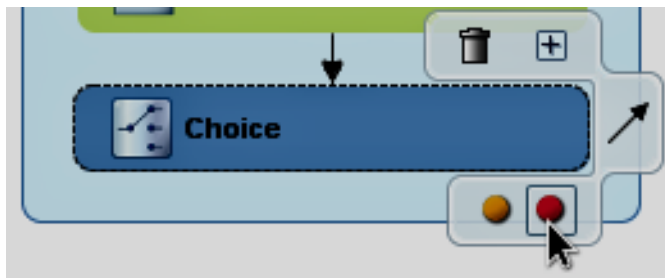
- [6章ルーティングコンテキストに別のルートを追加するチュートリアル](#)を完了します。
または
- [2章環境の設定チュートリアル](#)を完了し、「[リソースファイルについて](#)」に記載されているように、プロジェクトの `blueprint.xml` ファイルを、提供される `blueprintContexts/blueprint3.xml` ファイルに置き換える。

ブレークポイントの設定



デバッガーでは、条件付きブレークポイントと無条件ブレークポイントの両方を設定できます。このチュートリアルでは、無条件のブレークポイントのみを設定します。条件付きブレークポイント(デバッグセッション中に特定の条件が満たされたときにトリガーされるブレークポイント)を設定する方法は、[Tooling ユーザーガイド](#)を参照してください。

無条件のブレークポイントを設定するには:

1. 必要に応じて、ルートエディターで `ZooOrderApp/src/main/resources/OSGI-INF/blueprint/blueprint.xml` を開きます。
2. Project Explorer で **Camel Contexts** → `src/main/resources/OSGI-INF/blueprint/blueprint.xml` をデプロイメントして両方のルートエントリーを公開します。
3. `Route_route1` エントリーをダブルクリックして、Design タブで **Route_route1** にフォーカスを切り替えます。
4. キャンバスで **Choice_choice1** ノードを選択し、その  アイコンをクリックし、無条件のブレークポイントを設定します。





注記

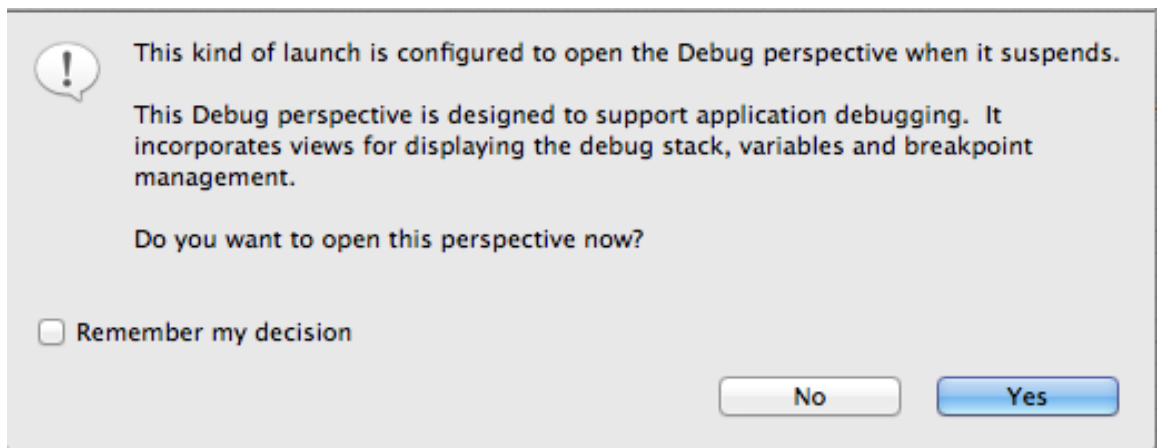
ルートエディターで、ノードの  アイコンまたは  アイコンをそれぞれクリックすることにより、特定のブレイクポイントを無効または削除できます。キャンバスを右クリックし、**Delete all breakpoints** を選択すると、設定したすべてのブレイクポイントを削除することができます。

5. 以下の **Route_Route1** ノードに無条件ブレイクポイントを設定します。
 - **Log_log1**
 - **SetHeader_setHeader1**
 - **To_Invalid**
 - **Log_log2**
 - **SetHeader_setHeader2**
 - **To_Fulfill**
6. Project Explorer で、**src/main/resources/OSGI-INF/blueprint** の下にある **Route_route2** をダブルクリックして、キャンバスで **Route_route2** を開きます。
7. 以下の **Route_Route2** ノードに無条件ブレイクポイントを設定します。
 - **Choice_choice2**
 - **SetHeader_setHead_usa**
 - **Log_usa**
 - **To_US**
 - **SetHeader_setHead_ger**
 - **Log_ger**
 - **To_GER**

ルーティングコンテキストのステップスルー

次の2つの方法でルーティングコンテキストをステップスルーできます。

- ステップオーバー () - ブレークポイントに関係なく、ルーティングコンテキストで実行の次のノードにジャンプします。
- 再開 () - ルーティングコンテキスト内の次のアクティブなブレークポイントにジャンプします。
 1. Project Explorer で ZooOrderApp プロジェクトの Camel Contexts フォルダをデプロイメントし、 **blueprint.xml** ファイルを公開します。
 2. **blueprint.xml** ファイルを右クリックしてコンテキストメニューを開き、 **Debug As → Local Camel Context (without tests)** をクリックします。
Camel デバッガーは、最初に検出したブレークポイントで実行を一時停止し、今すぐ **Debug** パースペクティブを開くかどうかを尋ねます。



3. **Yes** をクリックします。



注記

No をクリックすると、確認ペインがさらに数回表示されます。3回目の拒否後、それは消え、Camel デバッガーは実行を再開します。この時点でデバッガーを操作するには、**Window → Open Perspective → Debug** をクリックして、**Debug** パースペクティブを開く必要があります。

Debug ビューに示されるように、ルーティングコンテキストが **_choice1 in _route1 [blueprint.xml]** で一時停止された状態で **Debug** パースペクティブが開きます。

The screenshot shows the Red Hat Fuse IDE interface. The main window displays a Camel blueprint with two routes, Route1 and Route2. Route1 contains a choice node with two branches: 'When_when1' and 'Otherwise_else2'. The 'When_when1' branch has a log node, a setHeader node, and a 'To_invalid' endpoint. The 'Otherwise_else2' branch has a log node, a setHeader node, and a 'To_Fulfill' endpoint. The 'When_when1' branch is highlighted with a red box. The 'Variables' view on the right shows the current state of the Camel context, including the CamelDebugger, Processor, Exchange, and Message. The console at the bottom shows the execution log, including the message 'NodeBreakpoint at node _choice1 is waiting to continue for exchangeId: ID-janemurpheysmbp-home-55846-1471374645179-0-1'.




注記

ブレークポイントは、デバッガーが自動的に再開する前に最大 5 分間保持され、次のブレークポイントまたはルーティングコンテキストの最後のいずれかに移動します。


4. **Variables** ビューで、ノードをデプロイメントして、各ノードで使用可能な変数と値を表示します。

ルーティングコンテキストをステップ実行すると、最後のブレークポイント以降に値が変更された変数が黄色で強調表示されます。変更された変数を表示するには、各ブレークポイントでノードをデプロイメントする必要がある場合があります。

5.  をクリックし、次のブレークポイント **_log2 in _route1 [blueprint.xml]** に移動します。

The screenshot shows the Red Hat Fuse IDE interface. The 'Variables' view on the right is expanded to show the 'Message' object. The 'Message' object is highlighted in yellow, and its 'MessageBody' is displayed as '<?xml version="1.0" encoding="UTF-8"?><n\n<order><n <customer><n'. The console at the bottom shows the execution log, including the message 'NodeBreakpoint at node _choice1 is waiting to continue for exchangeId: ID-janemurpheysmbp-home-55846-1471374645179-0-1'.

6. **Variables** ビューでノードをデプロイメントし、**_choice1 in Route1 [blueprint.xml]** の最後のブレークポイントから変更された変数を確認します。

7.  をクリックし、次のブレークポイント **_setHeader2 in Route1 [blueprint.xml]** に移動します。
_log2 in Route1 [blueprint.xml] のブレークポイントから変更された変数 (黄色で強調表示) を検証します。

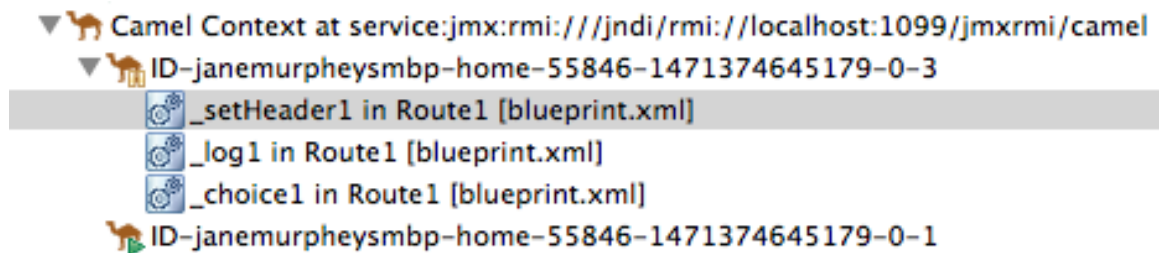
8. **Debug** ビューで `_log2 in _route1 [blueprint.xml]` をクリックし、**Variables** ビューにブレークポイント `_log2 in _route1 [blueprint.xml]` からの変数の値を投入し素早く比較します。
- Debug** ビューでは、同じメッセージフロー内のブレークポイントを切り替えて、**Variables** ビューで変化する変数値をすばやく比較および監視できます。



注記

メッセージフローの長さはさまざまです。**Route_route1** の **InvalidOrders** ブランチを移動するメッセージの場合、メッセージフローは短くなります。**Route_route2** に移行する **Route_route1** の **ValidOrders** ブランチを移動するメッセージでは、メッセージフローが長くなります。

9. ルーティングコンテキストのステップを続行します。1つのメッセージがルーティングコンテキストを完了し、次のメッセージがそのコンテキストに入ると、新しいメッセージフローが **Debug** ビューに表示され、新しいブレッドクラム ID でタグ付けされます。





























この場合、**ID-janemurpheysmbp-home-55846-1471374645179-0-3** は、ルーティングコンテキストに入った `message2.xml` に対応する 2 番目のメッセージフローを識別します。ブレッドクラム ID は 2 ずつ増加します。



注記

Exchange ID とメッセージ ID は同一であり、メッセージがルーティングコンテキストを通過する間に変更されません。それらの ID は、メッセージフローのブレッドクラム ID から作成され、1 ずつ増加します。そのため、`message2.xml` の場合、その **Exchangeld** および **MessageId** は **ID-janemurpheysmbp-home-55846-1471374645179-0-4** になります。

10. `Message3.xml` がブレークポイント `_choice1 in _route_route1 [blueprint.xml]` に入る際に、**Processor** 変数を確認します。表示される値は、これまでルーティングコンテキストを移動した `message1.xml` および `message2.xml` の累積メトリックです。


(x)= Variables  Breakpoints  Expressions 		
Name	Value	
▼  CamelDebugger	CamelDebuggerSettings (id=-914092373)	
 BodyMaxChars	131072	
 BodyIncludeFiles	true	
 BodyIncludeStreams	false	
 DebugCounter	13	
 LogLevel	INFO	
 Endpoint	_choice1	
▼  Processor	CamelProcessor (id=-1185022607)	
 ProcessorId	_choice1	
 RouteId	Route1	
 CamelId	_context1	
 CompletedExchanges	2	
 FailedExchanges	0	
 TotalExchanges	2	
 Redeliveries	0	
 ExternalRedeliveries	0	
 HandledFailures	0	
 LastProcessingTime	84876	
 MinProcessingTime	84876	
 AverageProcessingTime	122602	
 MaxProcessingTime	160329	
 TotalProcessingTime	245205	
▼  Exchange	CamelExchange (id=-1771897463)	


タイミングメトリックはミリ秒単位です。

- ルーティングコンテキストを介して各メッセージのステップを続行し、各処理ステップで変数とコンソール出力を調べます。**Message6.xml** がブレークポイント **To_GER in Route2 [blueprint.xml]** に入ると、デバッガーはパンくずスレッドのシャットダウンを開始します。
- メニューバーで  をクリックし、Camel デバッガーを終了します。コンソールは終了しますが、手動で出力をクリアする必要があります。



注記

Debug ビューの Camel Context ノードでスレッドまたはエンドポイントを選択した状態で、 を 2 回 クリックします。1 回目のクリックでスレッドまたはエンドポイントを終了し、2 回目のクリックで Camel Context (つまりセッション) を終了します。

- メニューバーで  **Debug** を右クリックし、コンテキストメニューを開き、**Close** を選択して **Debug** パースペクティブを閉じます。
CodeReady Studio は、Camel デバッガーを起動したパースペクティブに自動的に戻りません。

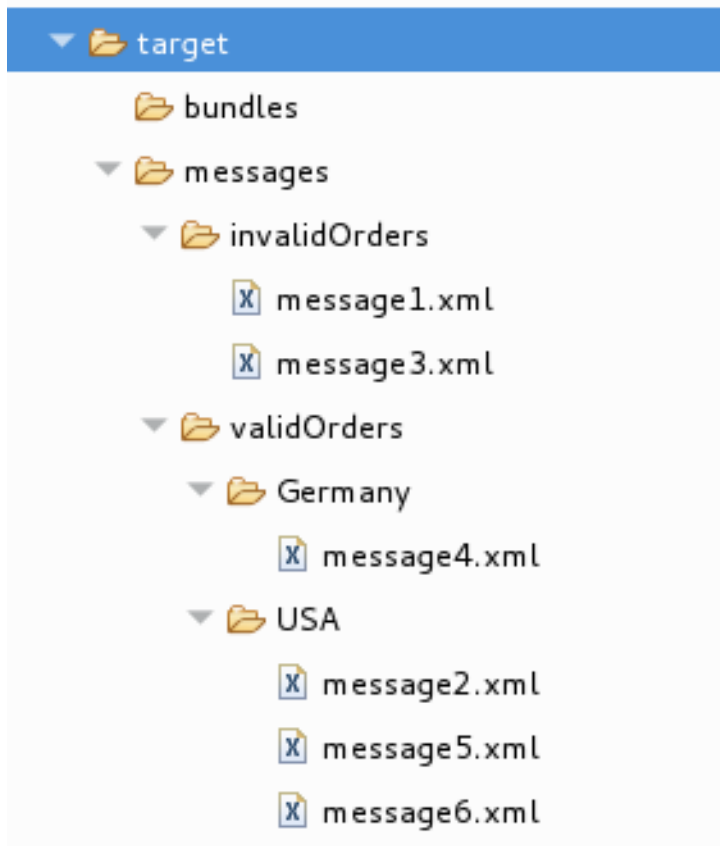
14. **Project Explorer** で、プロジェクトを右クリックし、**Refresh** を選択して表示を更新します。



注記

すべてのメッセージがルーティングコンテキストを移動する前にセッションを途中で終了した場合、**ZooOrderApp/src/data** フォルダーに **message3.xml.camelLock** のようなメッセージが表示されることがあります。プロジェクトでデバッガーを再度実行する前に、これを削除する必要があります。これを行うには、**.camelLock** メッセージをダブルクリックしてコンテキストメニューを開き、**Delete** を選択します。求められたら、**OK** をクリックして削除を確認します。

15. **ZooOrderApp/target/messages/** ディレクトリーをデプロイメントして、メッセージが予想される宛先に配信されていることを確認します。



すべてのブレークポイントを設定して有効にして、ルーティングコンテキストをそのままにします。

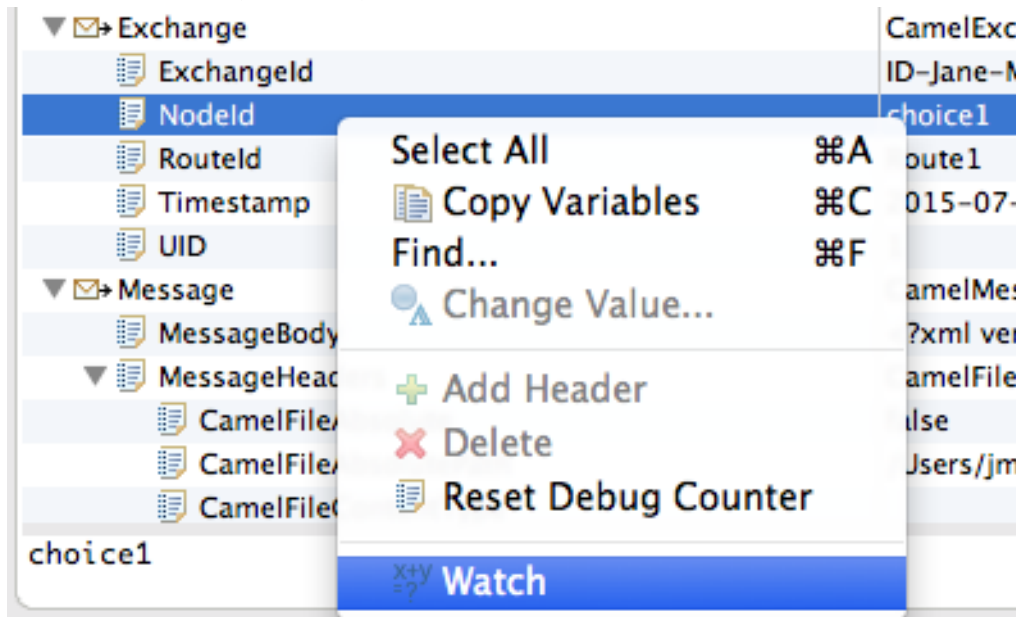
変数の値を変更する

このセクションでは、ウォッチリストに変数を追加して、メッセージがルーティングコンテキストを通過するときに変数の値がどのように変化するかを簡単に確認します。メッセージ本文の変数の値を変更してから、その変更がルーティングコンテキストを介したメッセージのルートにどのように影響するかを観察します。

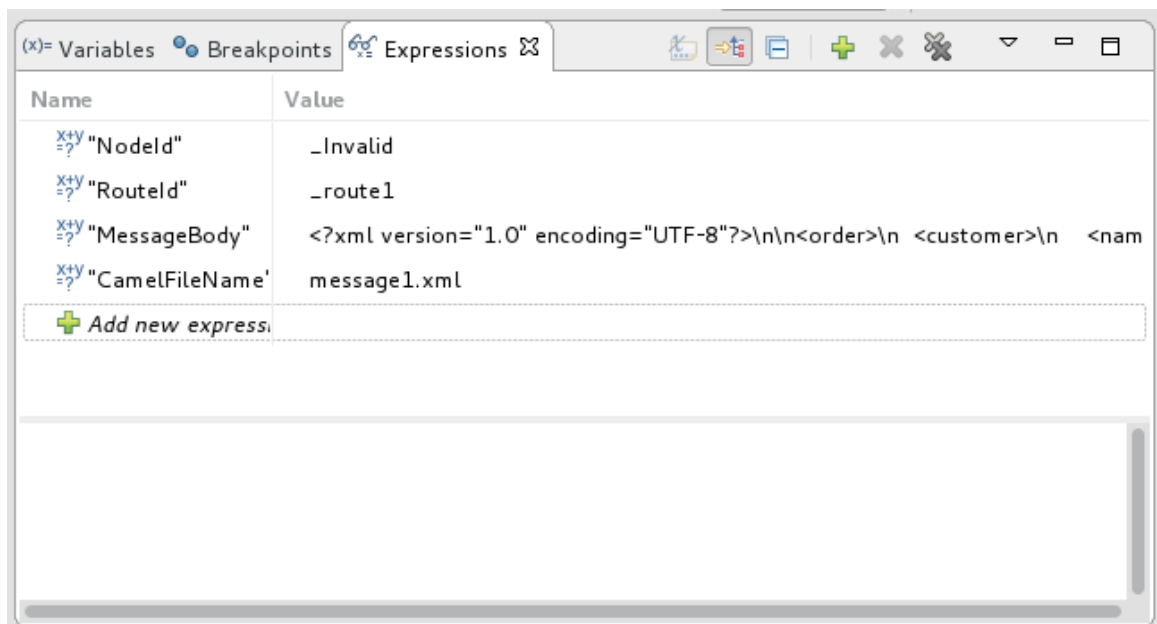
1. **ZooOrderApp** プロジェクトで Camel デバッガーを再度実行するには、**blueprint.xml** ファイルを右クリックして、**Debug As → Local Camel Context (without tests)** をクリックします。
2. **Message1** が最初のブレークポイント **_choice1 in _route1 [blueprint.xml]** で停止した場合、変数 **NodeId** および **RouteId(Exchange カテゴリー)** ならびに **MessageBody** および **CamelFileName(Message カテゴリー)** を監視リストに追加します。

4つの変数のそれぞれについて:

- a. **Variables** ビューで、適切なカテゴリをデプロイメントして、ターゲット変数を公開します。
- b. 変数 (ここでは **Exchange** カテゴリの **Nodeld**) を右クリックしてコンテキストメニューを開き、**Watch** を選択します。



Expressions タブが開き、監視するように選択した変数がリスト表示されます。

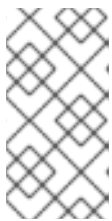
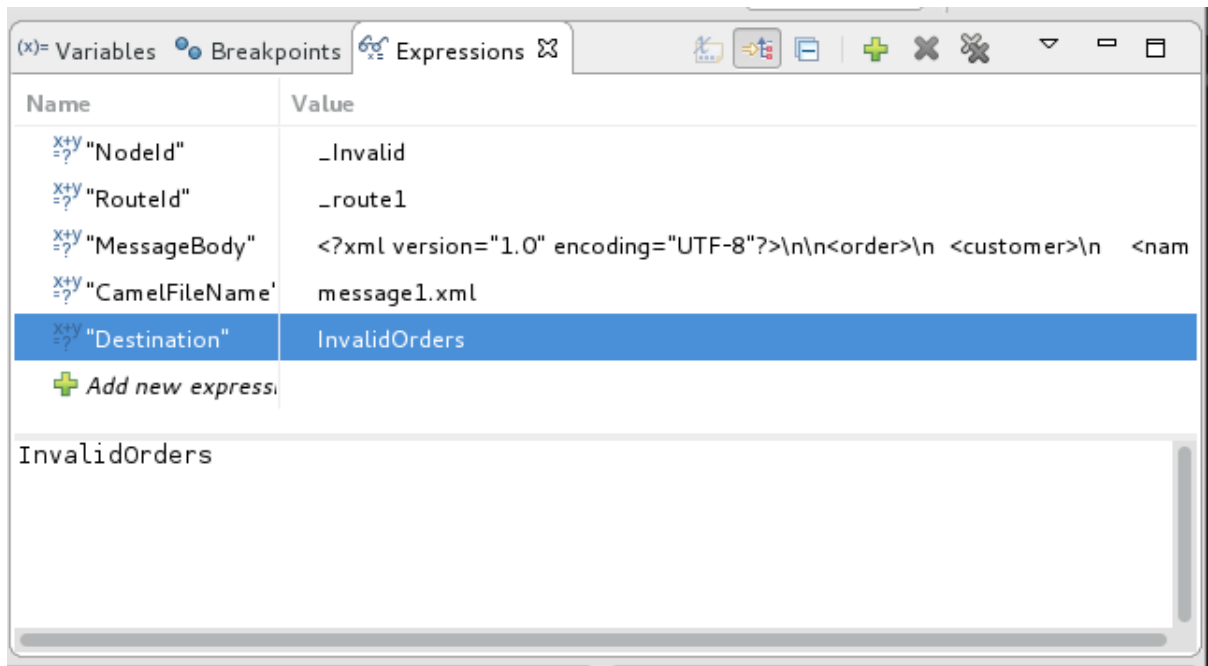


注記

ウォッチリストを作成すると、関心のある複数の変数の現在の値をすばやく簡単に確認できます。

3. 4番目のブレークポイント **_Fulfill in _route1 [blueprint.xml]** に到達するまで、ルーティングコンテキスト全体で **message1** を確認します。
4. **Variables** ビューで、**Message** カテゴリをデプロイメントします。

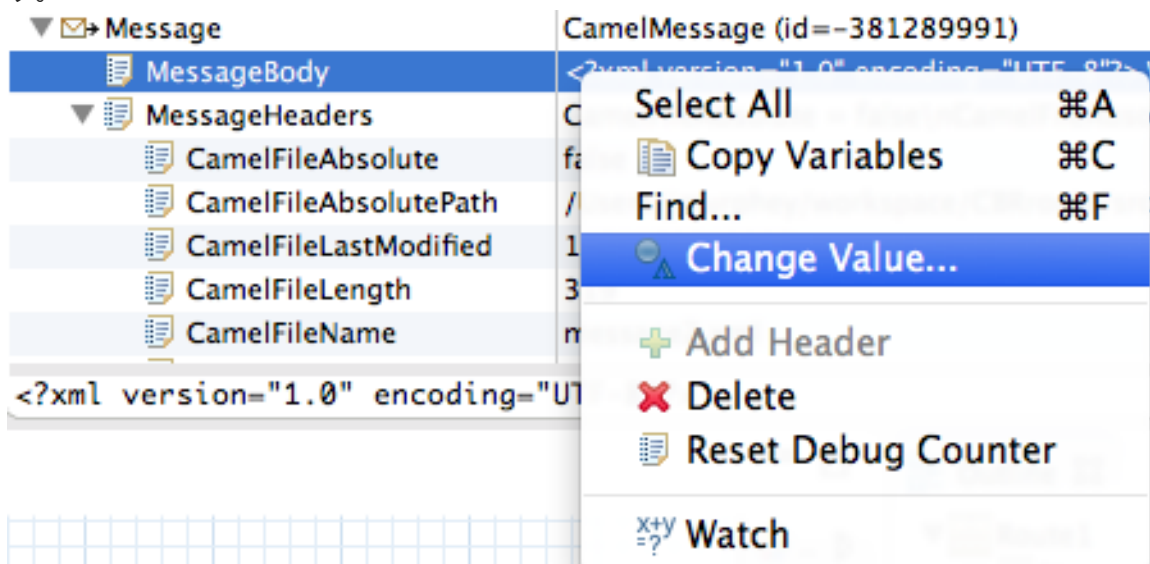
- 変数 **Destination** を監視リストに追加します。
これで、**Expressions** ビューに次の変数が含まれるはずですが。



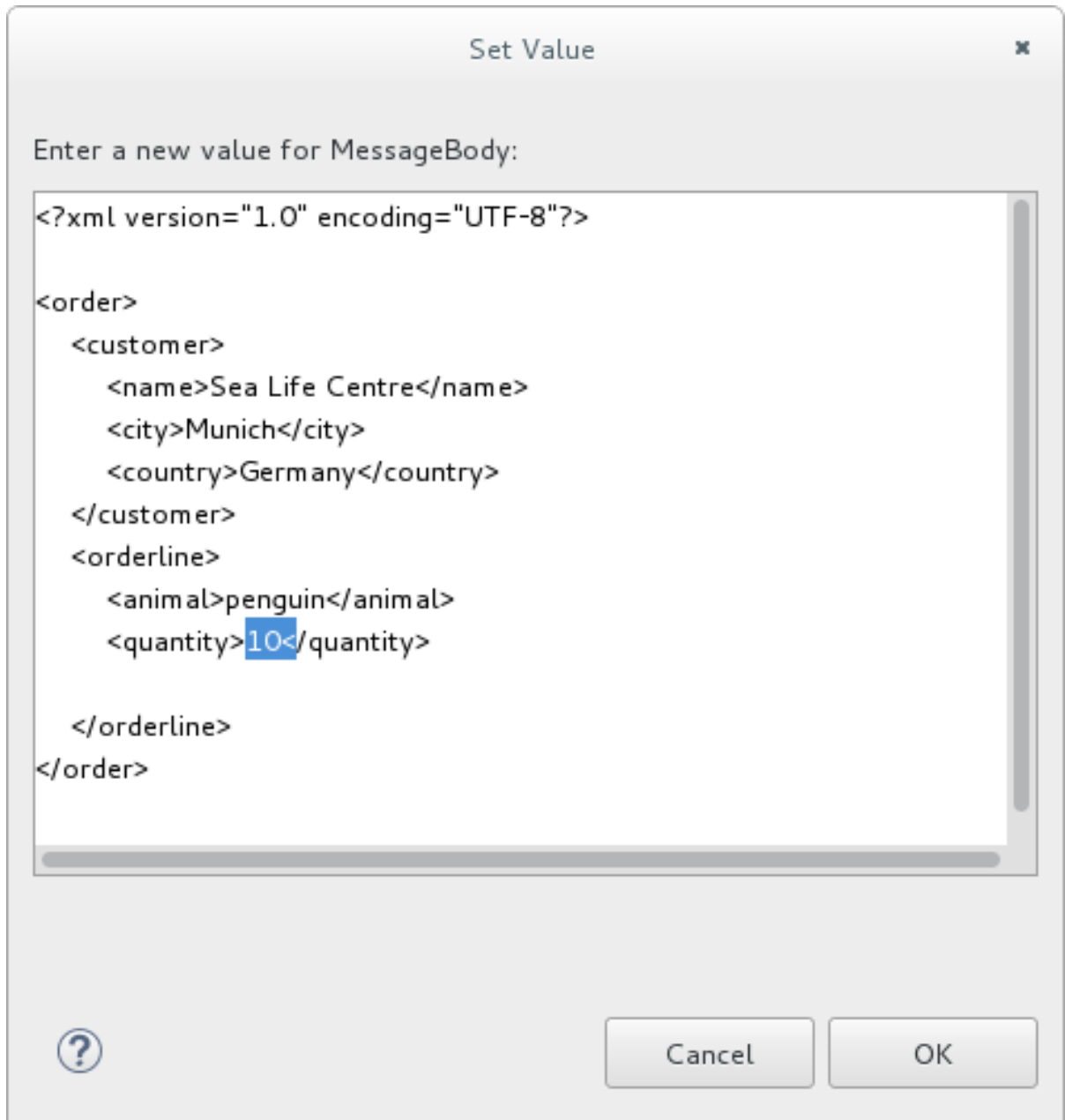
注記

- 変数のリストの下のペインには、選択した変数の値が表示されます。
- Expressions** ビューは、明示的に削除するまで、リストに追加したすべての変数を保持します。

- 残りのルーティングコンテキスト全体で **message1** を確認し、次にルーティングコンテキスト全体で **message2** を確認します。
- _choice1 in _route1 [blueprint.xml]** で **message3** を停止します。
- Variables** ビューで **Message** カテゴリをデプロイメントし、**MessageBody** 変数を表示します。
- MessageBody** を右クリックしてコンテキストメニューを開き、**Change Value** を選択します。



10. (これを無効な注文から有効な注文に変更するため)、**quantity** の値を 15 から 10 に変更します。



この操作により、インメモリーの値のみが変更されます (**message3.xml** ファイルは編集されません)。

11. **OK** をクリックします。
12. **Expressions** ビューに切り替え、**MessageBody** 変数を選択します。
変数リストの下にあるペインには **message3** のボディ全体が表示され、注文アイテムの現在の値を簡単に確認することができます。

Name	Value
X+Y =? "NodId"	_choice1
X+Y =? "RouteId"	_route1
X+Y =? "MessageBody"	<?xml version="1.0" encoding="UTF-8"?>\n\n<order>\n <customer>\n
X+Y =? "CamelFileName"	message3.xml
X+Y =? "Destination"	

```

<?xml version="1.0" encoding="UTF-8"?>
<order>
  <customer>
    <name>Sea Life Centre</name>
    <city>Munich</city>
    <country>Germany</country>
  </customer>
  <orderline>
    <animal>penguin</animal>
    <quantity>10</quantity>
  </orderline>
</order>

```

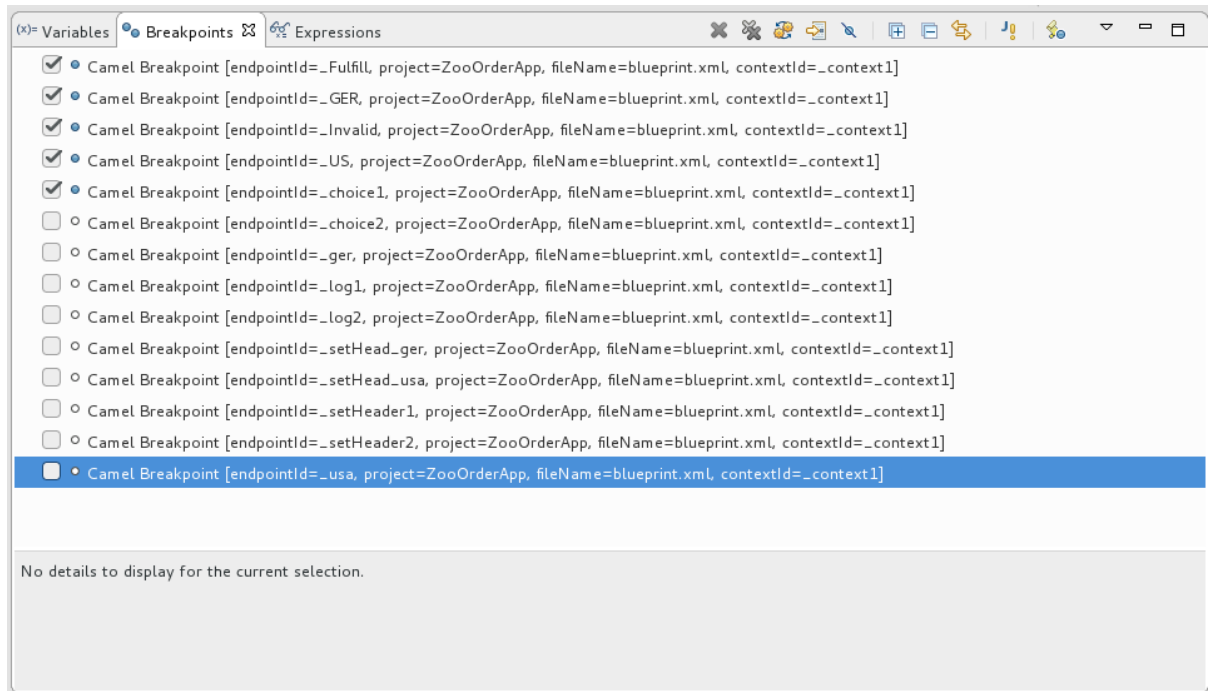
13.  をクリックし、次のブレークポイントに移動します。

To_Invalid へのブランチをたどる代わりに、**message3** は **To_Fulfill** および **Route_route2** へのブランチをたどります。

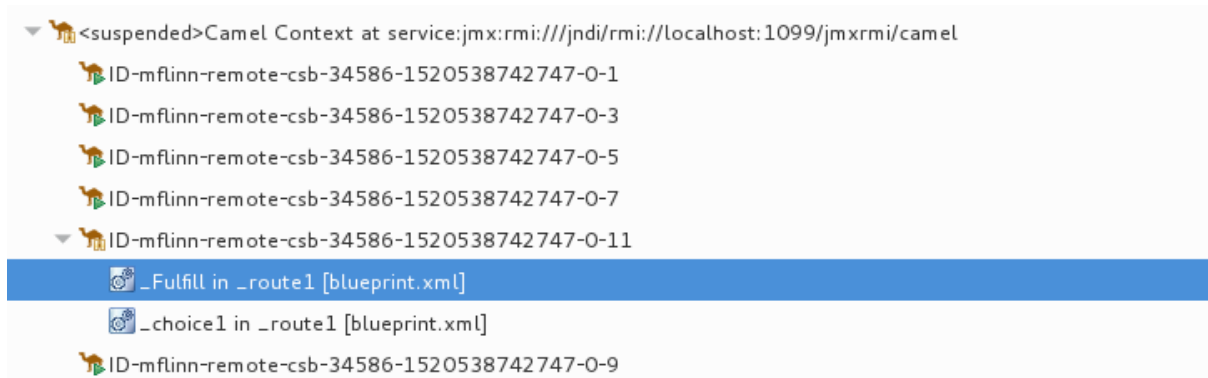
CAMEL デバッガーのフォーカスを絞る

ブレークポイントを無効にしてから再度有効にすることで、デバッガーのフォーカスを一時的に絞り込んでから再拡張できます。

1. ルーティングコンテキスト全体で **message4** を確認し、各ステップで **Debug** ビュー、**Variables** ビュー、および **Console** の出力をチェックします。
2. **_choice1 in _route1 [blueprint.xml]** で **message4** を停止します。
3. **Breakpoints** ビューに切り替え、**_choice1** の下にリスト表示されるブレークポイントの横にある各チェックボックスのチェックを外します。ブレークポイントのチェックボックスをオフにすると、一時的に無効になります。

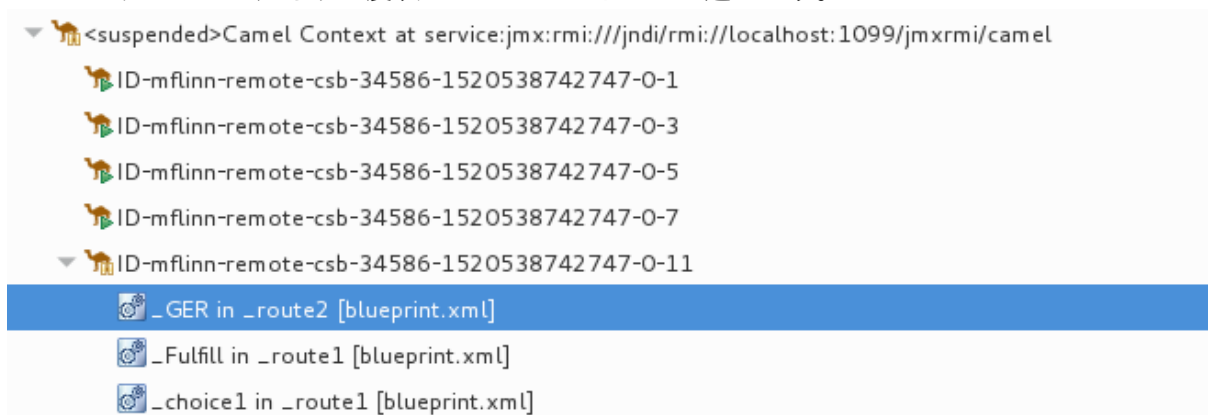


4.  をクリックして、次のブレークポイントに移動します。




デバッガーは、無効なブレークポイントをバイパスし、**_FulFill in _route1 [blueprint.xml]** にジャンプします。

5.  をクリックして、もう一度次のブレークポイントに進みます。



デバッガーは、**_GER in _route2 [blueprint.xml]** にジャンプします。

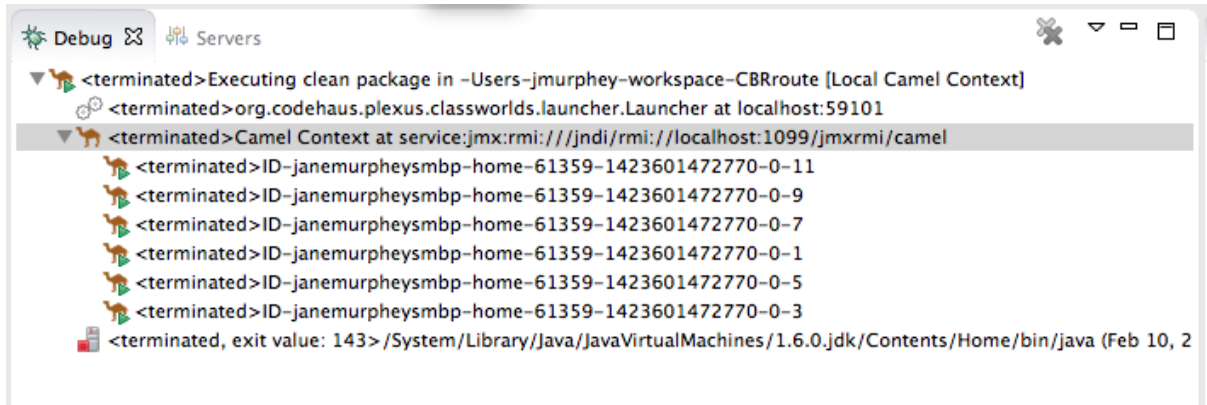
6.  を繰り返しクリックして、ルーティングテキスト全体で **message5** および **message6** 素早く確認します。


7. **Breakpoints** ビューに切り替え、すべてのブレークポイントの横にあるチェックボックスをオンにして、ブレークポイントを再度有効にします。

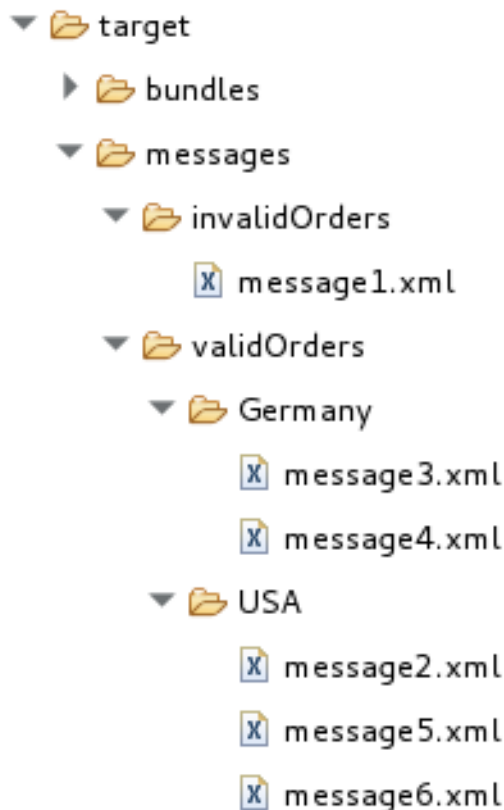
メッセージ変数値を変更した場合の影響を確認する

デバッガーを停止し、message1 の数量変数の値を変更した結果を確認するには:

1. ツールバーで、 をクリックし、Camel デバッガーを終了します。



2. コンソールの  ボタンをクリックして出力をクリアします。
3. **Debug** パースペクティブを閉じて、Camel デバッガーを起動したパースペクティブに戻ります。
4. **Project Explorer** で、表示を更新します。
5. **ZooOrderApp/target/messages/** ディレクトリーをデプロイメントして、メッセージが想定どおりに配信されたかどうかを確認します。



Message1 のみが **invalidOrders** に送信されたこと、および **message3.xml** が **validOrders/Germany** フォルダーに表示されることが確認できるはずです。

次のステップ

8章 [ルートを紹介したメッセージのトレース](#) チュートリアルでは、ルーティングコンテキストを介してメッセージをトレースし、ルーティングコンテキストのパフォーマンスを最適化および微調整できる場所を決定します。

第8章 ルートを介したメッセージのトレース

トレースを使用すると、メッセージが1つのノードから別のノードにルーティングされるときにメッセージをインターセプトできます。ルーティングコンテキストを介してメッセージをトレースし、ルーティングコンテキストのパフォーマンスを最適化および微調整できる場所を確認できます。このチュートリアルでは、ルートを介してメッセージをトレースする方法を示します。

ゴール

このチュートリアルでは、次のタスクを完了します。

- **Fuse Integration** パースペクティブで **ZooOrderApp** を実行する
- **ZooOrderApp** でトレースを有効にする
- **ZooOrderApp** にメッセージをドロップし、すべてのルートノードを通じてメッセージを追跡する


前提条件

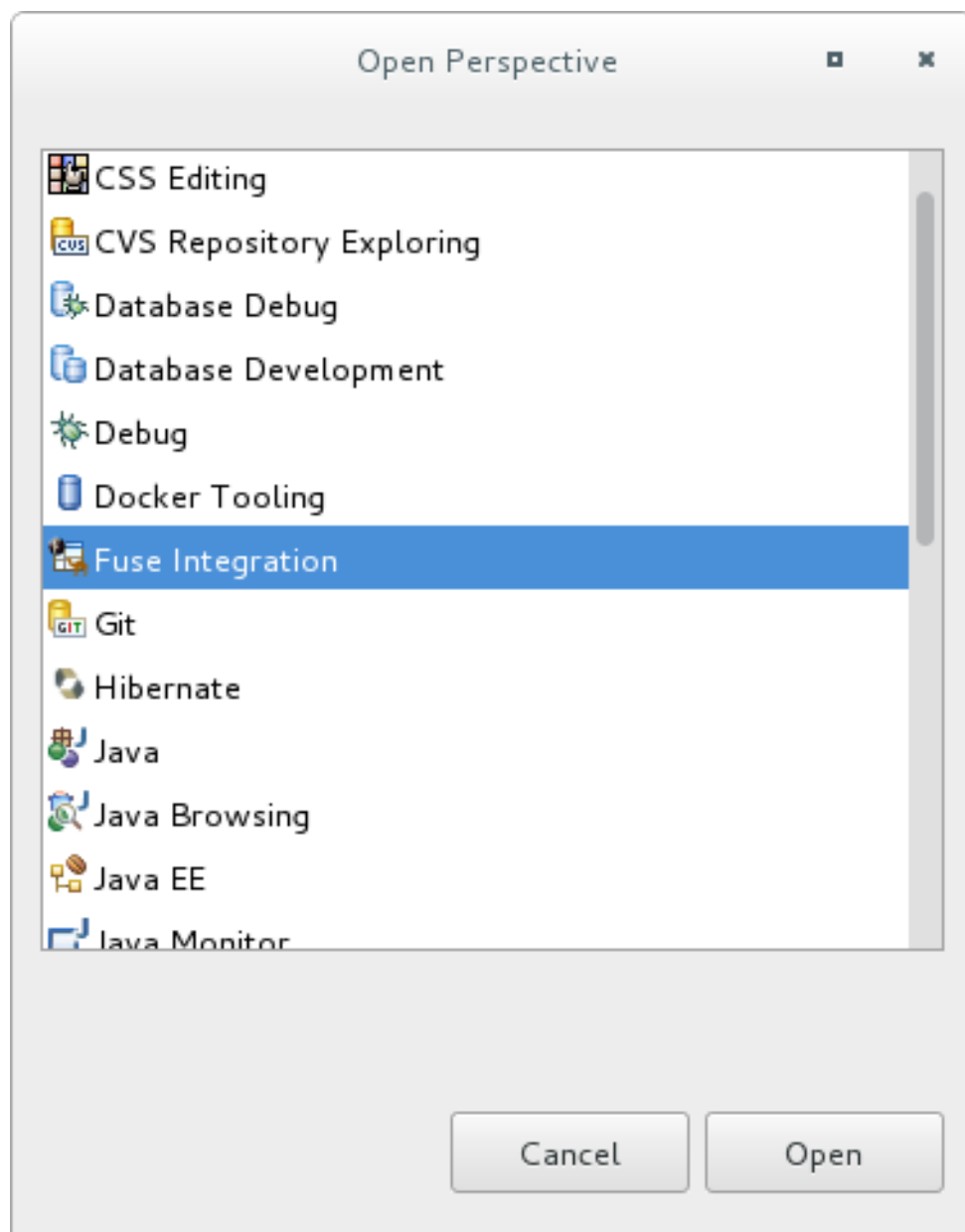
このチュートリアルを開始するには、次のいずれかの結果である **ZooOrderApp** プロジェクトが必要です。

- [6章 ルーティングコンテキストに別のルートを追加するチュートリアル](#)を完了します。
または
- [2章 環境の設定](#)チュートリアルを完了し、「[リソースファイルについて](#)」に記載されているように、プロジェクトの **blueprint.xml** ファイルを、提供される **blueprintContexts/blueprint3.xml** ファイルに置き換える。

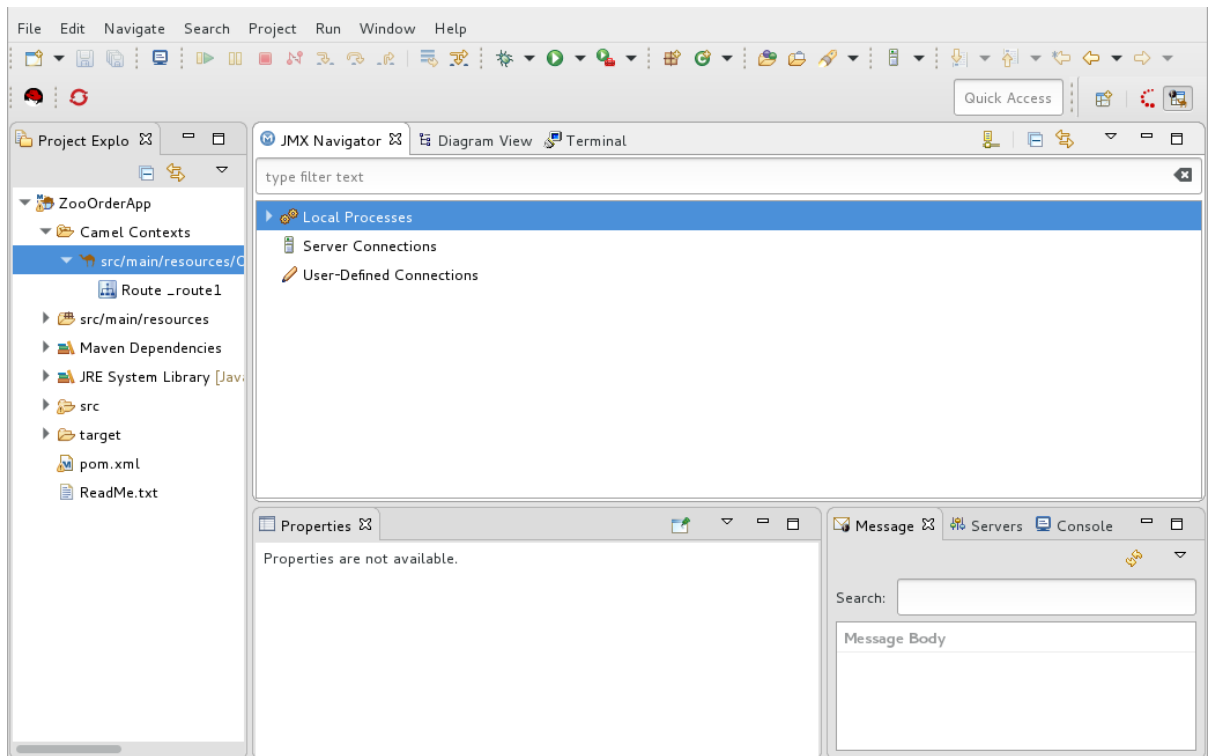
FUSE INTEGRATION パースペクティブの設定

メッセージトレースを容易にするためにワークスペースを設定するには:

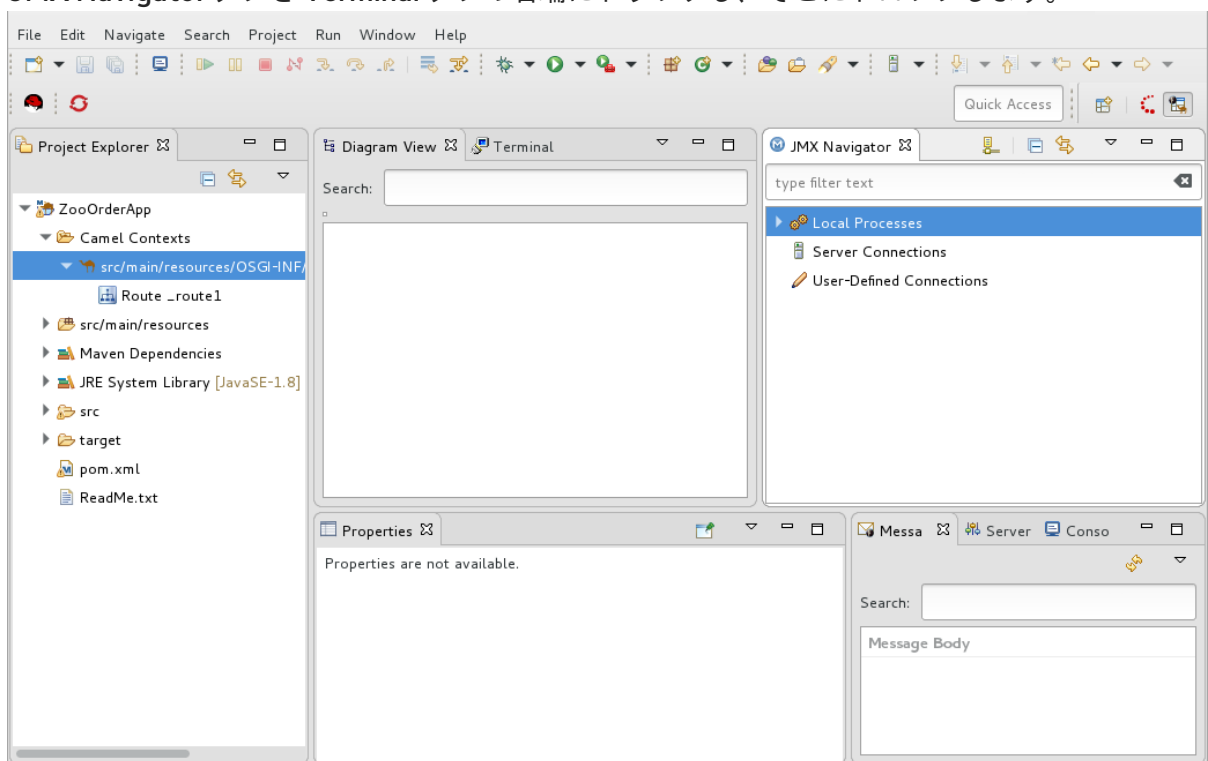
1. ツールバーの右側にある  ボタンをクリックし、リストから **Fuse Integration** を選択します。



Fuse Integration パースペクティブがデフォルトのレイアウトで開きます。



2. JMX Navigator タブを Terminal タブの右端にドラッグし、そこにドロップします。

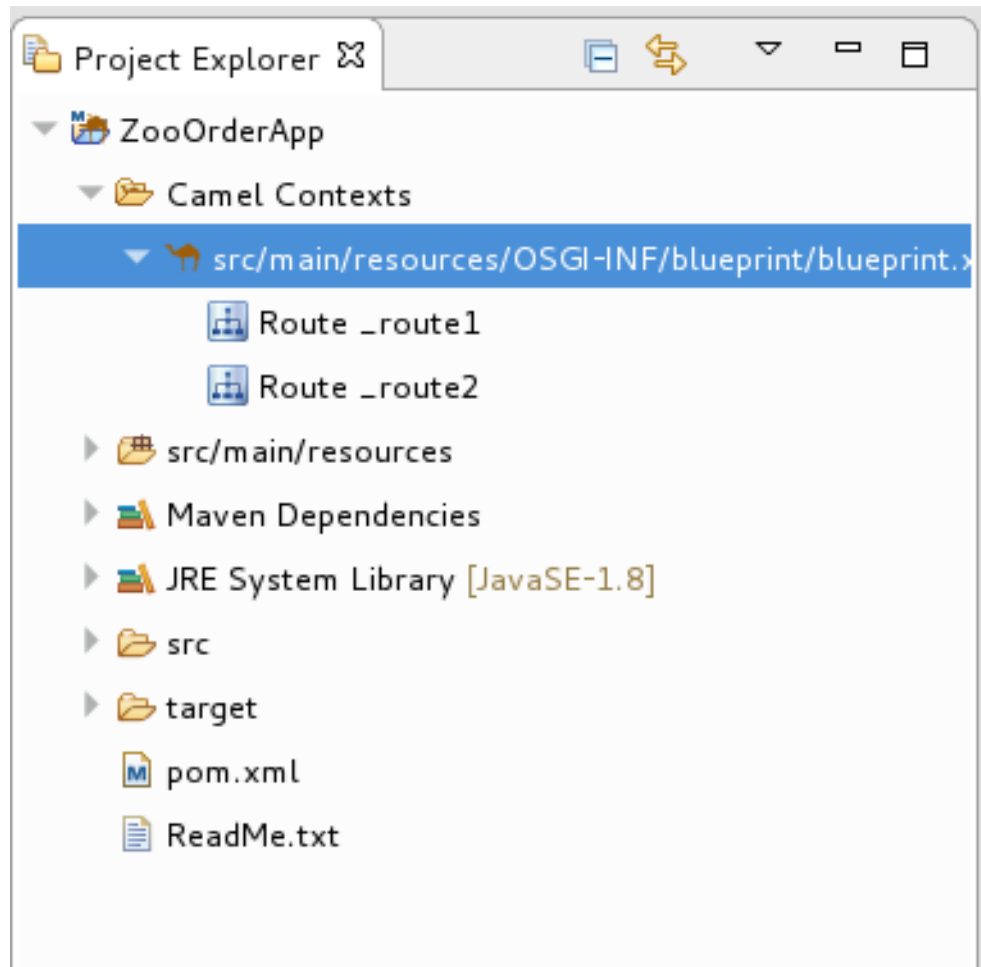


この配置により、Diagram View がルーティングコンテキストのノードをグラフィカルに表示するためのスペースが増え、メッセージがルーティングコンテキストを通過する際にたどるパスを視覚的に追跡しやすくなります。

注記

ルーティングコンテキストの **.xml** ファイルへのアクセスを容易にするために (特にプロジェクトが複数のコンテキストで設定される場合)、このツールは **Project Explorer** の **Camel Contexts** フォルダにコンテキストをリスト表示します。

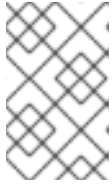
さらに、ルーティングコンテキスト内のすべてのルートは、コンテキストファイルエントリーのすぐ下にアイコンとして表示されます。キャンバスのルーティングコンテキストに単一のルートを表示するには、**Project Explorer** でそのアイコンをダブルクリックします。ルーティングコンテキスト内のすべてのルートを表示するには、コンテキストファイルエントリーをダブルクリックします。



メッセージトレースの開始

ZooOrderApp プロジェクトでメッセージのトレースを開始するには、以下を実行します。

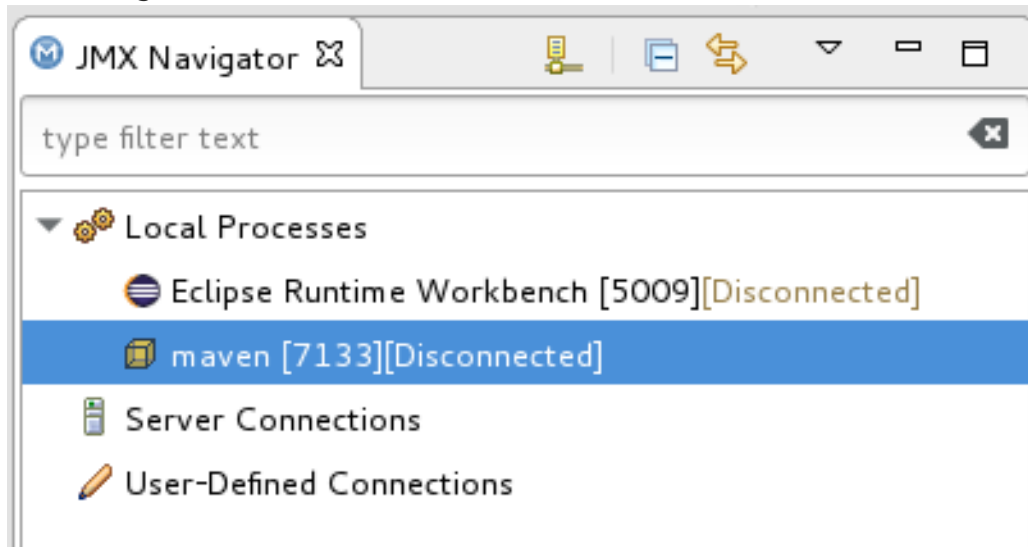
1. **Project Explorer** で **ZooOrderApp** プロジェクトをデプロイメントして **src/main/resources/OSGI-INF/blueprint/blueprint.xml** を公開します。
2. **src/main/resources/OSGI-INF/blueprint/blueprint.xml** を右クリックし、コンテキストメニューを開きます。
3. **Run As** → **Local Camel Context (without tests)** を選択します。



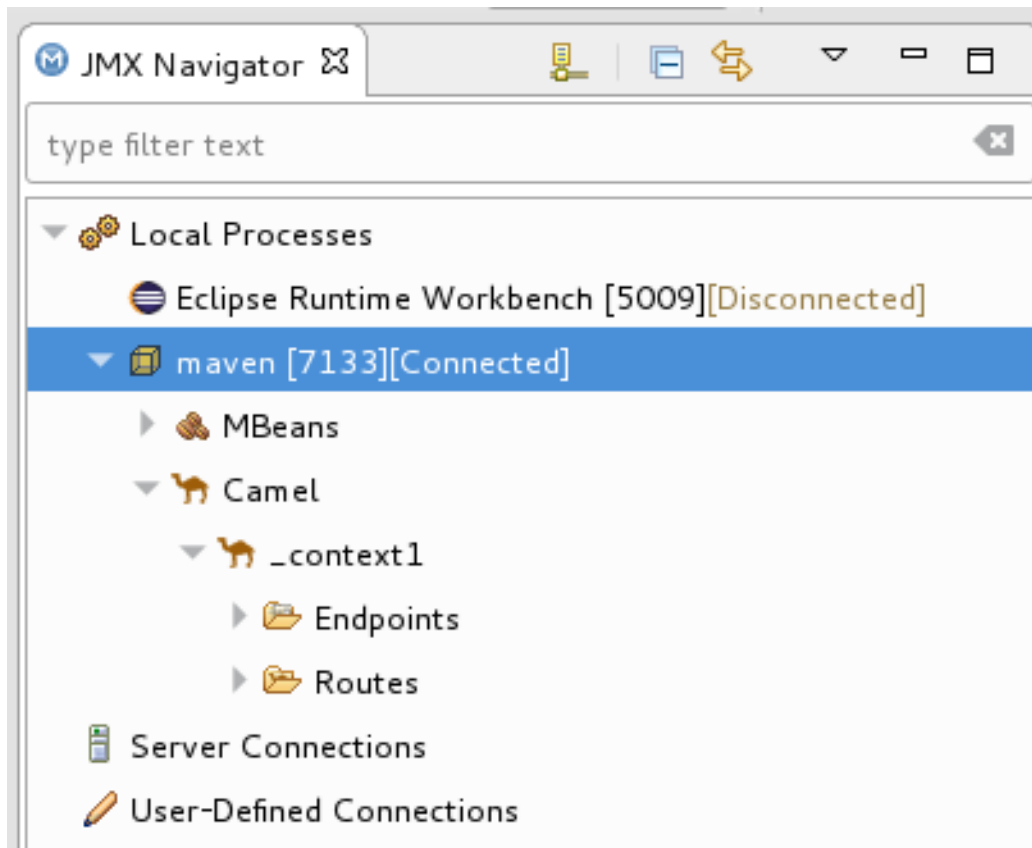
注記

Local Camel Context を選択すると、**ZooOrderApp** プロジェクトの JUnit テストがまだ作成されていないため、ツールはテストのない実行に戻します。9章 [JUnit を使用したルートのテスト](#) で、後でそれを行います。

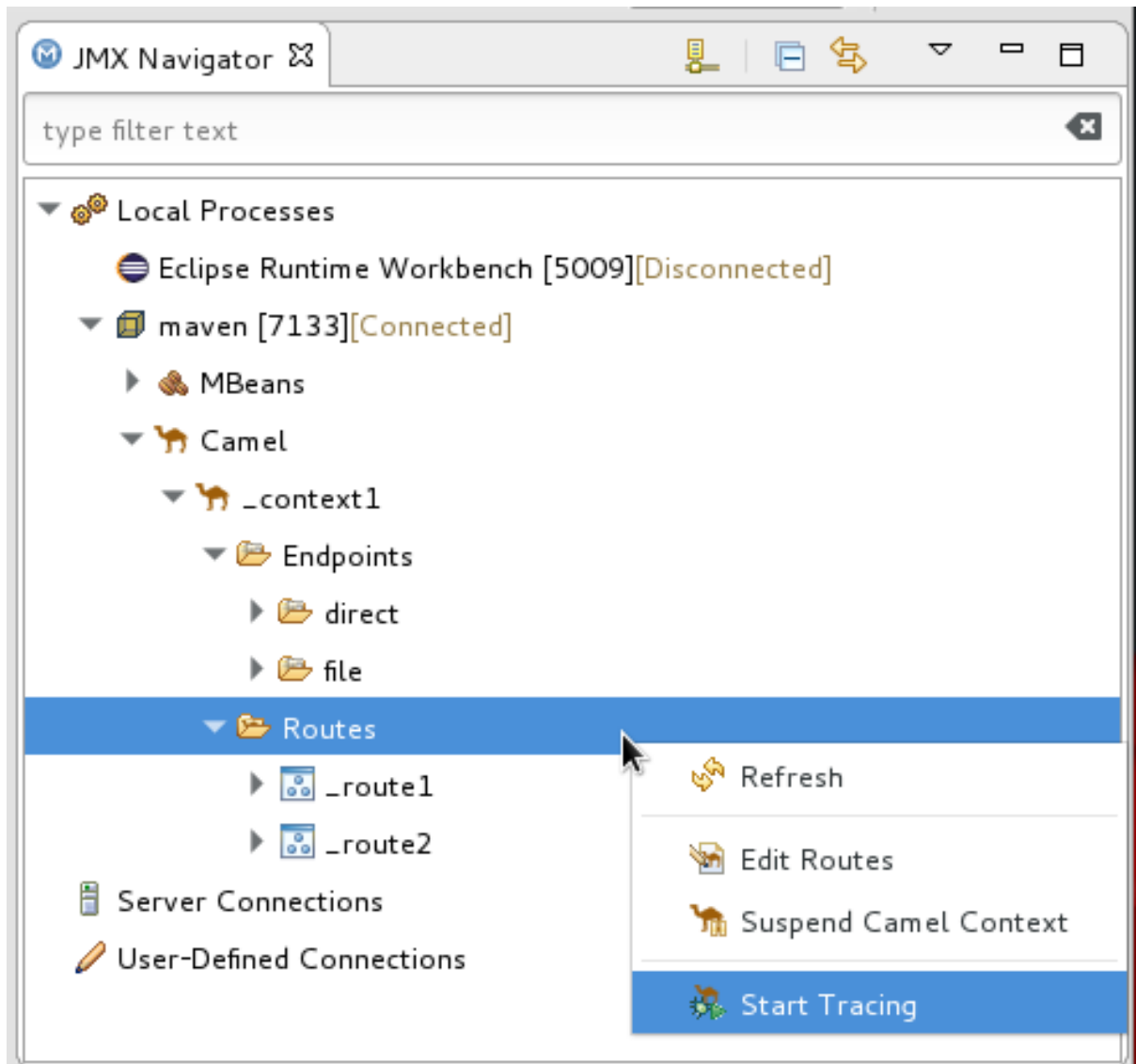
4. JMX Navigator で、**Local Processes** をデプロイメントします。



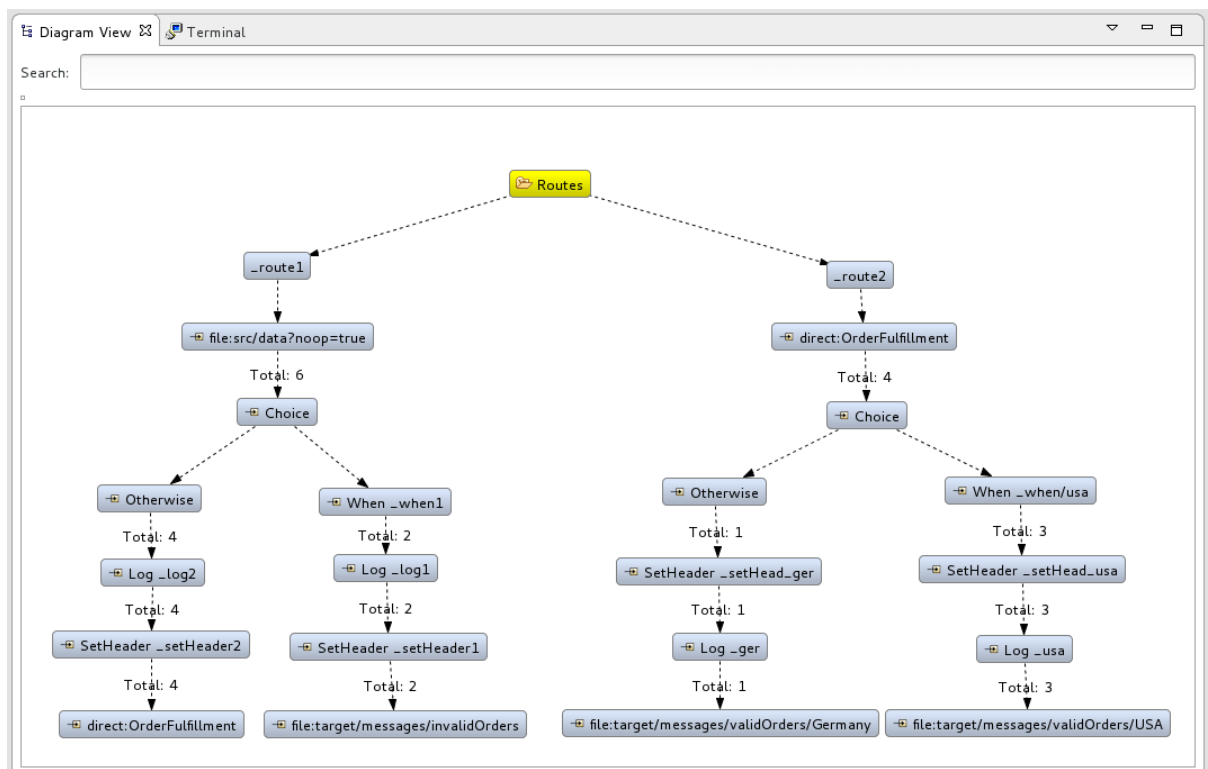
5. **maven [ID]** ノードを右クリックし、**Connect** を選択します。
6. ルートの要素をデプロイメントします。



7. **Routes** ノードを右クリックし、**Start Tracing** を選択します。



ツールは、ルーティングコンテキストのグラフィック表現を **Diagram View** に表示します。

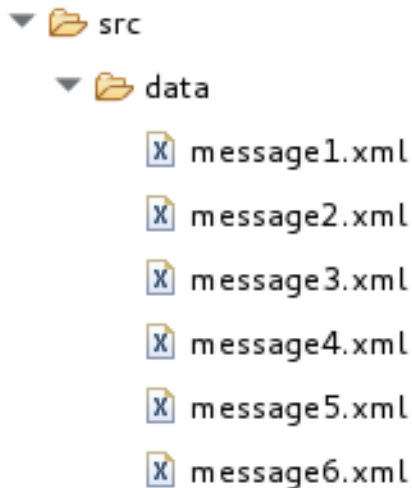


すべてのメッセージフローパスを明確に表示するには、ノードをドラッグして **Diagram View** タブにぴったり収まるようにノードを再配置する必要があります。また、Red Hat CodeReady Studio の他のビューとタブのサイズを調整して、**Diagram View** タブを拡張できるようにする必要があります。

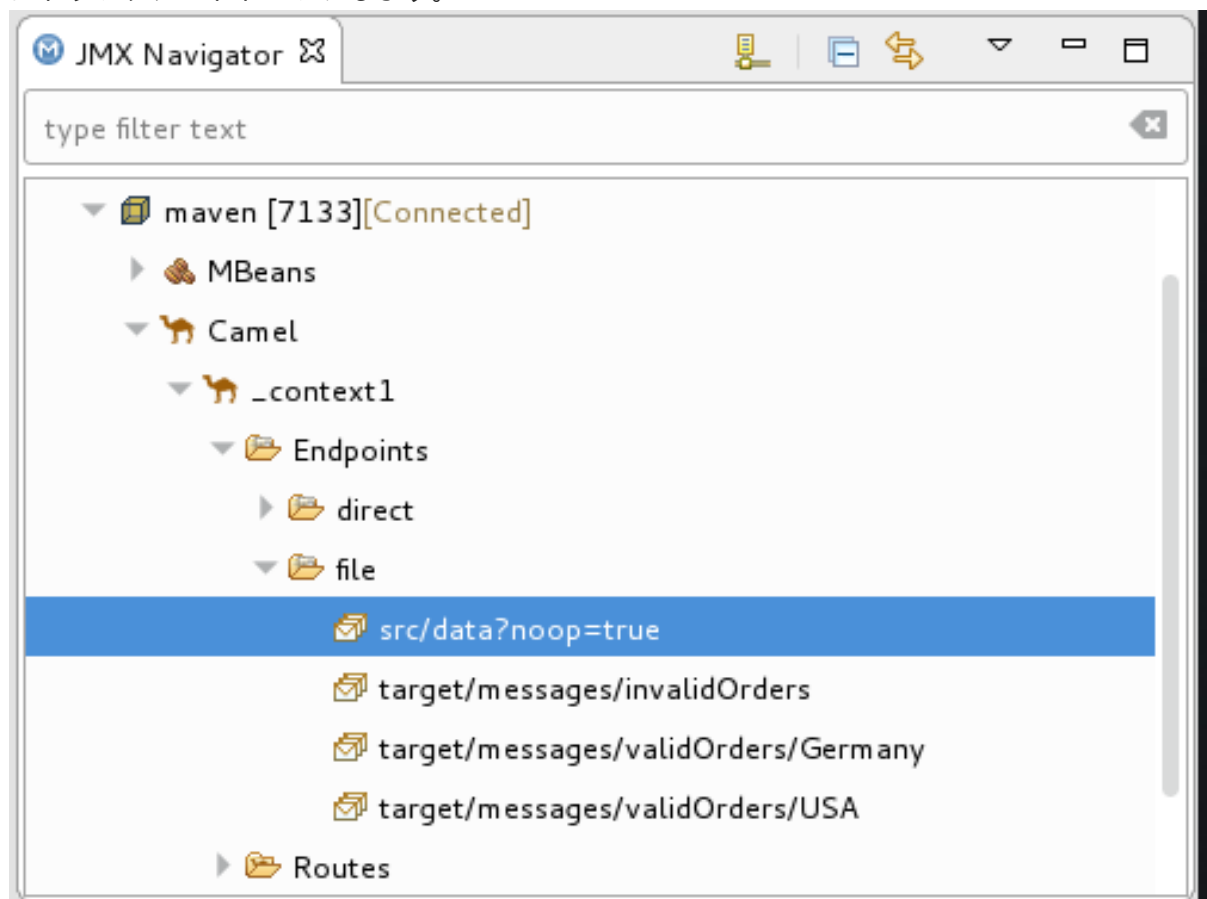
実行中の ZOOORDERAPP プロジェクトにメッセージをドロップする

実行中の ZooOrderApp プロジェクトにメッセージをドロップするには:

1. **Project Explorer** で、メッセージファイル (**message1.xml** から **message6.xml** まで) にアクセスできるように **ZooOrderApp/src/data** をデプロイメントします。





2. **message1.xml** を JMX Navigator の **_context1>Endpoints>file>src/data?noop=true** ノードにドラッグアンドドロップします。

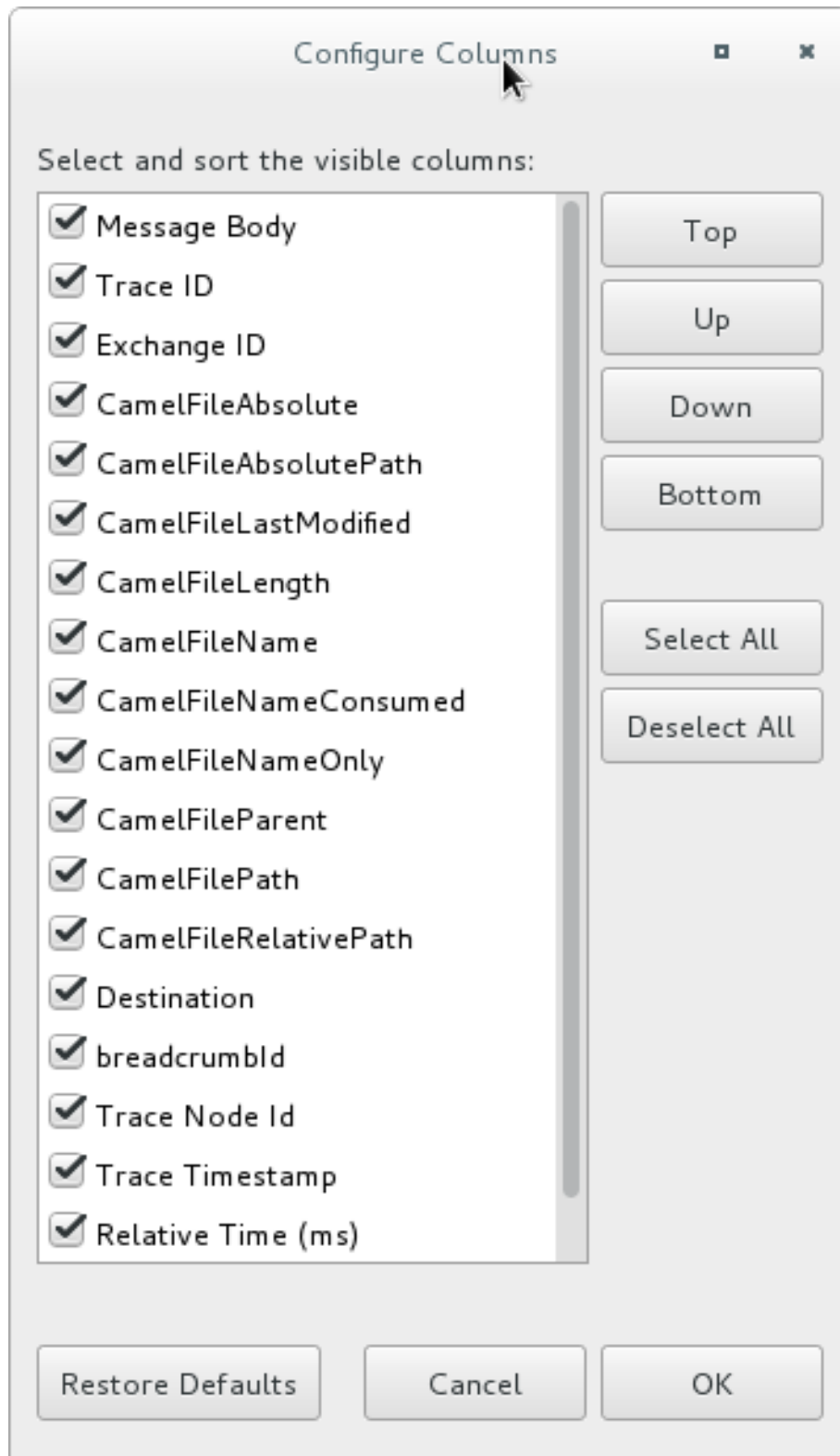


メッセージがルートを通過すると、ツールは各ステップでその通過をトレースして記録します。

メッセージビューの設定

メッセージトレースを表示する前に、**Messages View** を更新する必要があります。また、すべてのメッセージトレースにわたって列を保持する場合は、**Messages View** で列を設定する必要があります。

1. **Messages View** を開きます。
2. パネルのメニューバー右上にある  (リフレッシュボタン) をクリックし、表示に **message1.xml** のメッセージトレースを反映させます。
3. パネルのメニューバーの  アイコンをクリックし、**Configure Columns** を選択して、**Configure Columns** ウィザードを開きます。

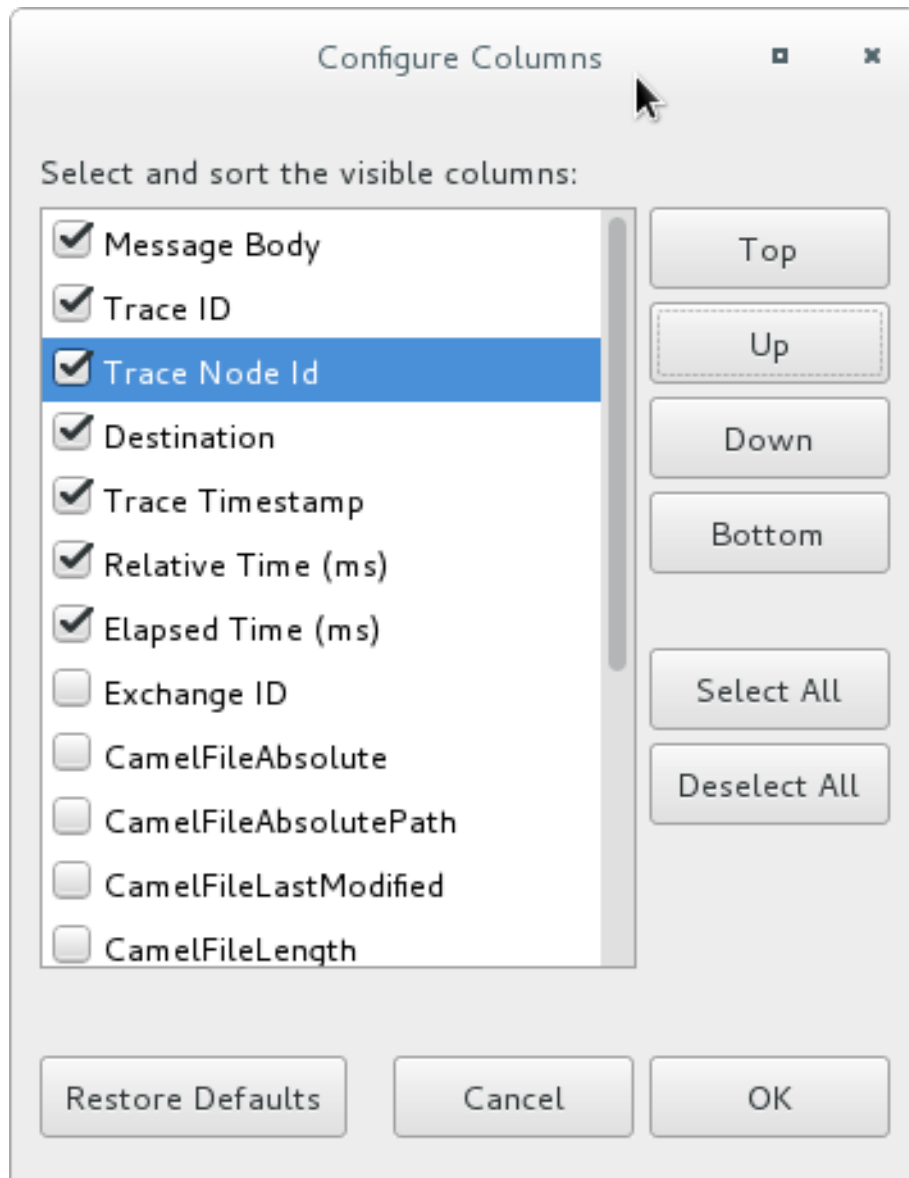


注記

ルーティングコンテキストでメッセージに設定したメッセージヘッダー **Destination** がリストに表示されていることに注意してください。

アイテムを選択または選択解除することにより、**Messages View** にアイテムを含めたり除外したりできます。選択した個々のアイテムを強調表示してリスト内で上下に移動することにより、**Messages View** にアイテムが表示される列の順序を並べ替えることができます。

4. **Configure Columns** ウィザードで、次の方法で列を選択して順序付けします。



これらの列とその順序は、再度変更するまで **Messages View** に保持されます。



注記

ツールのすべてのテーブルで、列方向のレイアウトを制御することができます。ドラッグ方式を使用して、表形式を一時的に再配置します。たとえば、列の境界線ルールをドラッグして、その幅を拡大または縮小します。列を非表示にするには、境界線を完全に縮小します。列ヘッダーをドラッグして、テーブル内の列を再配置します。配置を維持するには、代わりに **View → Configure Columns** メソッドを使用する必要があります。

メッセージトレースのステップスルー

メッセージトレースをステップスルーするには:

1. **message2.xml** を JMX Navigator の **_context1>Endpoints>file>src/data?noop=true** ノードにドラッグアンドドロップします。
2. **Console** から **Messages View** に切り替えます。
3. **Messages View** で 🔄 (リフレッシュボタン) をクリックし、表示に **message2.xml** のメッセージトレースを反映させます。

JMX Navigatorでメッセージをドロップするたびに、Messages Viewを更新してメッセージトレースを入力する必要があります。

4. メッセージトレースのいずれかをクリックし、Propertiesビューでその詳細を表示します。

The screenshot shows two windows in JMX Navigator. The left window is titled 'Properties' and displays a table of metrics for a Camel route. The right window is titled 'Messages View' and displays a table of message traces.

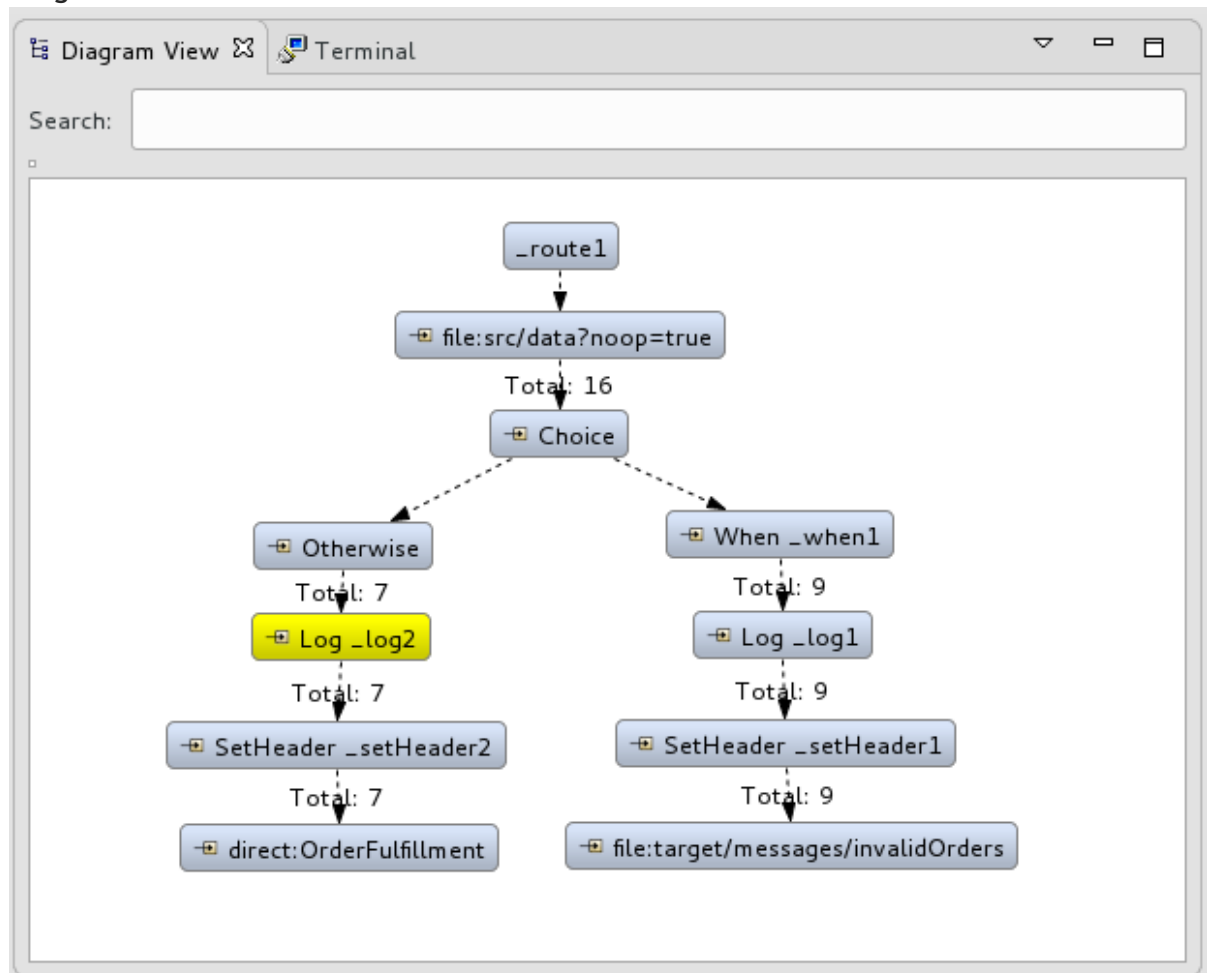
Property	Value
Camel Id	_context1
Exchanges Completed	7
Exchanges Failed	0
External Redeliveries	0
Failures Handled	0
Last Processing Time	0
Max Processing Time	0
Mean Processing Time	0
Min Processing Time	0
Processor Id	_log2
Redeliveries	0
Route Id	_route1
Total Processing Time	0

Message Body	Trace ID	Trace Node Id	Destination	Trace Time
<?xml version='1		_route1		Fri Mar 09 13:
<?xml version='2		_choice1		Fri Mar 09 13:
<?xml version='3		_log1		Fri Mar 09 13:
<?xml version='4		_setHeader1		Fri Mar 09 13:
<?xml version='5		_Invalid	InvalidOrders	Fri Mar 09 13:
<?xml version='6		_route1		Fri Mar 09 13:
<?xml version='7		_choice1		Fri Mar 09 13:
<?xml version='8		_log2		Fri Mar 09 13:
<?xml version='9		_setHeader2		Fri Mar 09 13:
<?xml version='10		_Fulfill	ValidOrders	Fri Mar 09 13:
<?xml version='11		_route2	ValidOrders	Fri Mar 09 13:

このツールでは、メッセージトレースに関する詳細 (メッセージヘッダーが設定されている場合はそれを含む) が Propertiesビューの上半分に、メッセージインスタンスのコンテンツが Propertiesビュー下半分に表示されます。したがって、アプリケーションがルート内の任意のステップでヘッダーを設定する場合は、Message Detailsをチェックして、それらが期待どおりに設定されているかどうかを確認できます。

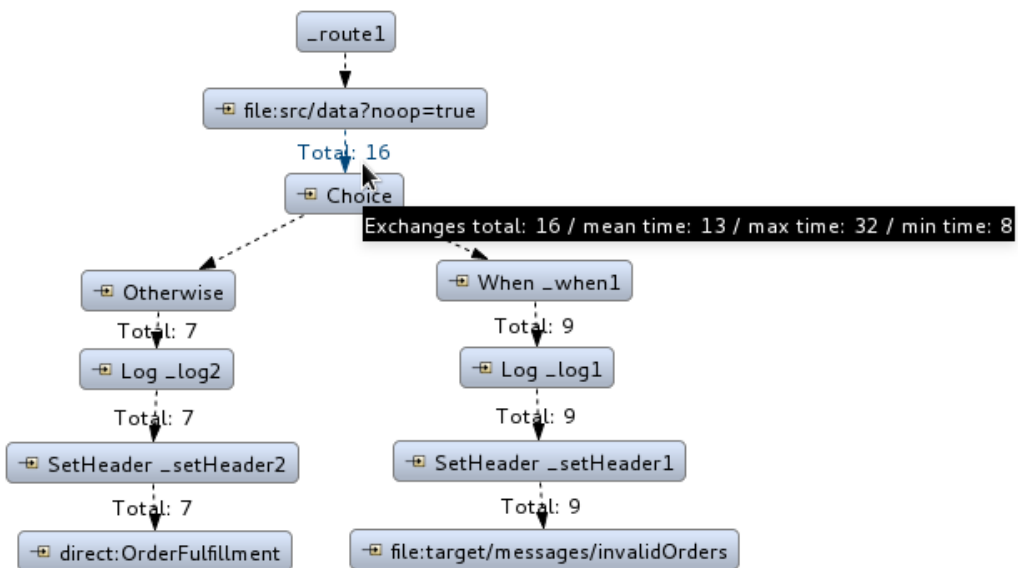
各メッセージを強調表示して、特定のメッセージがルートをどのように通過したか、およびルートの各ステップで期待どおりに処理されたかどうかを確認することで、メッセージインスタンスをステップスルーできます。



5. Diagram Viewを開き、ルートの関連するステップが強調表示されていることを確認します。



ツールは、**Diagram View** でルートを表示し、処理ステップを終了するパスにタイミングとパフォーマンスのメトリック (ミリ秒単位) をタグ付けします。メトリックの **Total exchanges** のみが図に表示されます。

- 表示されたメトリックにマウスポインターを合わせると、メッセージフローに関する追加のメトリックが表示されます。



- ステップがメッセージを処理するのにかかった平均時間
 - ステップがメッセージを処理するのにかかった最大時間
 - ステップがメッセージを処理するのにかかった最小時間
- 任意で、トレースが有効であれば、いつでも **ZooOrderApp/src/data/** の残りのメッセージを **JMX Navigator** の **_context1>Endpoints>file>src/data?noop=true** ノードにドラッグし、ドロップすることができます。
ドロップするたびに、💰 (更新ボタン) をクリックし、**Messages View** に新しいメッセージのトレースを入力することを忘れないでください。
 - それが終わったら:
 - **JMX Navigator** で **_context1** を右クリックし、**Stop Tracing Context** を選択します。
 - **Console** を開いて、パネル右上の  ボタンをクリックしてコンソールを停止します。  ボタンをクリックしてコンソール出力をクリアします。

次のステップ

9章 **JUnit** を使用したルートのテストチュートリアルで、プロジェクトの JUnit テストケースを作成し、プロジェクトを **Local Camel Context** として実行します。

第9章 JUNIT を使用したルートのテスト

このチュートリアルでは、**New Camel Test Case** ウィザードを使用してルートのテストケースを作成し、ルート进行测试する方法を示します。

概要

New Camel Test Case ウィザードは、定型的な JUnit テストケースを生成します。ルートを作成または変更する場合 (たとえば、プロセッサを追加する場合)、作成または変更したルートに固有の期待値とアサーションを追加するために、生成されたテストケースを作成または変更する必要があります。これにより、テストがルートに対して有効であることが保証されます。

ゴール

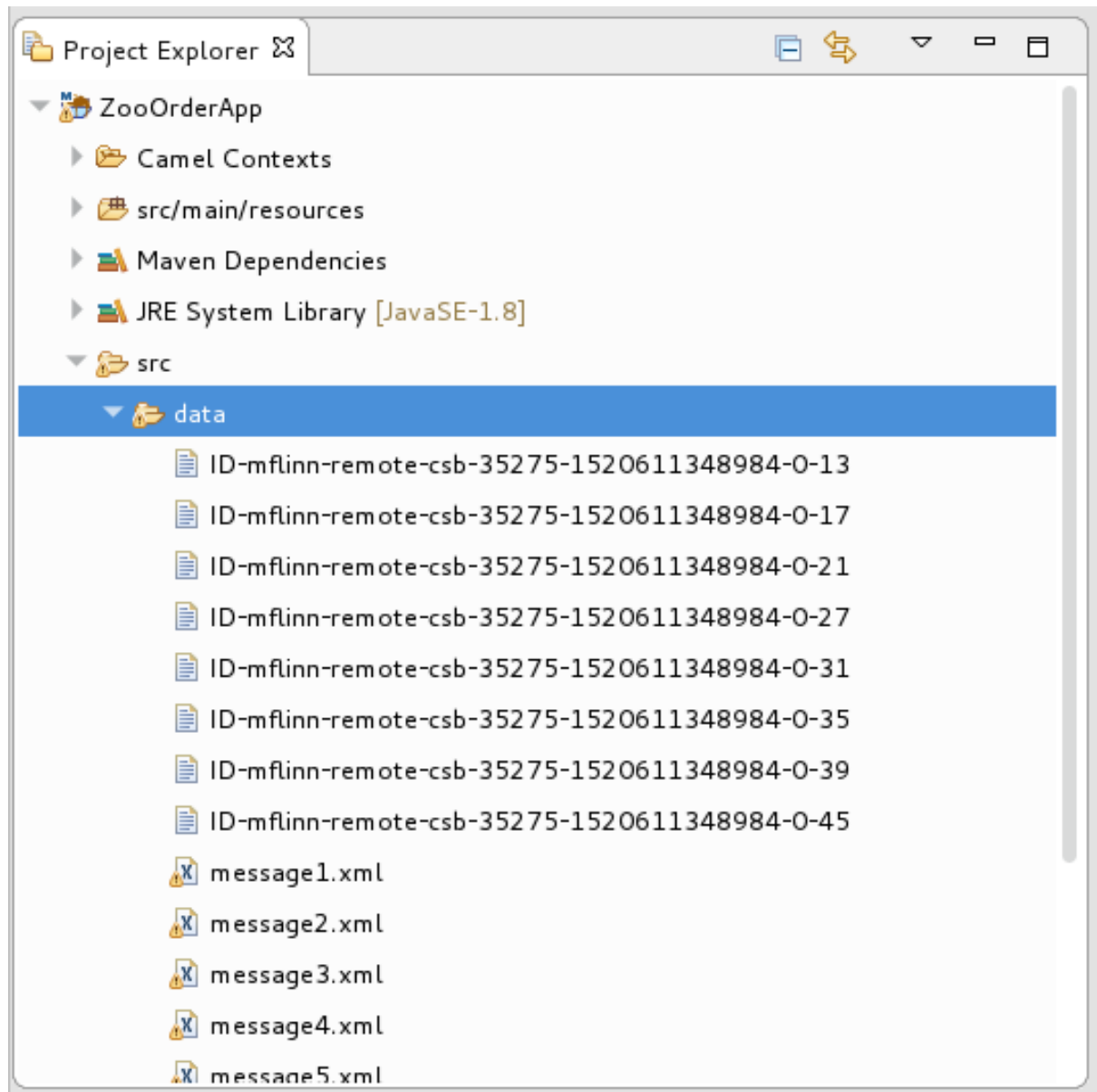
このチュートリアルでは、次のタスクを完了します。

- JUnit テストケースを保存する **/src/test/** フォルダーを作成する
- **ZooOrderApp** プロジェクトの JUnit テストケースを生成する
- 新しく生成された JUnit テストケースを変更します
- **ZooOrderApp** プロジェクトの **pom.xml** ファイルを変更する
- 新しい JUnit テストケースで **ZooOrderApp** を実行する
- 出力を観察します

前提条件

1. このチュートリアルを開始するには、次のいずれかの結果である **ZooOrderApp** プロジェクトが必要です。
 - [8章ルートを介したメッセージのトレースチュートリアル](#)を完了します。
または
 - [2章環境の設定](#)チュートリアルを完了し、「[リソースファイルについて](#)」に記載されているように、プロジェクトの **blueprint.xml** ファイルを、提供される **blueprintContexts/blueprint3.xml** ファイルに置き換える。
2. **Project Explorer** で、**ZooOrderApp** プロジェクトの **/src/data/** ディレクトリーおよび **/target/messages/** サブディレクトリーからトレースにより生成されたメッセージを削除します。トレースにより生成されたメッセージは、**ID-** 接頭辞で始まります。例えば、[図9.1「トレースで生成されたメッセージ」](#)トレースによって生成された8つのメッセージを示します。

図9.1 トレースで生成されたメッセージ

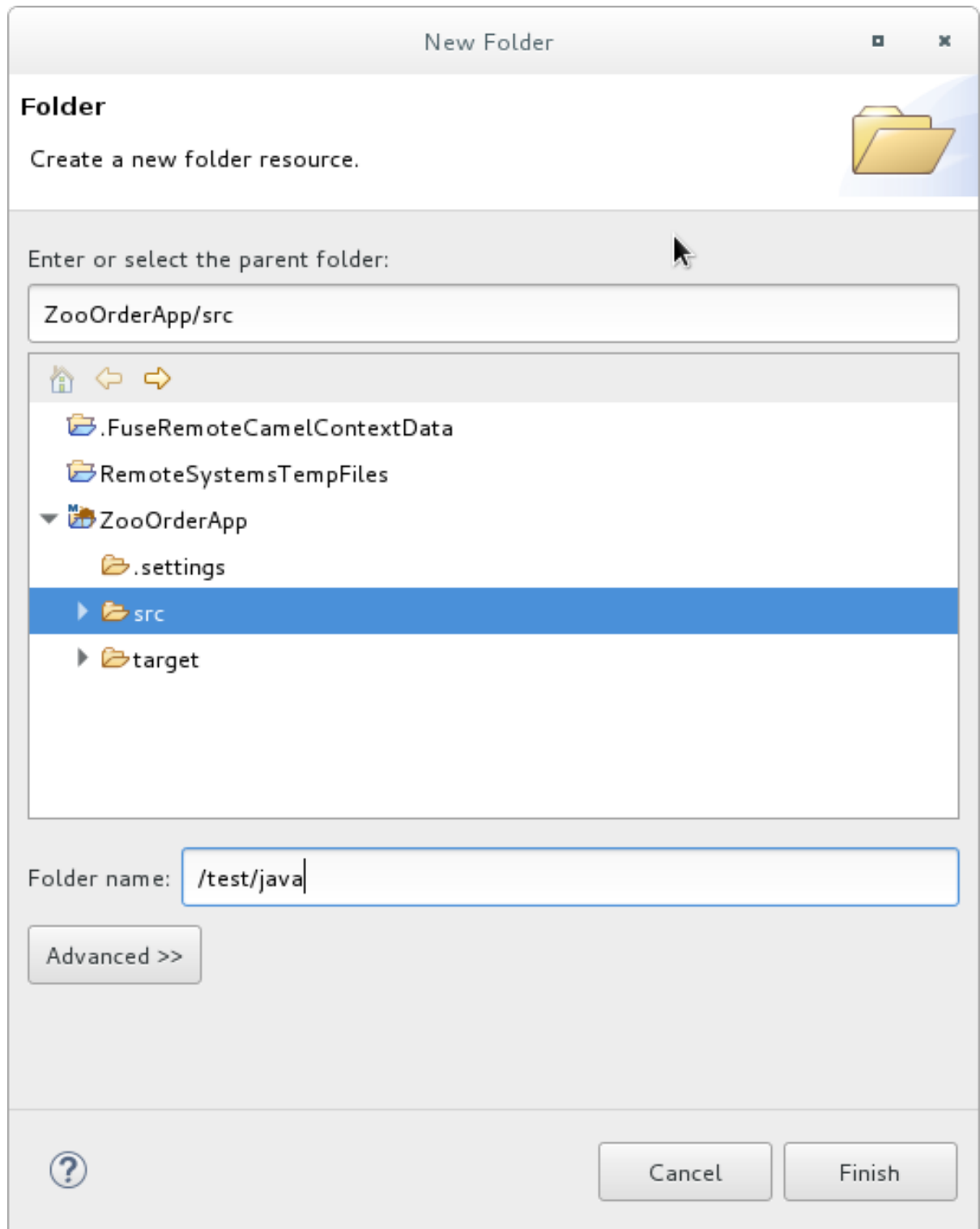


トレースで生成されたすべてのメッセージをバッチで選択し、右クリックして **Delete** を選択します。

SRC/TEST フォルダの作成

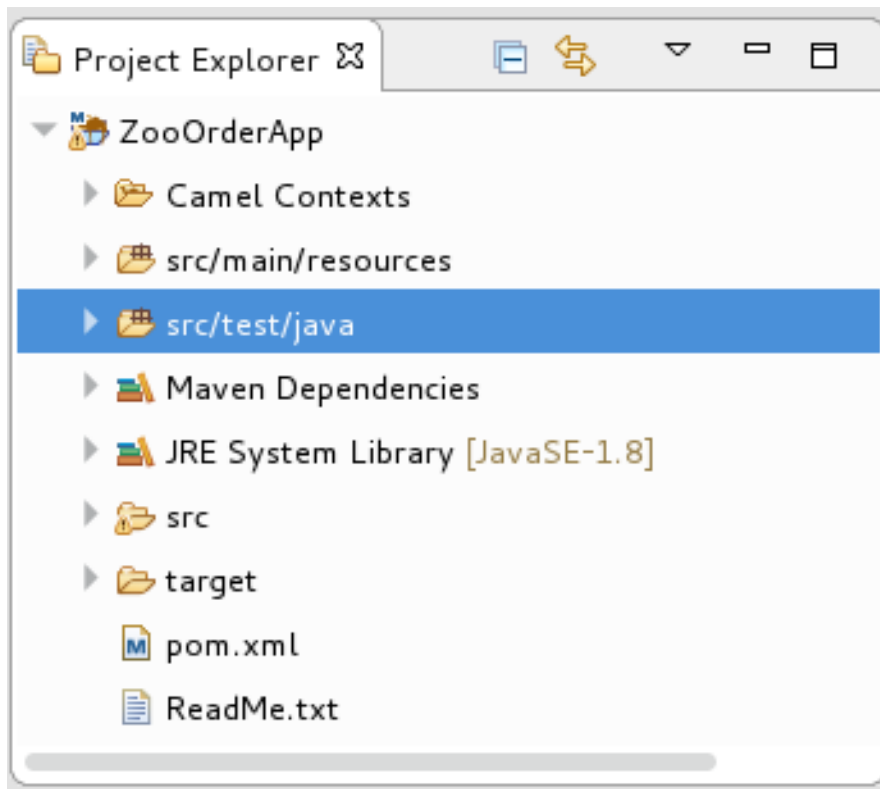
ZooOrderApp プロジェクトの JUnit テストケースを作成する前に、ビルドパスに含まれるフォルダを作成する必要があります。

1. **Project Explorer** で **ZooOrderApp** プロジェクトを右クリックし、**New → Folder** を選択します。
2. **New Folder** ダイアログで、プロジェクトツリーペインで **ZooOrderApp** ノードをデプロイメントし、**src** フォルダを選択します。
ZooOrderApp/src が **Enter or select the parent folder** フィールドに表示されるのを確認してください。
3. **Folder name** に **/test/java** を入力します。

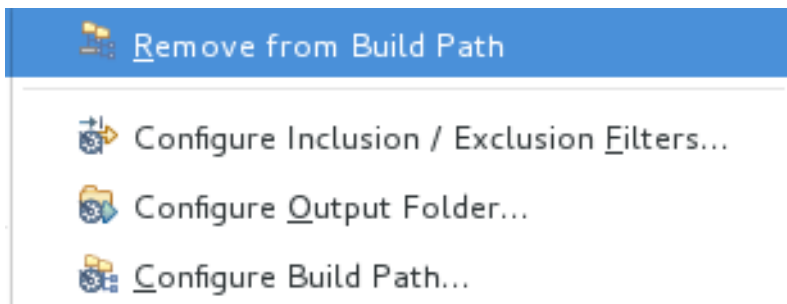


4. **Finish** をクリックします。

Project Explorer では、新しい **src/test/java** フォルダが **src/main/resources** フォルダの下に表示されます。



5. 新しい `/src/test/java` フォルダがビルドパスに含まれていることを確認します。
 - a. **Project Explorer** で `/src/test/java` フォルダを右クリックし、コンテキストメニューを開きます。
 - b. ビルドパスを選択して、メニューオプションを表示します。
Remove from Build Pathのメニューオプションで、`/src/test/java` フォルダが現在ビルドパスに含まれていることが確認できます。



JUNIT テストケースの作成

ZooOrderApp プロジェクトの JUnit テストケースを作成するには、以下を実行します。

1. **Project Explorer** で `src/test/java` を選択します。
2. 右クリックして、**New** → **Camel Test Case** を選択します。

New Camel JUnit Test Case

Camel JUnit Test Case

Select the name of the new JUnit test case. You have the options to specify the Camel XML file under test and on the next page, to select methods to be tested.

Source folder: ZooOrderApp/src/test/java

Package: (default)

Camel XML file under test:

Name:

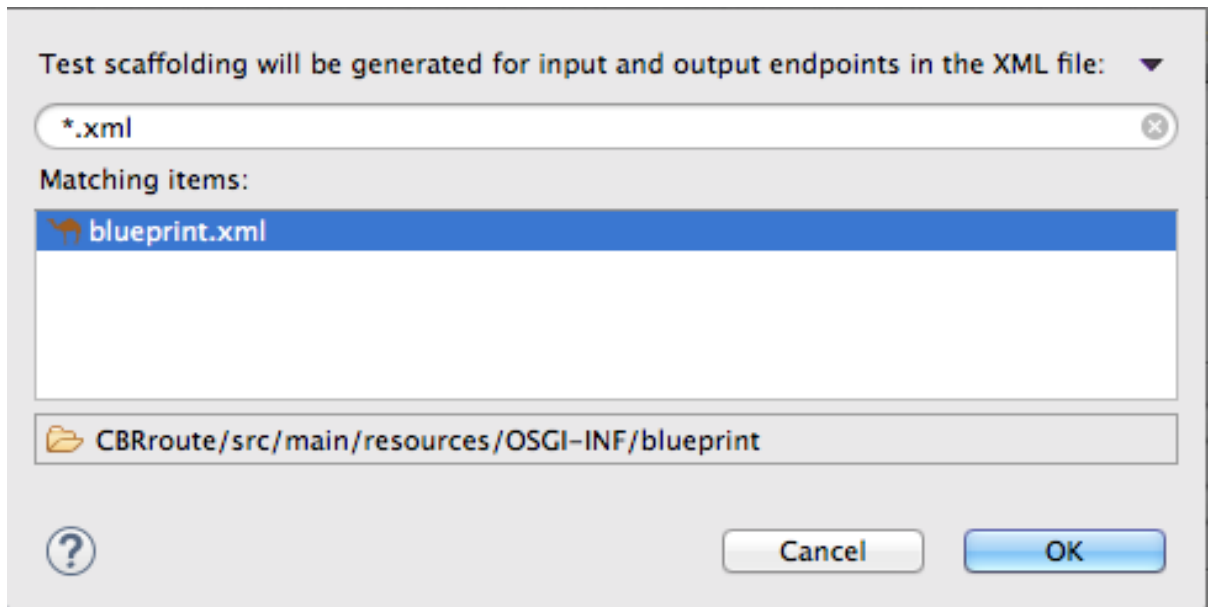
Which method stubs would you like to create?

setUpBeforeClass() tearDownAfterClass()
 setUp() tearDown()
 constructor

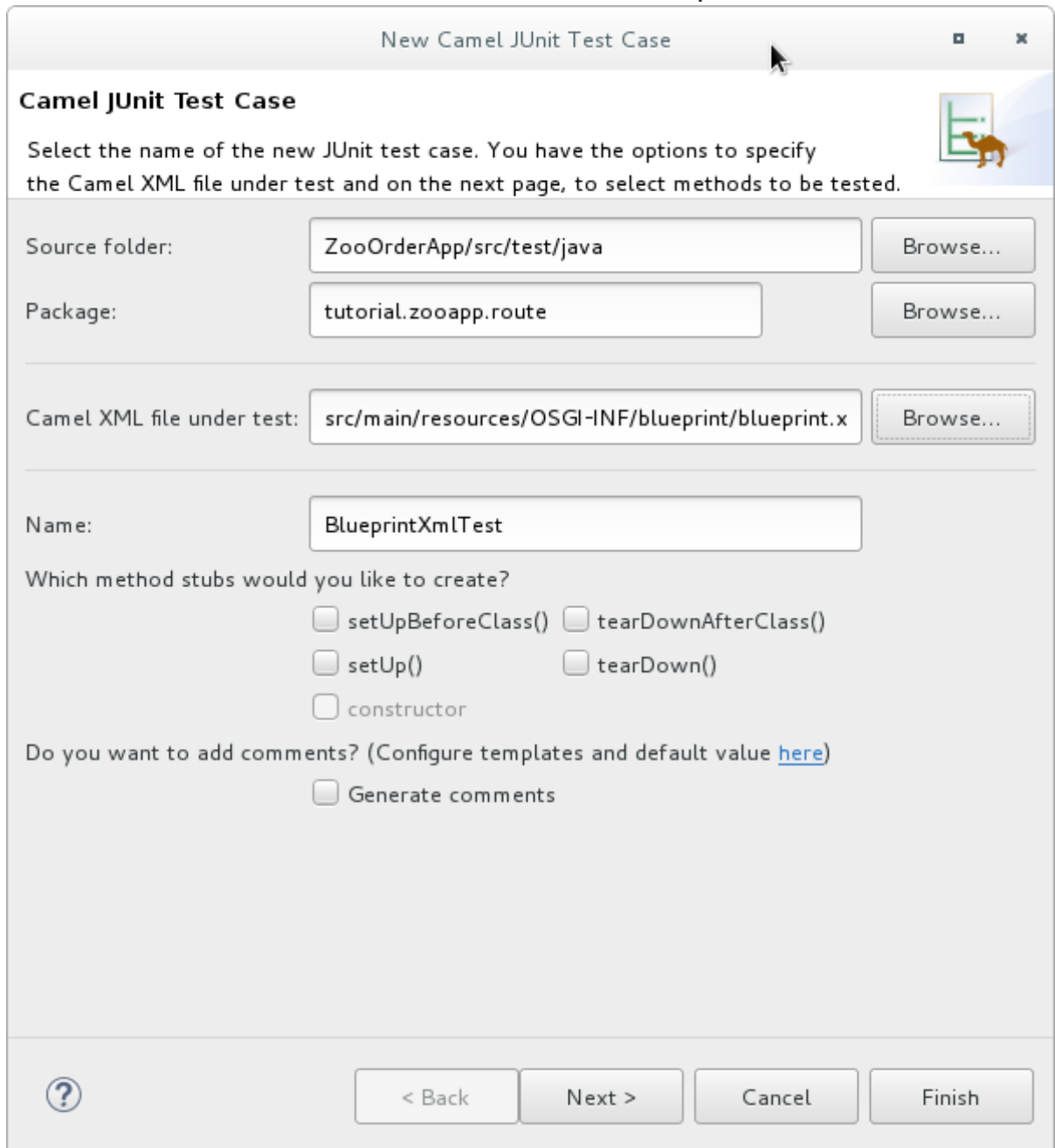
Do you want to add comments? (Configure templates and default value [here](#))

Generate comments

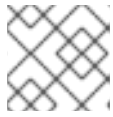
3. Camel JUnit Test Case ウィザードで、**Source folder** フィールドに **ZooOrderApp/src/test/java** が含まれるようにします。適切なフォルダーを見つけるには、 をクリックします。
4. **Package** フィールドに **tutorial.zooapp.route** と入力します。このパッケージには、新しいテストケースが含まれます。
5. **Camel XML file under test** フィールドで、 をクリックし、XML ファイルをフィルターするように設定されたファイルエクスプローラーを開き、**ZooOrderApp** プロジェクトの **blueprint.xml** ファイルを選択します。



6. OK をクリックします。Name フィールドのデフォルトは `BlueprintXmlTest` です。



7. **Next** をクリックして、**Test Endpoints** ページを開きます。
デフォルトでは、すべてのエンドポイントが選択され、テストケースに含まれます。
8. **Finish** をクリックします。



注記

プロンプトが表示されたら、ビルドパスに JUnit を追加します。

テストのアーティファクトはプロジェクトに追加され、**src/test/java** の下にある **Project Explorer** に表示されます。テストケースを実装するクラスは、ツールの Java エディターで開きます。

```
package tutorial.zooapp.route;

import org.apache.camel.EndpointInject;
import org.apache.camel.Produce;
import org.apache.camel.ProducerTemplate;
import org.apache.camel.builder.RouteBuilder;
import org.apache.camel.component.mock.MockEndpoint;
import org.apache.camel.test.blueprint.CamelBlueprintTestSupport;
import org.junit.Test;

public class BlueprintXmlTest extends CamelBlueprintTestSupport {

    // TODO Create test message bodies that work for the route(s) being tested
    // Expected message bodies
    protected Object[] expectedBodies = { "<something id='1'>expectedBody1</something>",
        "<something id='2'>expectedBody2</something>" };
    // Templates to send to input endpoints
    @Produce(uri = "file:src/data?noop=true")
    protected ProducerTemplate inputEndpoint;
    @Produce(uri = "direct:OrderFulfillment")
    protected ProducerTemplate input2Endpoint;
    // Mock endpoints used to consume messages from the output endpoints and then perform
    assertions
    @EndpointInject(uri = "mock:output")
    protected MockEndpoint outputEndpoint;
    @EndpointInject(uri = "mock:output2")
    protected MockEndpoint output2Endpoint;
    @EndpointInject(uri = "mock:output3")
    protected MockEndpoint output3Endpoint;
    @EndpointInject(uri = "mock:output4")
    protected MockEndpoint output4Endpoint;

    @Test
    public void testCamelRoute() throws Exception {
        // Create routes from the output endpoints to our mock endpoints so we can assert expectations
        context.addRoutes(new RouteBuilder() {
            @Override
            public void configure() throws Exception {
                from("file:target/messages/invalidOrders").to(outputEndpoint);
                from("file:target/messages/validOrders/USA").to(output3Endpoint);
                from("file:target/messages/validOrders/Germany").to(output4Endpoint);
            }
        });
    }
}
```

```
// Define some expectations

// TODO Ensure expectations make sense for the route(s) we're testing
outputEndpoint.expectedBodiesReceivedInAnyOrder(expectedBodies);

// Send some messages to input endpoints
for (Object expectedBody : expectedBodies) {
    inputEndpoint.sendBody(expectedBody);
}

// Validate our expectations
assertMockEndpointsSatisfied();
}

@Override
protected String getBlueprintDescriptor() {
    return "OSGI-INF/blueprint/blueprint.xml";
}
}
```

この生成された JUnit テストケースは **ZooOrderApp** プロジェクトには不十分なため、正常に実行できません。「[BlueprintXmlTest ファイルの変更](#)」および「[pom.xml ファイルの変更](#)」に記載されているように、テストケースおよびプロジェクトの **pom.xml** を変更する必要があります。

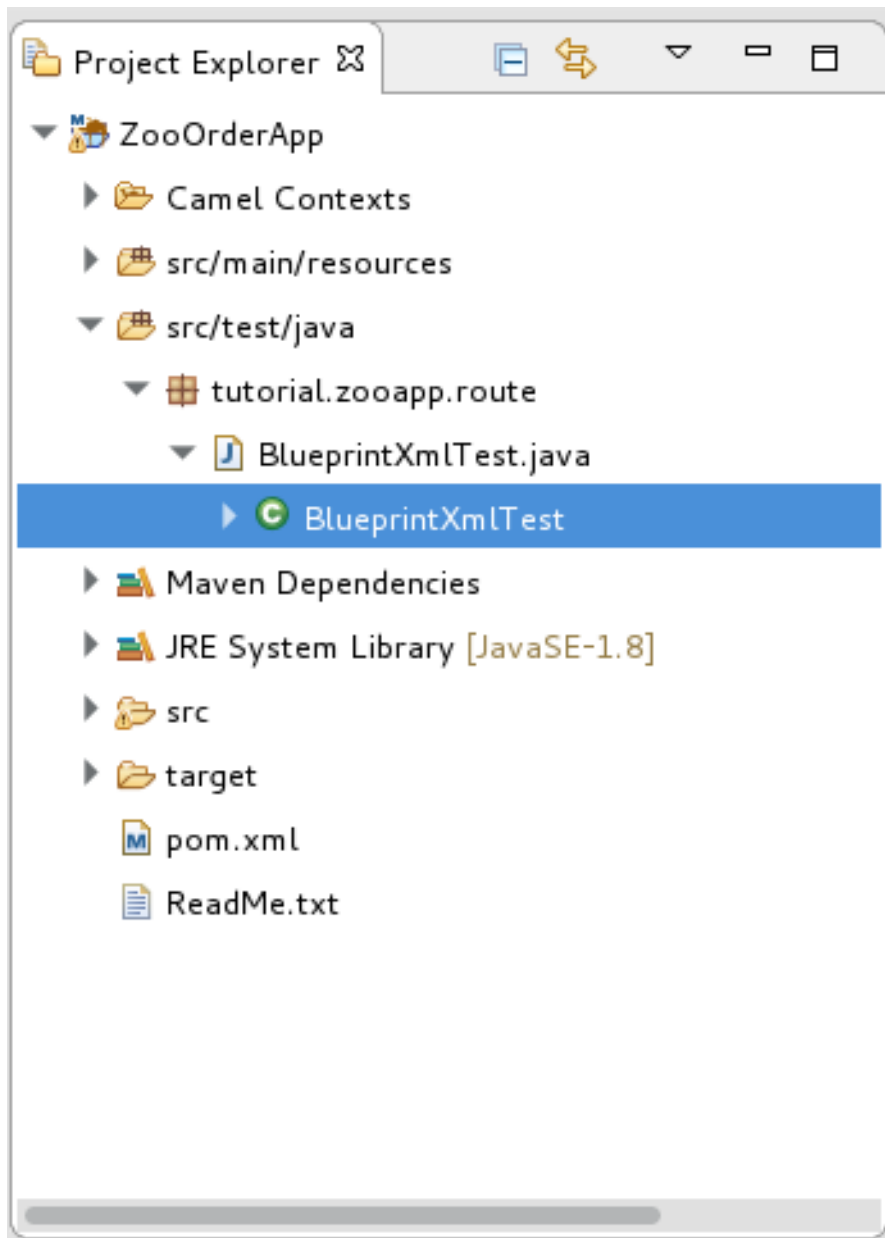
BLUEPRINTXMLTEST ファイルの変更

BlueprintXmlTest.java ファイルを変更し、以下を実行する必要があります。

- 必要なファイル機能をサポートするいくつかのクラスをインポートします
- さまざまなソース **.xml** ファイルの内容を保持する変数を作成する
- ソース **.xml** ファイルの内容を読み取る
- 適切な期待値を定義します

次の手順に従って、**BlueprintXmlTest.java** ファイルを変更します。

1. **Project Explorer** で、**ZooOrderApp** プロジェクトをデプロイメントして **BlueprintXmlTest.java** ファイルを公開します。



2. **BlueprintXmlTest.java** ファイルを開きます。
3. Java エディターで **import org.apache.camel.EndpointInject;** の横にあるデプロイメントボタンをクリックしてリストをデプロイメントします。
4. 太字で示されている 2 行を追加します。最初の行を追加するとエラーが発生しますが、次のセクションで指示どおりに **pom.xml** ファイルを更新すると解決されます。

```
package tutorial.zooapp.route;  
  
import org.apache.camel.EndpointInject;  
import org.apache.camel.Produce;  
import org.apache.camel.ProducerTemplate;  
import org.apache.camel.builder.RouteBuilder;  
import org.apache.camel.component.mock.MockEndpoint;  
import org.apache.camel.test.blueprint.CamelBlueprintTestSupport;  
import org.apache.commons.io.FileUtils;  
import org.junit.Test;  
import java.io.File;
```

5. // **Expected message bodies** のすぐ後の行までスクロールします。
6. **protected Object[] expectedBodies={ expectedBody2</something>"};** の行を **protected String body#;** の行に置き換えます。

```
protected String body1; protected String body2; protected String body3; protected String
body4; protected String body5; protected String body6;
```

7. **public void testCamelRoute() throws Exception {** の行まで下方方向にスクロールし、その直後に以下のように **body# = FileUtils.readFileToString(new File("src/data/message#.xml"), "UTF-8");** の行を挿入します。これらの行は、次のセクションで指示どおりに **pom.xml** ファイルを更新するまでエラーを示します。

```
// Valid orders
body2 = FileUtils.readFileToString(new File("src/data/message2.xml"), "UTF-8");
body4 = FileUtils.readFileToString(new File("src/data/message4.xml"), "UTF-8");
body5 = FileUtils.readFileToString(new File("src/data/message5.xml"), "UTF-8");
body6 = FileUtils.readFileToString(new File("src/data/message6.xml"), "UTF-8");
// Invalid orders
body1 = FileUtils.readFileToString(new File("src/data/message1.xml"), "UTF-8");
body3 = FileUtils.readFileToString(new File("src/data/message3.xml"), "UTF-8");
```

8. // **TODO Ensure expectations make sense for the route(s) we're testing** の直後の行まで下方方向にスクロールします。
9. **outputEndpoint.expectedBodiesReceivedInAnyOrder(expectedBodies);** で始まり **inputEndpoint.sendBody(expectedBody);** } で終わるコードのブロックを、ここに示す行に置き換えます。

```
// Invalid orders
outputEndpoint.expectedBodiesReceived(body1, body3);
// Valid orders for USA
output3Endpoint.expectedBodiesReceived(body2, body5, body6);
// Valid order for Germany
output4Endpoint.expectedBodiesReceived(body4);
```

残りのコードはそのままにしておきます。

10. ファイルを保存します。
11. 更新された **BlueprintXmlTest.java** ファイルに必要な変更が反映されていることを確認します。次のようになります。

```
package tutorial.zooapp.route;

import org.apache.camel.EndpointInject;
import org.apache.camel.Produce;
import org.apache.camel.ProducerTemplate;
import org.apache.camel.builder.RouteBuilder;
import org.apache.camel.component.mock.MockEndpoint;
import org.apache.camel.test.blueprint.CamelBlueprintTestSupport;
import org.apache.commons.io.FileUtils;
import org.junit.Test;
import java.io.file;

public class BlueprintXmlTest extends CamelBlueprintTestSupport {

    // TODO Create test message bodies that work for the route(s) being tested
    // Expected message bodies
    protected String body1;
```

```
protected String body2;
protected String body3;
protected String body4;
protected String body5;
protected String body6;
// Templates to send to input endpoints
@Produce(uri = "file:src/data?noop=true")
protected ProducerTemplate inputEndpoint;
@Produce(uri = "direct:OrderFulfillment")
protected ProducerTemplate input2Endpoint;
// Mock endpoints used to consume messages from the output endpoints and then perform
assertions
@EndpointInject(uri = "mock:output")
protected MockEndpoint outputEndpoint;
@EndpointInject(uri = "mock:output2")
protected MockEndpoint output2Endpoint;
@EndpointInject(uri = "mock:output3")
protected MockEndpoint output3Endpoint;
@EndpointInject(uri = "mock:output4")
protected MockEndpoint output4Endpoint;

@Test
public void testCamelRoute() throws Exception {
    // Create routes from the output endpoints to our mock endpoints so we can assert
    expectations
    context.addRoutes(new RouteBuilder() {
        @Override
        public void configure() throws Exception {
            // Valid orders
            body2 = FileUtils.readFileToString(new File("src/data/message2.xml"), "UTF-8");
            body4 = FileUtils.readFileToString(new File("src/data/message4.xml"), "UTF-8");
            body5 = FileUtils.readFileToString(new File("src/data/message5.xml"), "UTF-8");
            body6 = FileUtils.readFileToString(new File("src/data/message6.xml"), "UTF-8");

            // Invalid orders
            body1 = FileUtils.readFileToString(new File("src/data/message1.xml"), "UTF-8");
            body3 = FileUtils.readFileToString(new File("src/data/message3.xml"), "UTF-8");

            from("file:target/messages/invalidOrders").to(outputEndpoint);
            from("file:target/messages/validOrders/USA").to(output3Endpoint);
            from("file:target/messages/validOrders/Germany").to(output4Endpoint);
            from("direct:OrderFulfillment").to(output2Endpoint);
        }
    });

    // Define some expectations

    // TODO Ensure expectations make sense for the route(s) we're testing
    // Invalid orders
    outputEndpoint.expectedBodiesReceived(body1, body3);

    // Valid orders for USA
    output3Endpoint.expectedBodiesReceived(body2, body5, body6);

    // Valid order for Germany
    output4Endpoint.expectedBodiesReceived(body4);
```

```

// Validate our expectations
assertMockEndpointsSatisfied();
}

@Override
protected String getBlueprintDescriptor() {
    return "OSGI-INF/blueprint/blueprint.xml";
}
}

```

POM.XML ファイルの変更

commons-io プロジェクトの依存関係を ZooOrderApp プロジェクトの **pom.xml** ファイルに追加する必要があります。

1. **Project Explorer** で、**target** フォルダの下での **pom.xml** を選択し、ツールの XML エディターで開きます。
2. ページの下部にある **pom.xml** タブをクリックして、ファイルを表示して編集します。
3. **<dependencies>** セクションの最後に以下の行を追加します。

```

<dependency>
  <groupId>commons-io</groupId>
  <artifactId>commons-io</artifactId>
  <version>2.5</version>
  <scope>test</scope>
</dependency>

```

4. ファイルを保存します。

JUNIT テストの実行

テストを実行するには:

1. より多くのワークスペースを解放するには、**JBoss** パースペクティブに切り替えます。
2. **Project Explorer** で **ZooOrderApp** プロジェクトを右クリックします。
3. **Run As** → **JUnit Test** を選択します。
デフォルトでは、**JUnit** ビューはサイドバーで開きます。(見やすくするには、**Console**、**Servers**、および **Properties** のタブが表示されている右下のパネルにドラッグします。)

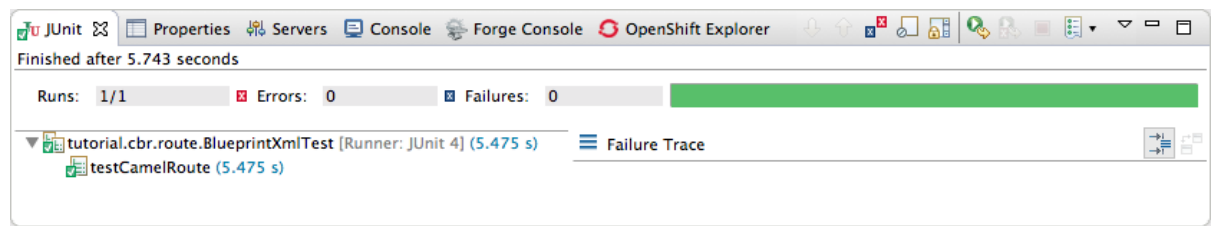


注記

プロジェクトで JUnit を初めて実行すると、テストが失敗することがあります。通常、テストを再実行すると、成功します。

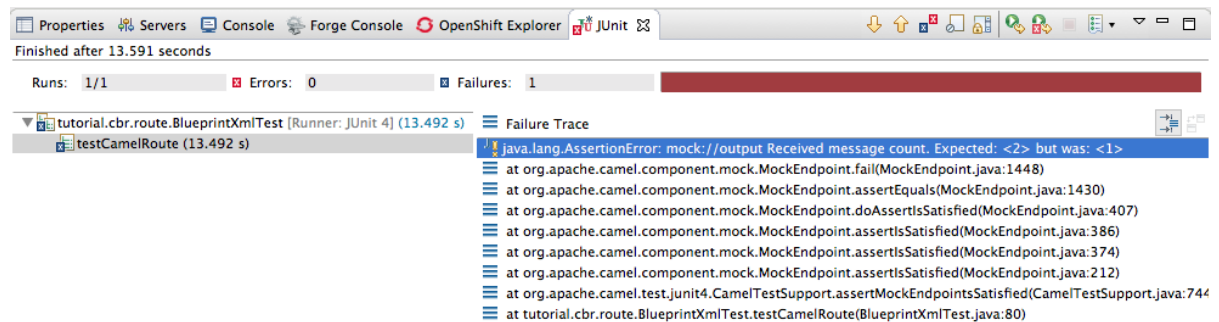
テストが正常に実行されると、次のように表示されます。

図9.2 JUnit の実行が成功しました



テストが失敗すると、次のように表示されます。

図9.3 JUnit の実行に失敗しました



注記

実行環境が Java SE 8 に設定されていない場合、JUnit は失敗します。JUnit タブの上部にあるメッセージバーに、正しい SDK が見つからないことを示すエラーメッセージが表示されます。

この問題を解決するには、プロジェクトのコンテキストメニューを開き、**Run As → Run Configurations → JRE** を選択します。[**Environments**] button next to the ***Execution environment** フィールドをクリックして、Java SE 8 環境を見つけて選択します。

- 出力を調べて、テストの失敗を解決するためのアクションを実行します。

JUnit パネルに表示されるエラーの詳細を表示するには、パネルのメニューバーの をクリックし、ビューを最大化します。

JUnit テストケースを再度実行する前に、**Project Explorer** で ZooOrderApp プロジェクトの **/src/data** フォルダーから JUnit が生成するテストメッセージを削除します (図9.1「**トレースで生成されたメッセージ**」を参照)。

関連資料

JUnit テストの詳細は、[JUnit](#) を参照してください。

次のステップ

10章 [プロジェクトを Red Hat Fuse に公開する](#) チュートリアルでは、Apache Camel プロジェクトを Red Hat Fuse に公開する方法を学びます。

第10章 プロジェクトを RED HAT FUSE に公開する

このチュートリアルでは、プロジェクトを Red Hat Fuse に公開するプロセスについて説明します。Red Hat Fuse Tooling を実行しているのと同じマシンに Red Hat Fuse のインスタンスがインストールされていることを前提としています。

ゴール

このチュートリアルでは、次のタスクを完了します。

- Red Hat Fuse サーバーを定義します
- 公開オプションを設定します
- Red Hat Fuse サーバーを起動し、**ZooOrderApp** プロジェクトをパブリッシュする
- Red Hat Fuse サーバーに接続します
- **ZooOrderApp** プロジェクトのバンドルが正常にビルドされ、公開されたかどうかを確認する
- **ZooOrderApp** プロジェクトのアンインストール

前提条件

このチュートリアルを開始する前に、次のものがが必要です。

- Red Hat Fuse インスタンスへのアクセス
- コンピューターにインストールされている Java 8
- 次のいずれかから生じる ZooOrderApp プロジェクト:
 - [9章JUnit を使用したルートのテスト](#) チュートリアルを完了する。
または
 - [2章環境の設定](#)チュートリアルを完了し、「リソースファイルについて」に記載されているように、プロジェクトの **blueprint.xml** ファイルを、提供される **blueprintContexts/blueprint3.xml** ファイルに置き換える。

RED HAT FUSE SERVER の定義

サーバーを定義するには:

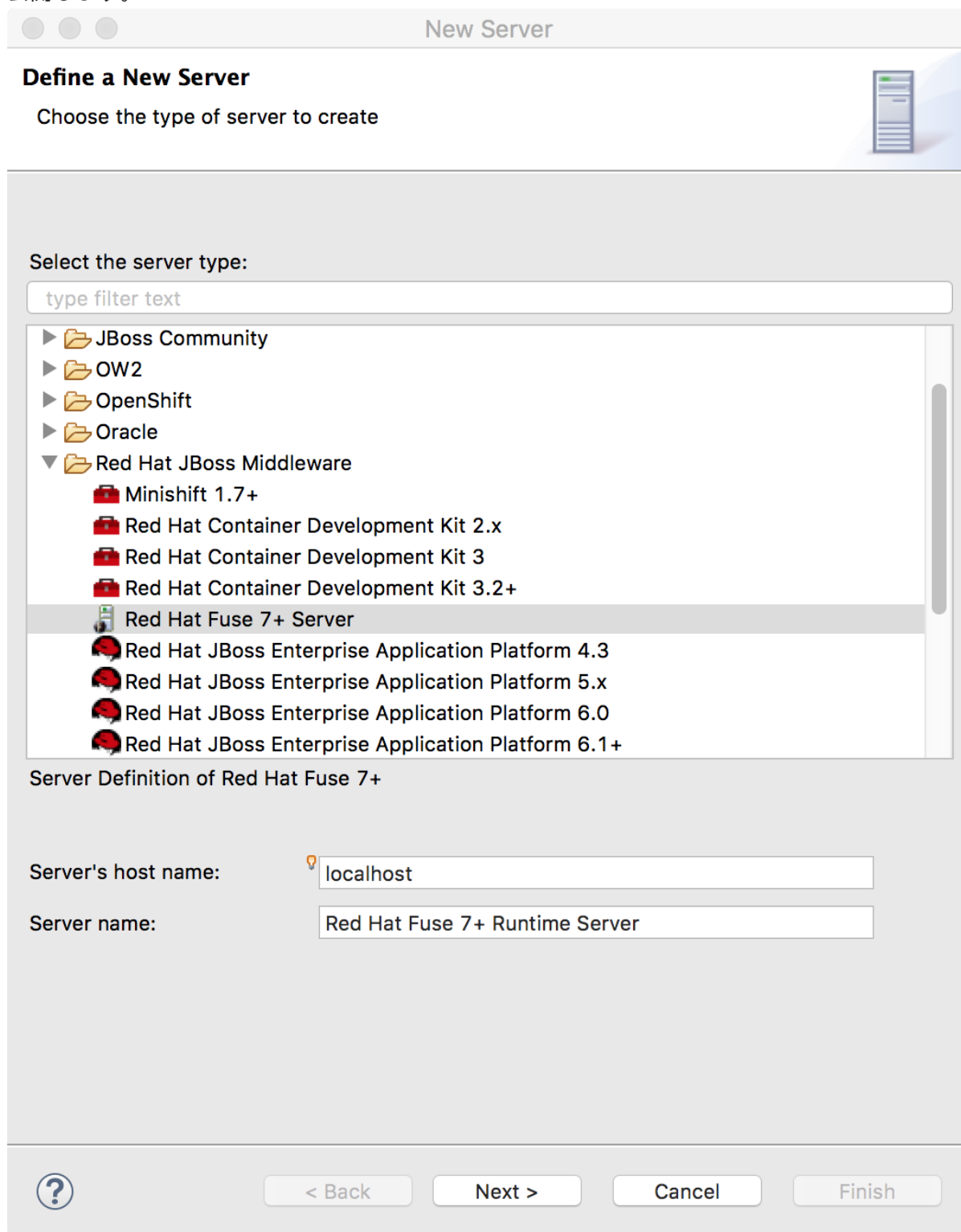
1. **Fuse Integration** パースペクティブを開きます。
2. 右下のパネルの **Servers** タブをクリックして、**Servers** ビューを開きます。
3. 以下をクリックします。利用可能なサーバーはありません。このリンクをクリックして、新しいサーバーを作成...リンクをクリックすると、**Define a New Server** ページが表示されます。



注記

すでにサーバーが定義されているときに新しいサーバーを定義するには、**Servers** ビュー内を右クリックして、**New → Server** を選択します。

4. Red Hat JBoss Middleware ノードをデプロイメントして、使用可能なサーバーオプションを公開します。




5. Red Hat Fuse サーバーを選択します。
6. **Server's host name** (localhost) と **Server name** (Fuse n.n Runtime Server) のデフォルトを受け入れ、**Next** へをクリックして **Runtime** ページを開きます。

New Server

Red Hat Fuse Runtime

Runtime definition for Red Hat Fuse 7+



Please point to a Red Hat Fuse installation.

Name

Home Directory [Download and install runtime...](#)

Runtime JRE

Execution Environment:

Alternate JRE:

?



注記

Fuse をまだインストールしていない場合は、**Download and install runtime** リンクを使用して今すぐダウンロードできます。

すでにサーバーを定義している場合、ツールはこのページをスキップし、代わりに設定の詳細ページを表示します。

7. **Name** のデフォルトを受け入れます。
8. **Home Directory** フィールドの横にある **Browse** をクリックして、インストールに移動して選択します。
9. **Execution Environment** の横にあるドロップダウンメニューからランタイム JRE を選択します。
JavaSE-1.8(推奨) を選択します。必要に応じて、**Environments** ボタンをクリックしてリストから選択します。



注記

Fuse サーバーには Java 8 が必要です (推奨)。Execution Environment で選択するには、事前にインストールしておく必要があります。

10. Alternate JRE オプションはそのままにしておきます。
11. Next をクリックして Fuse Server のランタイム定義を保存し、Fuse server configuration details ページを開きます。

12. SSH Port のデフォルト (8101) を受け入れます。
ランタイムは SSH ポートを使用して、サーバーの Karaf シェルに接続します。このデフォルトが正しくない場合は、Red Hat Fuse `installDir/etc/org.apache.karaf.shell.cfg` ファイルを確認して正しいポート番号を検出できます。
13. User Name に、サーバーへのログインに使用する名前を入力します。
これは、Red Hat Fuse `installDir`/etc/users.properties`` ファイルに保存されているユーザー名です。



注記

`/etc/users.properties` ファイルのデフォルトのユーザーがアクティベートされている場合 (アンコメント)、ツールは User Name および Password にデフォルトユーザーの名前およびパスワードを自動入力します。

設定されていない場合は、`user=password,role` のフォーマットを使用してそのファイルにユーザーを追加するか (例: `joe=secret,Administrator`)、karaf `jaas` コマンドセットを使用して設定できます。

- `jaas:realms`: レルムをリスト表示します。
- `jaas:manage --index 1`: 最初の (サーバー) レルムを編集します。
- `jaas:useradd <username> <password>`: ユーザーと関連するパスワードを追加します。
- `jaas:roleadd <username> Administrator`: 新規ユーザーのロールを指定します。

- **jaas:update**: 新しいユーザー情報を使用してレルムを更新します。
サーバーに対して jaas レルムがすでに選択されている場合は、コマンド **JBossFuse:karaf@root>jaas:users** を実行してユーザー名を検出できます。
14. **Password** には、サーバーにログインする際に **User name** に必要なパスワードを入力します。
これは、Red Hat Fuse の **installDir/etc/users.properties** ファイルまたは karaf **jaas** コマンドによって設定されたパスワードです。
 15. **Finish** をクリックします。
Runtime Server [stopped, Synchronized] が **Servers** ビューに表示されます。
 16. **Servers** ビューで、ランタイムサーバーをデプロイメントします。



JMX[Disconnected] は、**Runtime Server [stopped, Synchronized]** エントリーの下にノードとして表示されます。

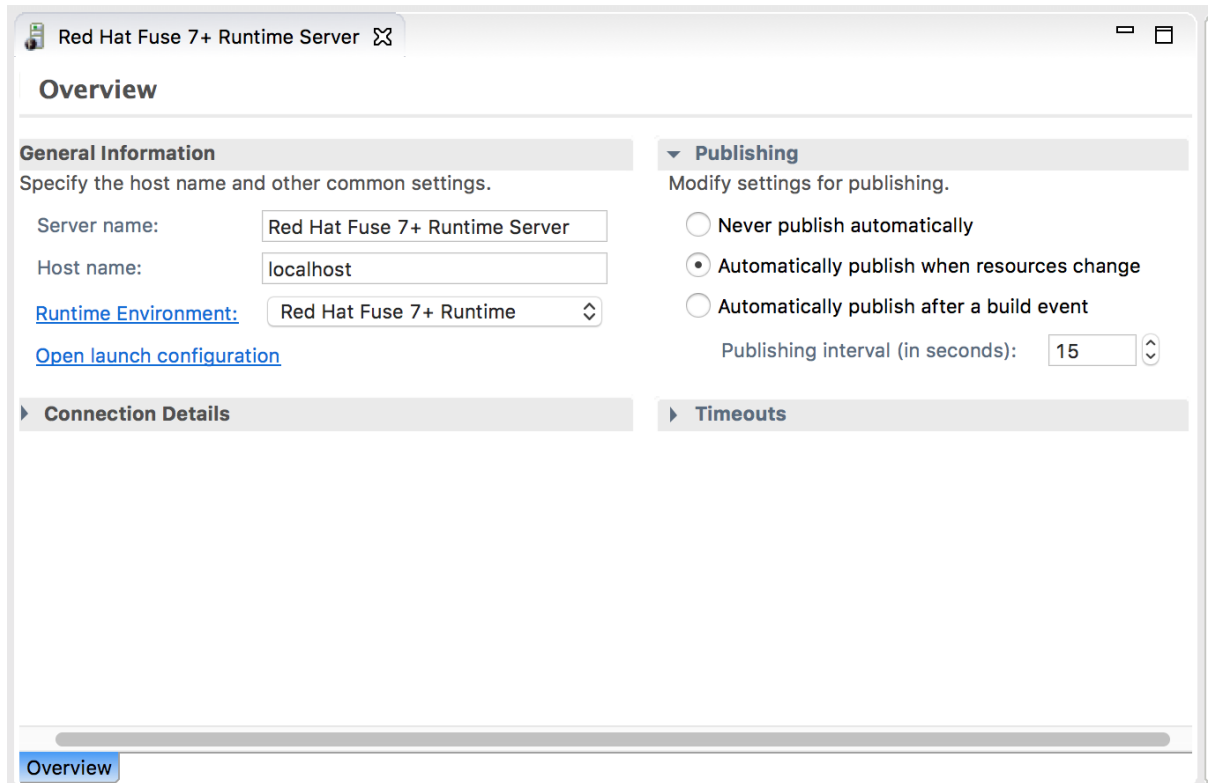
公開オプションの設定

パブリッシュオプションを使用して、**ZooOrderApp** プロジェクトを稼働中のサーバーにパブリッシュする方法およびタイミングを設定できます。

- プロジェクトに加えられた変更を保存するとすぐに自動的に
- プロジェクトを変更して保存した後、設定された間隔で自動的に
- 手動で、公開操作を選択した場合


このチュートリアルでは、**ZooOrderApp** プロジェクトへの変更を保存したら直ちにパブリッシュするように設定します。これを行うには、以下を行います。

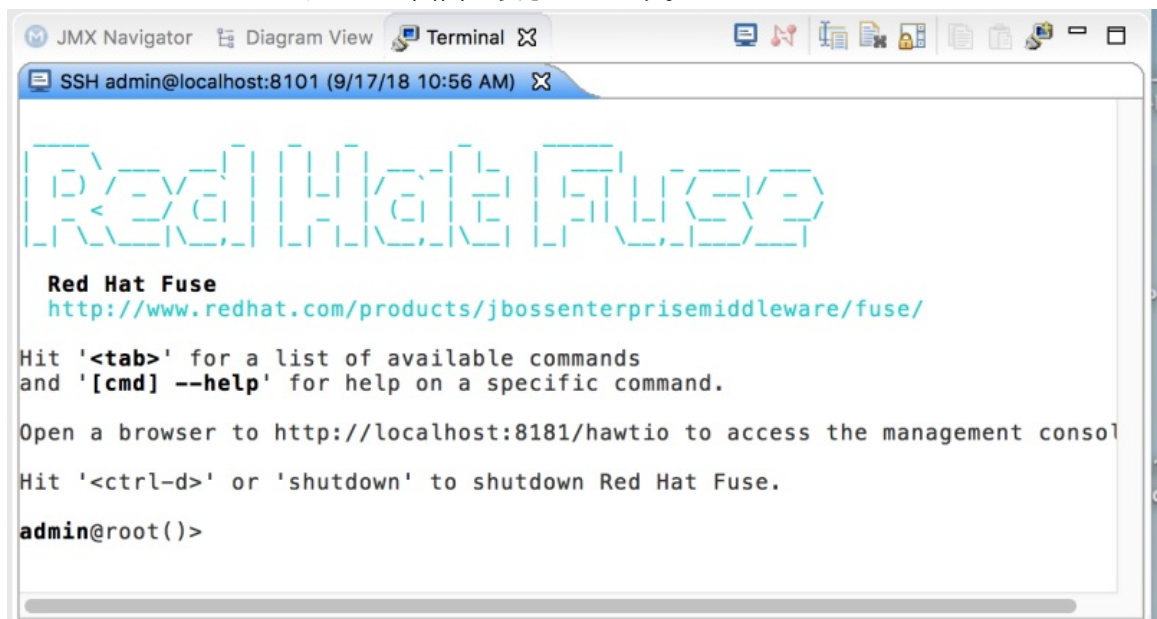
1. **Servers** ビューで、**Runtime Server [stopped, Synchronized]** エントリーをダブルクリックして、その概要を表示します。
2. サーバーの **Overview** ページで、**Publishing** セクションをデプロイメントしてオプションを表示します。



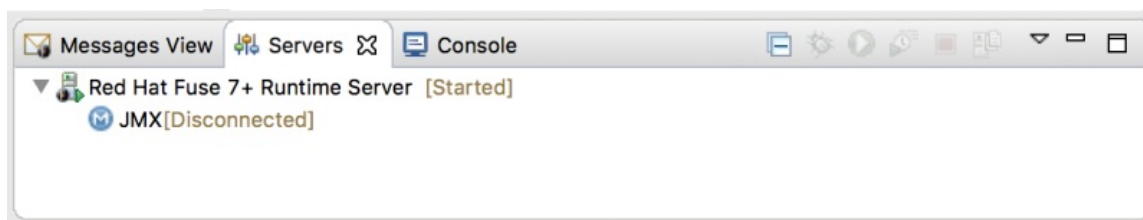
Automatically publish when resources change オプションが有効になっていることを確認します。

必要に応じて、Publishing interval の値を変更して、変更が加えられたときにプロジェクトの公開を高速化または遅延させます。

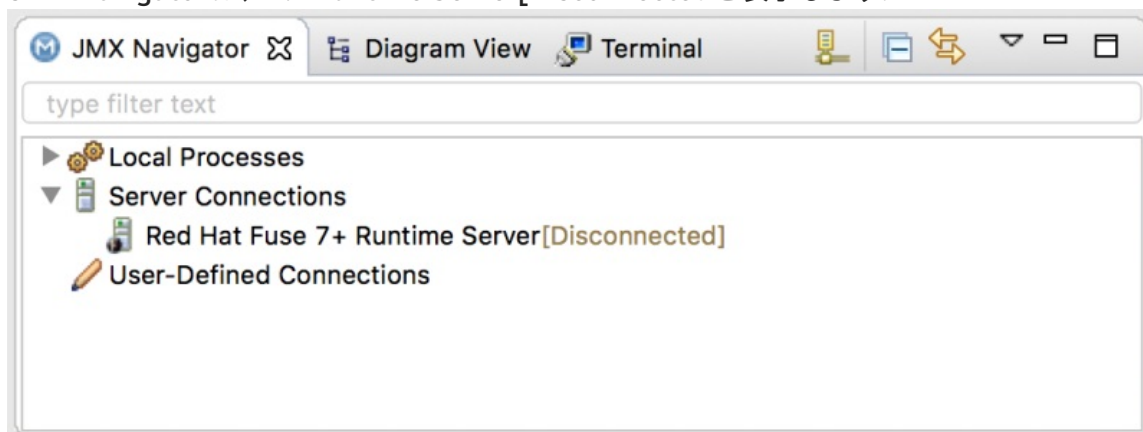
3. Servers ビューで、 をクリックします。
4. サーバーが起動するまで数秒待ちます。その場合:
 - Terminal ビューにスプラッシュ画面が表示されます。



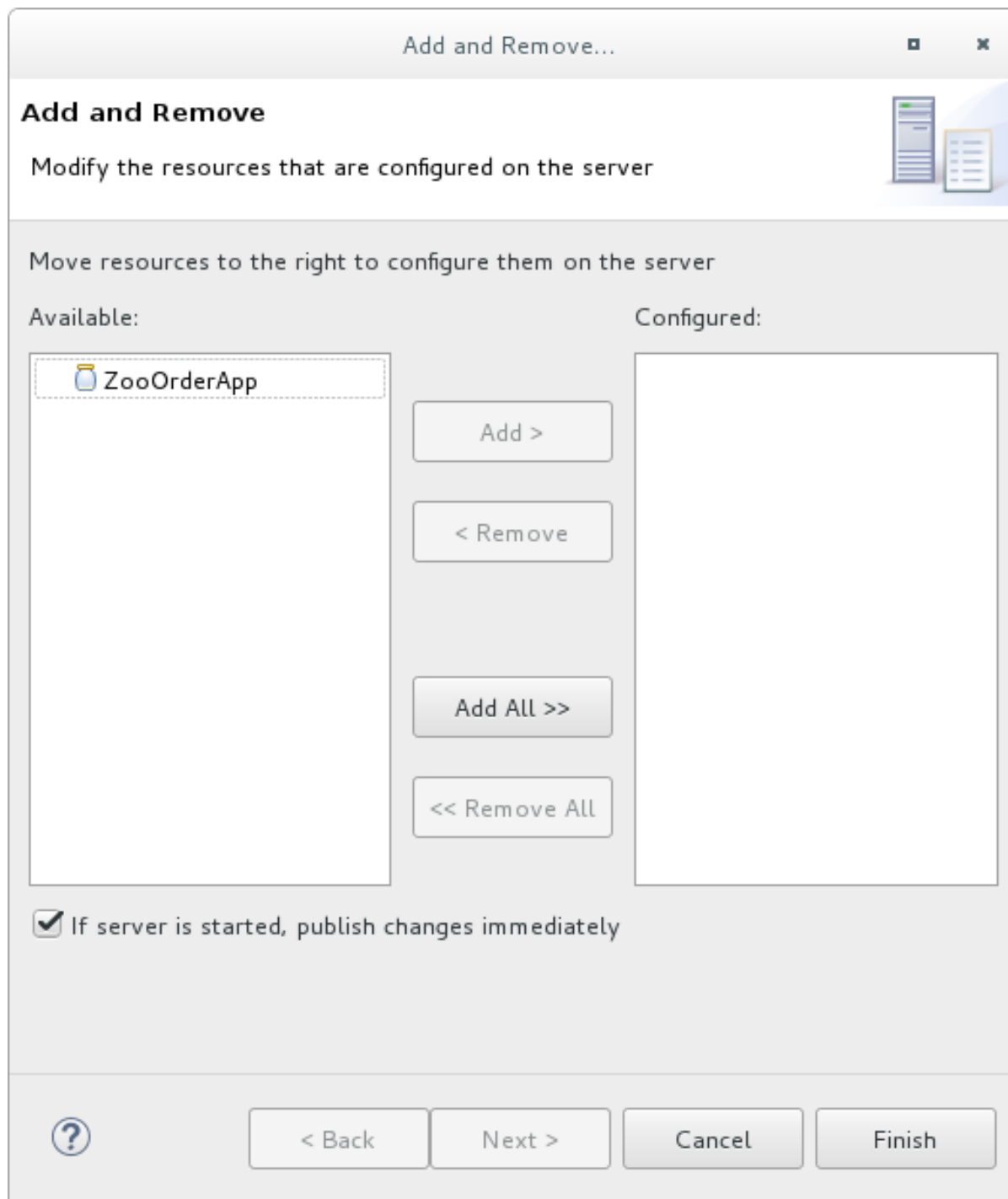
- Servers ビューには以下が表示されます。



- JMX Navigator は、n.n Runtime Server[Disconnected]と表示します:

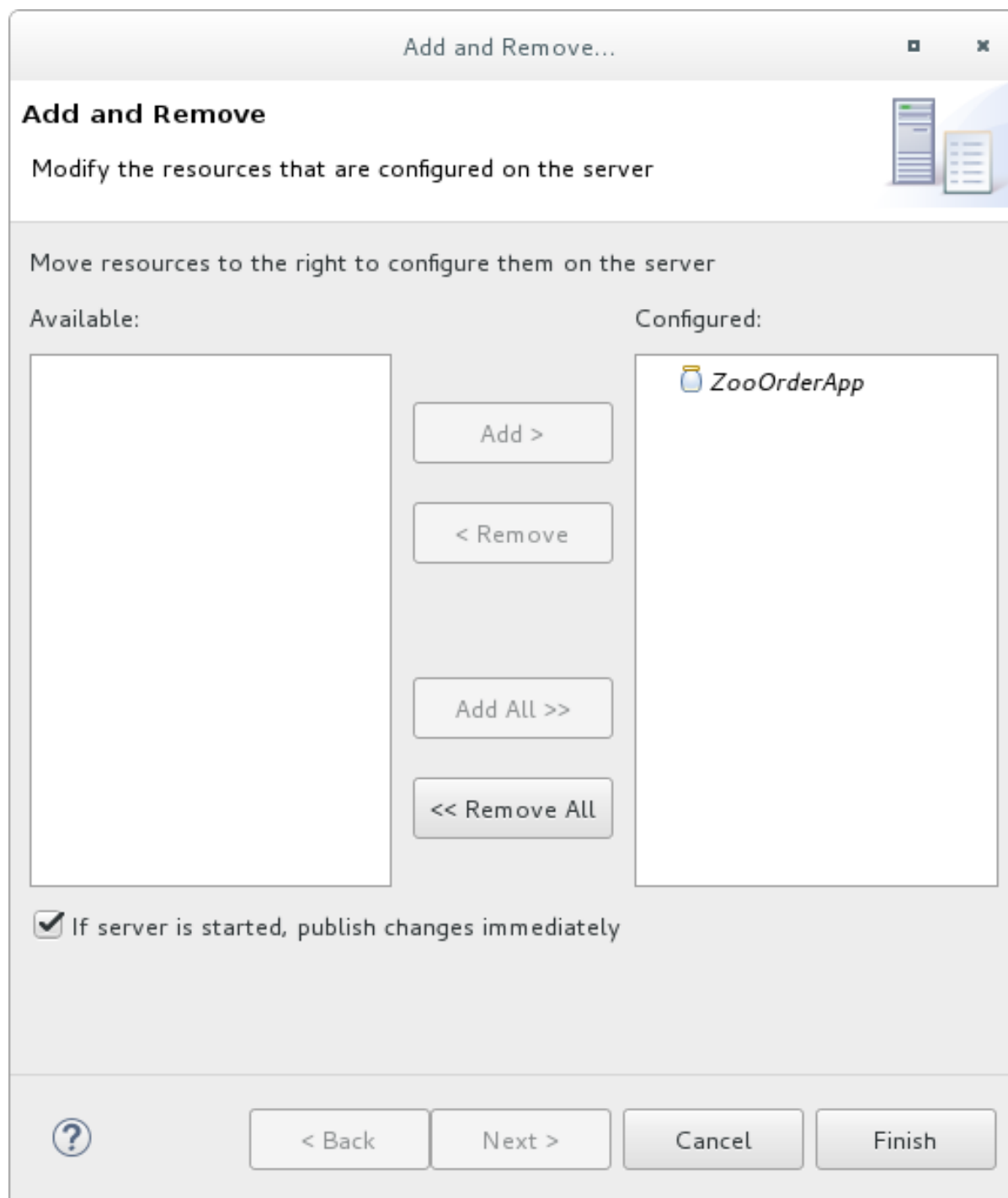


5. Servers ビューで、n.n Runtime Server [Started]を右クリックし、Add and Removeを選択して、Add and Remove ページを開きます。



If server is started, publish changes immediately オプションがオンになっていることを確認します。

6. **ZooOrderApp** を選択し、**Add** をクリックして Fuse サーバーに割り当てます。



7. **Finish** をクリックします。
Servers ビューには、次のように表示されます。



- Runtime Server [Started, Synchronized]



注記

サーバーの場合、**synchronized** とは、サーバーで公開されているすべてのモジュールがローカルの対応するモジュールと同一であることを意味します。

- ZooOrderApp [Started, Synchronized]



注記

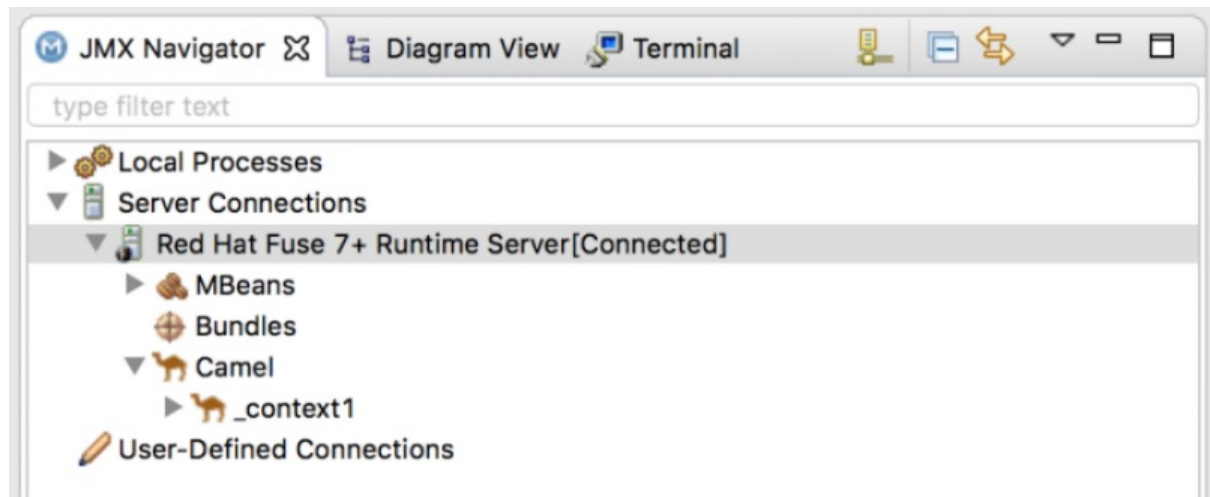
モジュールの場合、**synchronized** とは、公開されたモジュールがローカルの対応するモジュールと同一であることを意味します。自動公開が有効になっているため、ZooOrderApp プロジェクトに加えられた変更は、(Publishing interval の値に応じて) 数秒で公開されます。

- JMX[Disconnected]

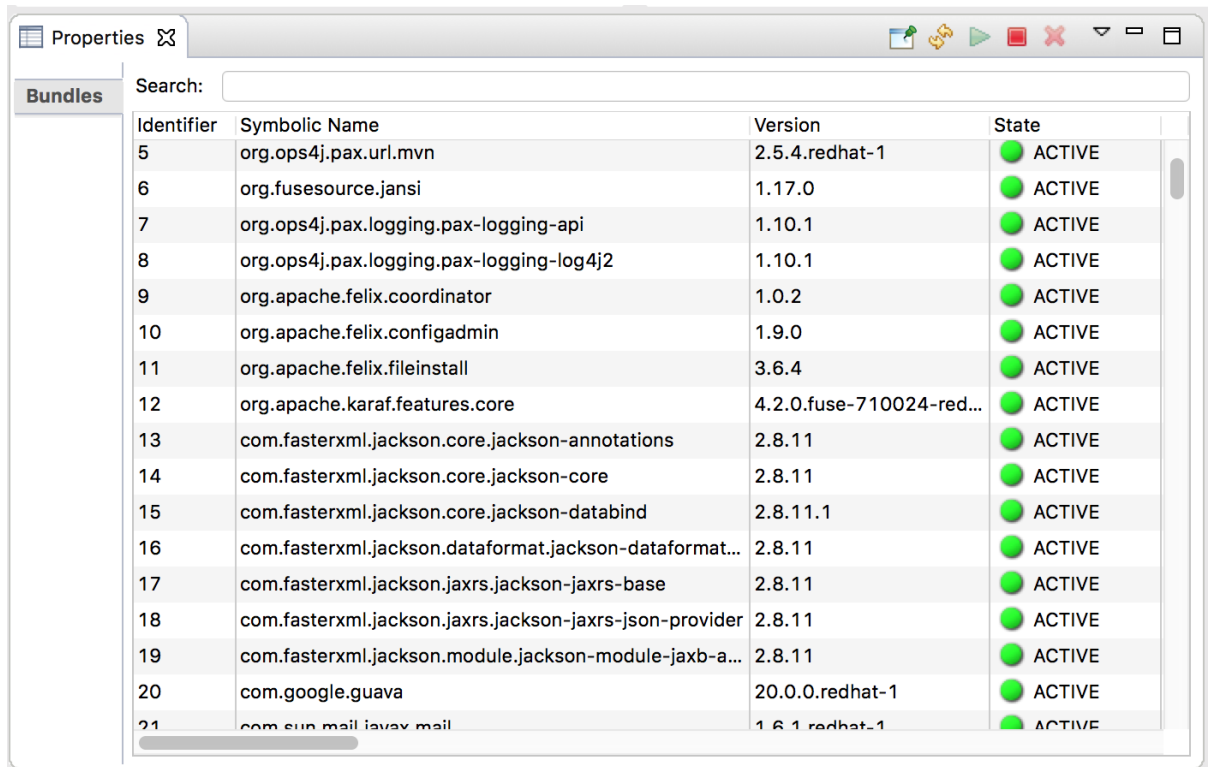
ランタイムサーバーへの接続

ランタイムサーバーへの接続後、**ZooOrderApp** プロジェクトのパブリッシュされた要素が表示され、それらと対話できます。

1. **Servers** ビューで、**JMX[Disconnected]** をダブルクリックしてランタイムサーバーに接続します。
2. **JMX Navigator** で **Camel** フォルダをデプロイメントし、**ZooOrderApp** の要素を表示します。

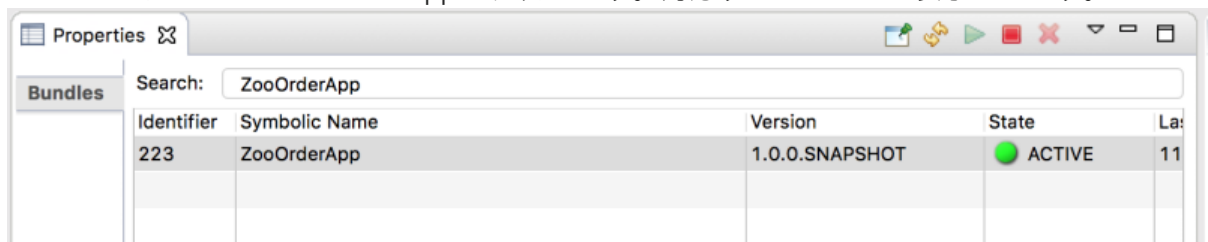


3. **Bundles** ノードをクリックして、ランタイムサーバーにインストールされているバンドルのリストを **Properties** ビューに入力します。



Identifier	Symbolic Name	Version	State
5	org.ops4j.pax.url.mvn	2.5.4.redhat-1	ACTIVE
6	org.fusesource.jansi	1.17.0	ACTIVE
7	org.ops4j.pax.logging.pax-logging-api	1.10.1	ACTIVE
8	org.ops4j.pax.logging.pax-logging-log4j2	1.10.1	ACTIVE
9	org.apache.felix.coordinator	1.0.2	ACTIVE
10	org.apache.felix.configadmin	1.9.0	ACTIVE
11	org.apache.felix.fileinstall	3.6.4	ACTIVE
12	org.apache.karaf.features.core	4.2.0.fuse-710024-red...	ACTIVE
13	com.fasterxml.jackson.core.jackson-annotations	2.8.11	ACTIVE
14	com.fasterxml.jackson.core.jackson-core	2.8.11	ACTIVE
15	com.fasterxml.jackson.core.jackson-databind	2.8.11.1	ACTIVE
16	com.fasterxml.jackson.dataformat.jackson-dataformat...	2.8.11	ACTIVE
17	com.fasterxml.jackson.jaxrs.jackson-jaxrs-base	2.8.11	ACTIVE
18	com.fasterxml.jackson.jaxrs.jackson-jaxrs-json-provider	2.8.11	ACTIVE
19	com.fasterxml.jackson.module.jackson-module-jaxb-a...	2.8.11	ACTIVE
20	com.google.guava	20.0.0.redhat-1	ACTIVE
21	com.sun.mail.javax.mail	1.6.1.redhat-1	ACTIVE

4. Search フィールドに ZooOrderApp と入力します。対応するバンドルが表示されます。

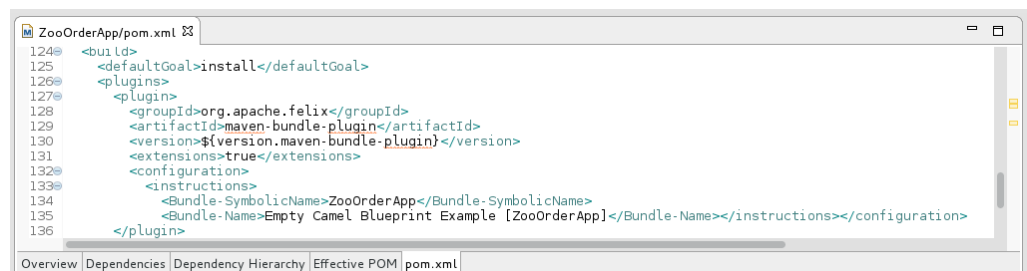


Identifier	Symbolic Name	Version	State	La:
223	ZooOrderApp	1.0.0.SNAPSHOT	ACTIVE	11

注記

または、Terminal ビューで **osgi:list** コマンドを実行し、サーバーランタイムにインストールされているバンドルに関する生成されたリストを表示することもできます。ツールは、**osgi:list** コマンドが表示する OSGi バンドルに異なる命名スキームを使用します。この場合、コマンドは **Camel Blueprint Quickstart** を返します。これはインストールされたバンドルのリストの最後に表示される

プロジェクトの **pom.xml** ファイルの **<build>** セクションで、バンドルのシンボリック名およびそのバンドル名 (OSGi) が **maven-bundle-plugin** エントリーにリスト表示されます。



```

124 <build>
125   <defaultGoal>install</defaultGoal>
126   <plugins>
127     <plugin>
128       <groupId>org.apache.felix</groupId>
129       <artifactId>maven-bundle-plugin</artifactId>
130       <version>${version.maven-bundle-plugin}</version>
131       <extensions>true</extensions>
132       <configuration>
133         <instructions>
134           <Bundle-SymbolicName>ZooOrderApp</Bundle-SymbolicName>
135           <Bundle-Name>Empty Camel Blueprint Example [ZooOrderApp]</Bundle-Name></instructions></configuration>
136       </plugin>

```

ZOOORDERAPP プロジェクトのアンインストール

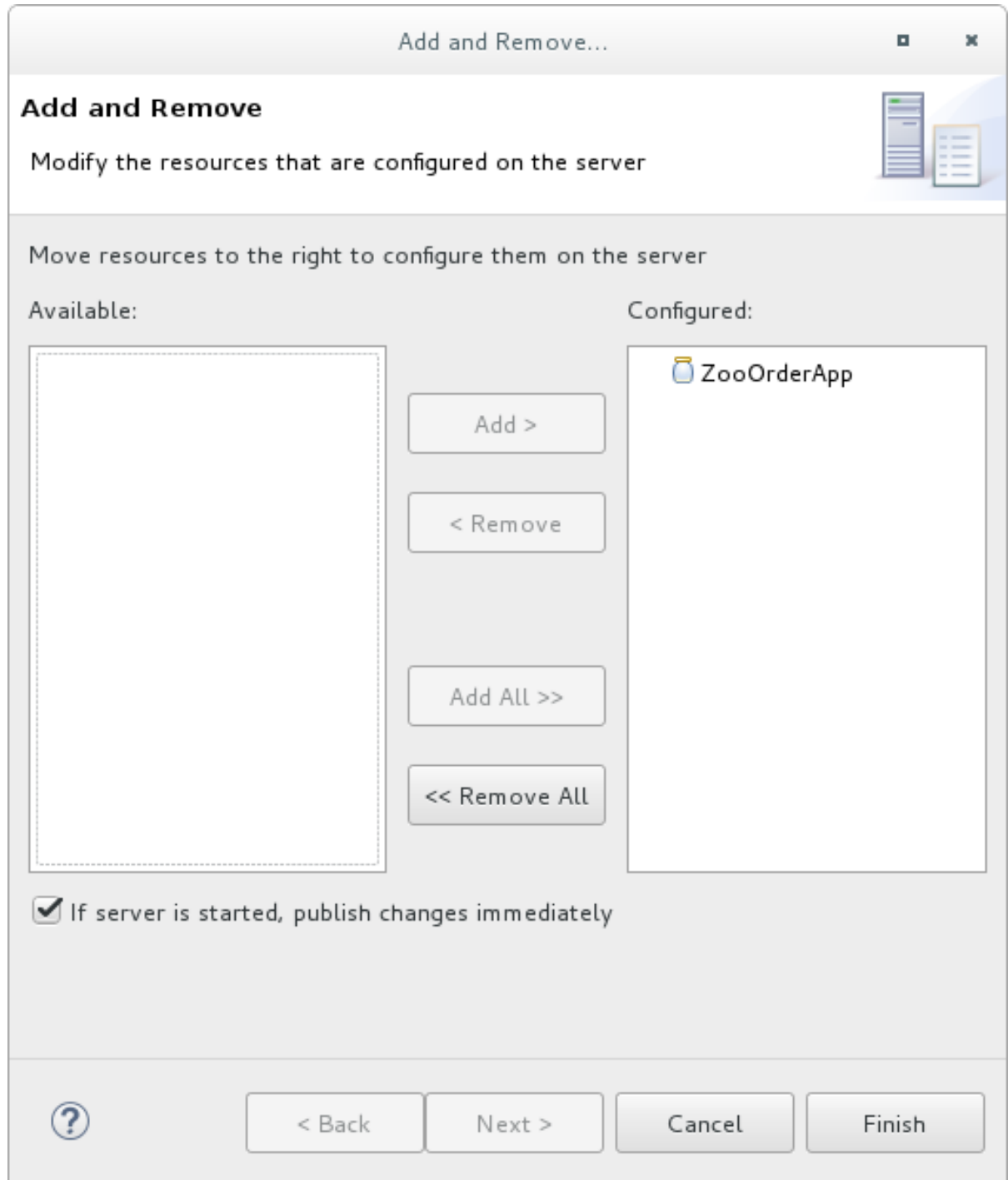


注記

公開されたリソースをアンインストールするために、JMX 接続を切断したり、サーバーを停止したりする必要はありません。

ランタイムサーバーから **ZooOrderApp** リソースを削除するには、以下を実行します。

1. **Servers** ビューで、**n.n Runtime Server** を右クリックしてコンテキストメニューを開きます。
2. **Add and Remove** を選択します:



3. **Configured** 列で **ZooOrderApp** を選択し、**Remove** をクリックして **ZooOrderApp** リソースを **Available** 列に移動します。
4. **Finish** をクリックします。

5. **Servers** ビューで、**JMX[Connected]** を右クリックし、**Refresh** をクリックします。
JMX[Connected] 下の **Camel** ツリーが消えます。



注記

JMX Navigator でも、**Server Connections > n.n Runtime Server[Connected]** の **Camel** ツリーが消失します。

6. **Properties** ビューに **Bundles** ページが表示されたら、リストの最後までスクロールして、**ZooOrderApp** のバンドルがリストされていないことを確認します。