JBoss Enterprise SOA Platform 5

ESB ツールリファレンスガイド

このガイドは開発者向けです。

Last Updated: 2023-03-28

JBoss Enterprise SOA Platform 5 ESB ツールリファレンスガイド

このガイドは開発者向けです。

Enter your first name here. Enter your surname here.
Enter your organisation's name here. Enter your organisational division here.
Enter your email address here.

法律上の通知

概要

このガイドは、JBDS の ESB Tools プラグインの使用方法を開発者に説明します。

目次

はじめに

1. ドキュメント規則

本ガイドでは、いくつかの規則を使用して特定の単語やフレーズを強調表示し、特定の情報への注意を促しています。

1.1. 表記規則

特定の単語や句への注意を促すために 4 つの表記慣習を使用しています。これらの規則や、これらが適用される状況は以下のとおりです。

等幅ボールド

シェルコマンド、ファイル名、パスなど、システム入力を強調表示するために使用されます。キーとキーの組み合わせを強調表示するためにも使用されます。以下に例を示します。

> 現在の作業ディレクトリーのファイル **my_next_bestselling_novel** の内容を表示するには、シェルプロンプトで **cat my_next_bestselling_novel** コマンドを入力し、**Enter** を押してコマンドを実行します。

上記には、ファイル名、シェルコマンドおよびキーが含まれます。これはすべて等幅ボールドで表示され、コンテキストにより区別可能なものになります。

キーの組み合わせは、各部分をつなぐプラス記号によって、個別のキーと区別できます。以下に例を示します。

> **Enter** を押してコマンドを実行します。
>
> **Ctrl**+**Alt**+**F2** を押して、仮想ターミナルに切り替えます。

最初の例では、押す特定のキーを強調表示しています。2 つ目の例は、同時に押す 3 つのキーのセットというキーの組み合わせを強調表示しています。

ソースコードの場合、段落内で記述されるクラス名、メソッド、関数、変数名、および戻り値は、上記のように **等幅ボールド** で示されます。以下に例を示します。

> ファイル関連のクラスには、ファイルシステムの **filesystem**、ファイルの **file**、ディレクトリーの **dir** が含まれます。各クラスには、独自の関連付けられたパーミッションセットがあります。

プロポーショナルボールド

これは、アプリケーション名、ダイアログボックステキスト、ラベルが付いたボタン、チェックボックスおよびラジオボタン、メニュータイトルおよびサブメニュータイトルなど、システムで発生した単語またはフレーズを示します。以下に例を示します。

> メインメニューバーから System → Preferences → Mouse を選択し、Mouse Preferences を起動します。**Buttons** タブで、**Left-handed mouse** チェックボックスを選択し、**Close** をクリックしてメインのマウスボタンを左から右に切り替えます (マウスを左手で使い易くします)。
>
> 特殊文字を gedit ファイルに挿入するには、メインメニューバーから Applications → Accessories → Character Map を選択します。次に、Character Map メニューバーから Search → Find... を選択し、**Search** フィールドに文字の名前を入力して **Next** をクリックします。目的の文字が **Character Table** で強調表示されます。この強調表示し

た文字をダブルクリックして **Text to copy** フィールドに配置し、**Copy** ボタンをクリックします。ここでドキュメントに戻り、**gedit** メニューバーから **Edit → Paste** を選択します。

上記のテキストにはアプリケーション名、システム全体のメニュー名および項目、アプリケーション固有のメニュー名、GUI インターフェイス内のボタンおよびテキストなどがあります。すべては proportional bold で示され、コンテキストと区別できます。

等幅ボールドイタリック または ***プロポーショナルボールドイタリック***

等幅ボールドまたはプロポーショナルボールドのいずれでも、イタリックが追加されると、置換または変数テキストを意味します。イタリックは、状況に応じて変化するテキストや、文字を入力しないテキストを表します。以下に例を示します。

ssh を使用してリモートマシンに接続するには、シェルプロンプトで **ssh *username@domain.name*** を入力します。リモートマシンが **example.com** で、そのマシン上でのユーザー名が john の場合は、**ssh john@example.com** と入力します。

mount -o remount *file-system* コマンドにより、指定したファイルシステムが再マウントされます。たとえば、**/home** ファイルシステムを再マウントする場合、コマンドは **mount -o remount /home** となります。

現在インストールされているパッケージのバージョンを表示するには、**rpm -q *package*** コマンドを使用します。これにより、***package-version-release*** のような結果が返されます。

上記の太字のイタリック体の用語、username、domain.name、file-system、package、version、および release に注意してください。各単語はプレースホルダーで、コマンドの発行時に入力するテキストまたはシステムによって表示されるテキストのどちらかになります。

作業のタイトルを示す標準的な使用法のほかに、イタリックは新用語と重要な用語の最初の使用を示します。以下に例を示します。

Publican は *DocBook* 公開システムです。

1.2. 引用規則

端末の出力およびソースコードの一覧は、周りのテキストから視覚的に表示されます。

ターミナルに送信される出力は **mono-spaced roman** に設定され、以下のように表示されます。

```
books       Desktop   documentation  drafts  mss    photos   stuff  svn
books_tests  Desktop1  downloads      images  notes  scripts  svgs
```

ソースコードの一覧も **mono-spaced roman** に設定されますが、以下のように構文の強調表示が追加されます。

```
static int kvm_vm_ioctl_deassign_device(struct kvm *kvm,
                 struct kvm_assigned_pci_dev *assigned_dev)
{
        int r = 0;
        struct kvm_assigned_dev_kernel *match;

        mutex_lock(&kvm->lock);

        match = kvm_find_assigned_dev(&kvm->arch.assigned_dev_head,
```

```
                                assigned_dev->assigned_dev_id);
        if (!match) {
                printk(KERN_INFO "%s: device hasn't been assigned before, "
                  "so cannot be deassigned\n", __func__);
                r = -EINVAL;
                goto out;
        }

        kvm_deassign_device(kvm, match);

        kvm_free_assigned_device(kvm, match);

out:
        mutex_unlock(&kvm->lock);
        return r;
}
```

1.3. 注記および警告

最後に、3 つの視覚的スタイルを使用して、見落とす可能性のある情報に注意を促します。

注記

注記とは、タスクへのヒント、ショートカット、または代替アプローチです。注意を無視しても悪い結果を招くことはありませんが、便利なヒントを見逃してしまう可能性があります。

重要

見落としやすい詳細のある重要なボックス: 現行セッションにのみ適用される設定変更や、更新を適用する前に再起動が必要なサービスなどです。Important というラベルが付いたボックスを無視しても、データが失われることはありませんが、スムーズな操作が行えないことがあります。

警告

警告は無視すべきではありません。警告を無視すると、データが失われる可能性があります。

2. ヘルプの利用とフィードバック提供

2.1. ヘルプが必要ですか？

本ガイドで説明されている手順で問題が発生した場合は、Red Hat カスタマーポータル http://access.redhat.com にアクセスしてください。カスタマーポータルでは、以下を行うことができます。

- Red Hat 製品に関する技術サポート記事の知識ベースの検索または閲覧。

- Red Hat グローバルサポートサービス (GSS) へのサポートケースの送信。
- その他の製品ドキュメントへのアクセス。

Red Hat は、Red Hat のソフトウェアおよびテクノロジーについて、多くの電子メーリングリストも提供しています。一般に公開されているメーリングリストの一覧は、https://www.redhat.com/mailman/listinfoを参照してください。メーリングリストの名前をクリックして、その一覧をサブスクライブするか、またはメーリングリストのアーカイブにアクセスします。

2.2. ご意見をお寄せください

本ガイドで誤字脱字を発見されたり、このマニュアルを改善するための提案をお持ちの場合は、弊社までご連絡ください。JBoss Enterprise SOA Platform の製品に対して、http://bugzilla.redhat.com/ から Bugzilla レポートを送信してください。

バグレポートを送信する際には、マニュアル識別子である 『ESB_Tools_Reference_Guide』 を指定してください。

本ガイドを改善するためのご意見やご提案をお寄せいただく場合は、できるだけ具体的にご説明ください。エラーが見つかった場合は、簡単に確認できるように、セクション番号と前後のテキストを含めてください。

第1章 はじめに

1.1. ビジネス統合

動的かつアジャイルなビジネスインフラストラクチャーを提供するためには、異なるアプリケーションとデータソースが最小限のオーバーヘッドで相互に通信できるように、サービス指向のアーキテクチャーを用意することが重要です。

JBoss Enterprise SOA Platform は、ビジネスプロセスの変更に対応するために、継続的に再利用することなく、ビジネスサービスをオーケストレーションできるフレームワークです。JBoss Enterprise SOA Platform では、ビジネスルールとメッセージの変換およびルーティング機能を使用することで、複数のソースからビジネスデータを操作できます。

バグの報告

1.2. サービス指向アーキテクチャーとは

はじめに

SOA (*Service Oriented Architecture*) は単一のプログラムまたはテクノロジーではありません。ソフトウェア設計パラダイムと見なします。

すでに分かっているように、*ハードウェアバス* は、複数のシステムとサブシステムを結び付ける物理コネクターです。これを使用する場合は、システムのペア間で多数のポイントツーポイントコネクターを使用する代わりに、各システムを中央バスに接続するだけです。*エンタープライズサービスバス* (ESB) は、ソフトウェアとまったく同じことを行います。

アーキテクトは、メッセージングシステムの上記のアーキテクチャー層にあります。このメッセージングシステムは、このメッセージングシステムを使用することでサービス間の *非同期通信* を容易にします。実際、アーキテクトを使用している場合、すべてを概念的に、*サービス*（このコンテキストではアプリケーションソフトウェア）またはサービス間で送信される *メッセージ* のいずれかです。サービスは接続アドレスとして一覧表示されます (*エンドポイント参照* と呼ばれています)。

このコンテキストでは、サービスは必ずしも Web サービスであるとは限らないことに注意することが重要です。File Transfer Protocol や Java Message Service などのトランスポートを使用する他のタイプのアプリケーションもサービスになる可能性があります。

注記

この時点で、エンタープライズサービスバスがサービス指向のアーキテクチャーと同じ場合は、妨げられる場合があります。回答は Not exactly です。 アーキテクトは、サービス指向のアーキテクチャーを形成しません。代わりに、ツールを多数使用して構築できます。特に、SOA が必要とする *loose-coupling* および *非同期メッセージを渡すこと* が容易になります。SOA は常にソフトウェア以外のものと考えてください。これは一連の原則、パターン、およびベストプラクティスです。

バグの報告

1.3. サービス指向アーキテクチャーの重要なポイント

以下は、サービス指向のアーキテクチャーの主要なコンポーネントです。

1. 交換される *メッセージ*
2. サービスリクエスターおよびプロバイダーとして動作する *エージェント*
3. メッセージを送受信できるようにする *共有トランスポートメカニズム*。

バグの報告

1.4. JBOSS ENTERPRISE SOA PLATFORM とは

JBoss Enterprise SOA Platform は、エンタープライズアプリケーションインテグレーション (EAI) およびサービス指向アーキテクチャー (SOA) ソリューションを開発するためのフレームワークです。これは、エンタープライズサービスバス (JBoss Warehouse) およびビジネスプロセス自動化インフラストラクチャーで設定されます。これにより、ビジネスサービスの構築、デプロイ、統合、オーケストレーションを行うことができます。

バグの報告

1.5. SERVICE-ORIENTED ARCHITECTURE PARADIGM

SOA (Service-ient Architecture)は、リクエスター、プロバイダー、およびブローカーの 3 つのロールで設定されます。

サービスプロバイダー

サービスプロバイダーはサービスへのアクセスを許可し、サービスの説明を作成し、サービスブローカーに公開します。

サービスリクエスター

サービスリクエスターは、サービスブローカーが提供するサービスの説明を検索して、サービスを検出します。リクエスターはサービスプロバイダーが提供するサービスにバインドするロールも果たします。

サービスブローカー

サービスブローカーは、サービスの説明のレジストリーをホストします。リクエスターをサービスプロバイダーにリンクします。

バグの報告

1.6. コアおよびコンポーネント

JBoss Enterprise SOA Platform は、データ統合のニーズに対応する包括的なサーバーを提供します。基本的なレベルでは、Enterprise Service Bus を介してビジネスルールを更新し、メッセージをルーティングすることができます。

JBoss Enterprise SOA Platform の中心となるのは、Enterprise Service Bus です。JBoss (ESB) はメッセージの送受信を行う環境を作成します。メッセージにアクションを適用して変換し、サービス間でルーティングすることができます。

JBoss Enterprise SOA Platform を設定するコンポーネントは複数あります。ESB とともに、レジストリ (jUDDI)、変換エンジン (Smooks)、メッセージキュー (HornetQ)、BPEL エンジン (Riftsaw) が用意されています。

バグの報告

1.7. JBOSS ENTERPRISE SOA PLATFORM のコンポーネント

- 完全な Java EE 準拠のアプリケーションサーバー (JBoss Enterprise Application Platform)
- Enterprise Service Bus (JBoss ESB)
- ビジネスプロセス管理システム (jBPM)
- ビジネスルールエンジン (JBoss ルール)
- オプションの JBoss Enterprise Data Services (EDS) 製品のサポート。

バグの報告

1.8. JBOSS ENTERPRISE SOA PLATFORM の機能

JBoss Enterprise Service Bus (ESB)

ドメインはサービス間でメッセージを送信し、それらを変換して、異なるタイプのシステムで処理できるようにします。

Business Process Execution Language (BPEL)

Web サービスを使用して、この言語を使用してビジネスルールをオーケストレーションできます。これは、ビジネスプロセス命令を簡単に実行するために SOA に含まれています。

Java Universal Description、Discovery and Integration (jUDDI)

これは SOA のデフォルトサービスレジストリーです。ここで説明する内容は、インストラクター上のサービスに関する情報をすべて保管する場所です。

Smooks

この変換エンジンは SOA と組み合わせて使用してメッセージを処理できます。また、メッセージを分割して正しい宛先に送信するためにも使用できます。

JBoss ルール

これは、SOA にパッケージ化されたルールエンジンです。受信するメッセージからデータを推測して、実行する必要があるアクションを判別できます。

バグの報告

1.9. JBOSS ENTERPRISE SOA PLATFORM の JBOSSESB コンポーネントの機能

JBoss Enterprise SOA Platform の JBossESB コンポーネントは以下をサポートします。

- 複数のトランスポートおよびプロトコル
- リスナーアクションモデル (これにより、サービスを相互に選択可能)
- コンテンツベースのルーティング (JBoss Rules エンジン、XPath、Regex、および Smooks 経由)
- サービスオーケストレーション機能を提供するために JBoss Business Process Manager (jBPM) との統合
- ビジネスルールの開発機能を提供するために、JBoss ルールとの統合
- BPEL エンジンとの統合。

さらに、新しいデプロイメントにレガシーシステムを統合し、同期または非同期で通信できるようにします。

さらに、エンタープライズサービスバスは、以下を可能にするインフラストラクチャーおよびツールセットを提供します。

- さまざまなトランスポートメカニズム (電子メールや JMS など) と連携するよう設定されます。
- 汎用オブジェクトリポジトリーとして使用します。
- プラグ可能なデータ変換メカニズムを実装できるようにします。
- 対話のロギングをサポートします。

重要

ソースコードには、**org.jboss.internal.soa.esb** と **org.jboss.soa.esb** の 2 つのツリーがあります。**org.jboss.internal.soa.esb** パッケージの内容をそのまま使用します。これは、通知なしに変更される可能性があるためです。これとは対照的に、**org.jboss.soa.esb** パッケージ内のすべての内容は、Red Hat の非推奨ポリシーでカバーされます。

バグの報告

1.10. タスク管理

JBoss SOA は、影響を受けるすべてのシステムで汎用的に実行するタスクを指定することにより、タスクを簡素化します。つまり、ユーザーは各ターミナルで個別に実行するようにタスクを設定する必要はありません。ユーザーは、Web サービスを使用してシステムを簡単に接続できます。

JBoss SOA を使用して各マシンに複数回ではなく、ネットワーク全体でトランザクションを 1 回委譲することで、時間を節約できます。これにより、エラーが発生する可能性も低くなります。

バグの報告

1.11. 統合のユースケース

ACME Equity は、大規模な経理サービスです。会社には多くのデータベースやシステムがあります。旧式の COBOL ベースのレガシーシステムや、近年で小規模な企業で取得されるデータベースもあります。ビジネスルールが頻繁に変化するため、これらのデータベースを統合することは困難でコストがかかります。会社は、クライアント向け e-commerce の Web サイトを新たに開発することを希望していますが、現在使用している既存のシステムとは同期しない場合があります。

会社は、安価なソリューションを希望していますが、企業セクターの厳密な規制およびセキュリティー要件に準拠するソリューションを検討しています。会社が望ましくないことは、レガシーデータベースおよびシステムを接続するために glob コードを作成および維持する必要があることです。

JBoss Enterprise SOA Platform は、これらのレガシーシステムを新しいお客様の Web サイトに統合するためにミドルウェアレイヤーとして選択されました。フロントエンドシステムとバックエンドシステムの間のブリッジを提供します。JBoss Enterprise SOA Platform で実装されたビジネスルールは、すぐに簡単に更新できます。

その結果、SOA の統合方法により、古いシステムは新しいシステムと同期できるようになりました。1 カ月あたり数万のトランザクションであっても、ボトルネックはありません。XML、JMS、FTP などのさまざまな統合タイプは、システム間でデータを移動するために使用されます。数多くのエンタープライズ標準のメッセージングシステムの 1 つを JBoss Enterprise SOA Platform にプラグインすることで、柔軟性を高めることができます。

さらに利点は、既存のインフラストラクチャーにより多くのサーバーやデータベースが追加されると、システムを簡単にスケールアップできることです。

バグの報告

1.12. ビジネス環境での JBOSS ENTERPRISE SOA PLATFORM の使用

エラーメッセージが発生する可能性が低いサービスの実装により、コストを削減できます。生産性とソーシングオプションの強化により、継続的なコストを削減できます。

情報およびビジネスプロセスは、接続が増加するため、迅速に共有できます。これは Web サービスによって強化され、クライアントを簡単に接続するために使用できます。

レガシーシステムは Web サービスと組み合わせて使用して、異なるシステムが同じ言語にピークにすることができます。これにより、システムの同期に必要なアップグレードおよびカスタムコードの量が減ります。

バグの報告

第2章 はじめに

2.1. ESB ツールリファレンスガイドの対象読者

はじめに

本書は、タスクの実行と JBoss Enterprise Service Bus の機能の使用について学びたい開発者を対象としています。

バグの報告

2.2. ESB ツールリファレンスガイドの目的

本書は、JBoss ESB プラグインが JBoss ESB と連携するために提供する機能についてユーザーに説明することを目的としています。読者は、ESB プロジェクトの作成、ESB エディターの使用などの基本的なタスクの実行方法、および機能を利用してより効率的なワークフローを作成する方法を学習します。

バグの報告

2.3. データのバックアップ

警告

Red Hat は、本ガイドに記載されている設定タスクを行う前に、システム設定とデータのバックアップを行うことを推奨します。

バグの報告

第3章 タスク

3.1. SOA プラットフォームの使用と設定

1. JBoss Enterprise SOA Platform を設定するには、**Window → Preferences → Server → Runtime Environments** を選択します。これにより、サーバーランタイム環境を追加、削除、および編集できるサーバーランタイム環境ページが開きます。

2. **Add** を選択し、ランタイム環境のタイプとして **JBoss Enterprise Middleware → JBoss Enterprise Application Platform 5.x Runtime** を選択し、**Create a new local server** ボックスをチェックして **Next** をクリックします。

3. 次のステップでは、サーバーランタイム環境の名前を指定し、その場所を参照できます。**Finish** をクリックして、サーバーランタイム環境を追加します。

4. これで、SOA プラットフォームが設定されました。ESB プロジェクトを作成して実行し、設定を確認します。

バグの報告

3.2. JBOSS SOA ツールプラグインのインストール

手順3.1 タスク

1. JBoss Developer スタジオを起動します。

2. メインメニューから **Get started with JBoss Central** を選択します。

3. メニューから **Software/Update** を選択します。

4. **SOA Tooling Plug-in** を選択し、**Install** をクリックします。

5. インストールの詳細が正しいことを確認し、**Next** へをクリックしてから、もう一度 **Next** をクリックします。

6. 使用許諾契約を読んで同意し、**Finished** をクリックします。

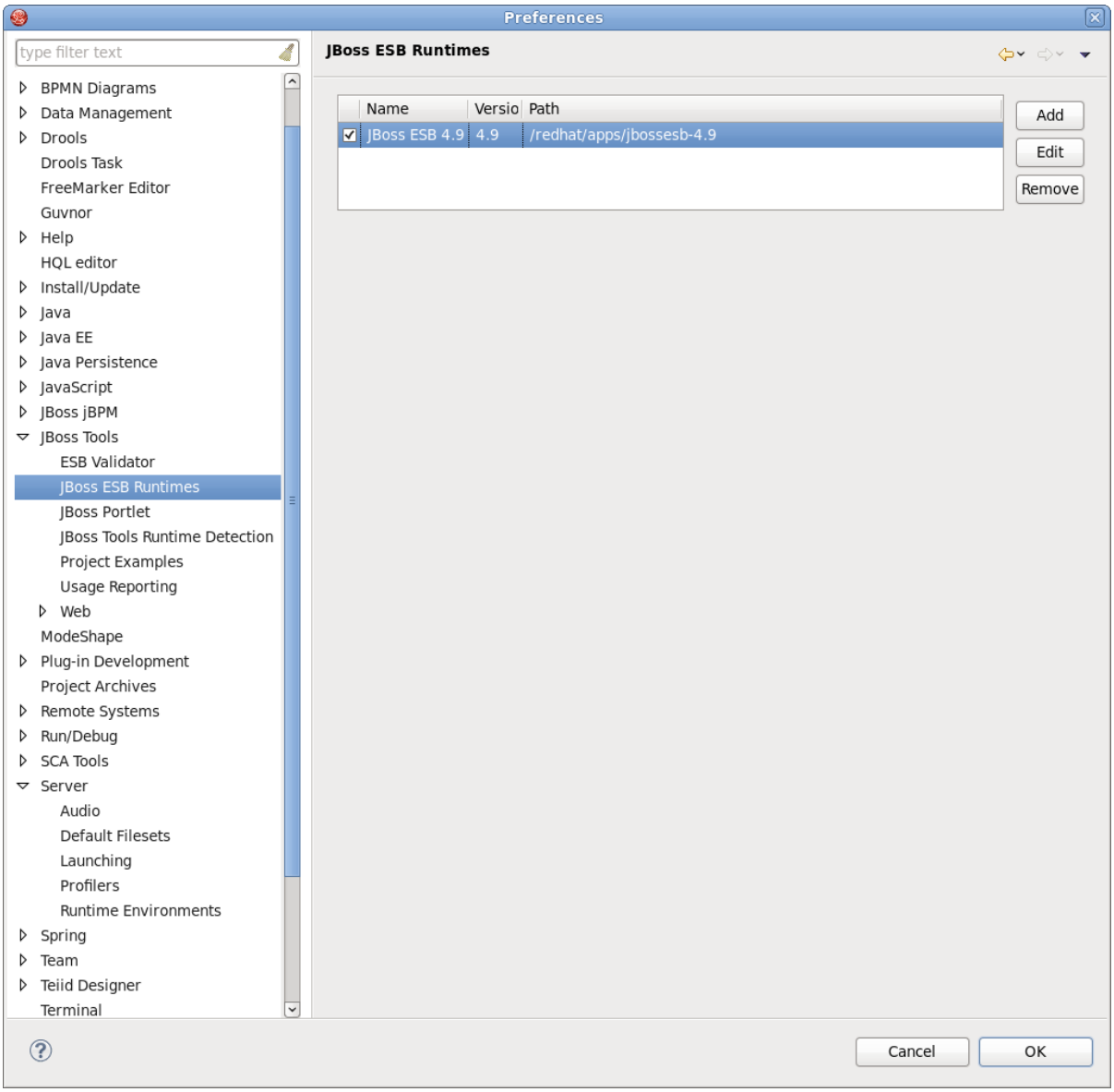
7. JBoss Developer Studio を再起動して、インストールを完了します。

バグの報告

3.3. 新しい ESB ランタイムの設定

1. **Window → Preferences → JBoss Tools → JBoss ESB Runtimes** をクリックして、JBoss ESB ランタイムを追加、削除、編集できる JBoss ESB ランタイムページを開きます。

図3.1 ランタイム



2. **Add** を選択して、JBoss ESB ランタイムの場所、名前、およびバージョン番号を指定できるダイアログボックスを開きます。ホームディレクトリーを JBoss AS または SOA-P として指定すると、設定を定義できます。スタンドアロン ESB ランタイムの場所を選択すると、設定は空になり、無視できます。**Customize JBoss ESB Runtime jars** ボックスをチェックして、ランタイムのライブラリーをカスタマイズすることもできます。

3. 新しい JBoss ESB ランタイムが設定されます。**Finish** をクリックして設定を終了し、保存します。JBoss ESB プロジェクトの作成時に設定を使用できます。

4. ESB ランタイムがプロジェクト用に設定されている場合、ESB ランタイムのクラスパスコンテナーページを使用して変更できます。これを行うには、Package Explorer ビューに移動し、JBoss ESB Runtime ライブラリーを右クリックします。

5. **Properties** を選択すると、利用可能なすべての JBoss ESB ランタイムを一覧表示するテーブルが表示されます。

6. ESB プロジェクトに設定するランタイムの 1 つを選択し、**OK** をクリックします。

7. ESB コンテナーでは、含まれている各 JAR のプロパティーダイアログを介して、ソースと JavaDoc の場所を設定できます。任意の JAR ファイルを右クリックし、**Properties** を選択します。

8. **Java Source Attachment** を選択し、選択した JAR ファイルの新しいソースを含む場所 (フォルダー、JAR、または zip) を選択します。これは、推奨されるオプション (ワークスペース、外部フォルダーまたはファイル) のいずれかを使用して行うことができます。パスを手動で入力することもできます。

9. **Apply** をクリックしてから **OK** をクリックします。

10. Javadoc の場所を変更するには、**Javadoc Location** を選択し、Javadoc によって生成されたドキュメントへの URL を指定します。Javadoc の場所には、**package-list** というファイルが含まれます。

11. **Apply** をクリックしてから **OK** をクリックします。

バグの報告

3.4. ESB プロジェクトの作成

1. メインメニューバーで **File → New → Project...** を選択して JBoss ESB Project Creation ウィザードを開きます。(サーバーランタイムと ESB ランタイムが JBDS で定義されていることを確認してください。)

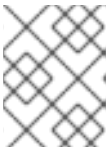
2. ダイアログボックスで **ESB → ESB Project** を選択します。

3. **Next** をクリックすると、プロジェクト名、ESB バージョン、およびターゲット JBoss ランタイムを指定する JBoss ESB プロジェクトウィザードページに移動します。たとえば、次のように指定します。 helloworld をプロジェクト名として入力し、デフォルトの JBoss ESB バージョンを受け入れます。

4. **Next** ボタンをクリックしてウィザードの次のステップに進み、ソースフォルダーと出力フォルダーを選択します。

5. **Next** ボタンをクリックして、ESB コンテンツディレクトリーを表示できる ESB ファセットのインストールウィザードページを表示します。このフォルダーには、ESB アーカイブに必要なほとんどのアーティファクトが含まれています。

6. **Server Supplied ESB Runtime** を使用して、ESB ライブラリーをプロジェクトに設定できます。プロジェクトにターゲットランタイムセットがあり、このランタイムに ESB ランタイムがインストールされていることを確認してください。



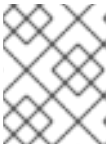
注記

JBoss SOA Platform 5.3 以降を使用している場合は、ESB 設定バージョン 1.3.1 を選択します。

7. **Finish** ボタンをクリックすると、デフォルトの **jboss-esb.xml** ファイルを含む ESB プロジェクトが作成されます。

バグの報告

3.5. ESB プロジェクトの使用例

1. 

   注記

   ESB プロジェクトの例を作成すると、完成したプロジェクトがどのように動作するかを観察するのに役立ちます。

   ESB プロジェクト例を作成する前に、 JBoss SOA-P ランタイムを作成します。

2. メインメニューバーで **File → New → Other...** を選択し、新しいダイアログで **Examples → JBoss Tools → Project Examples** をクリックします。

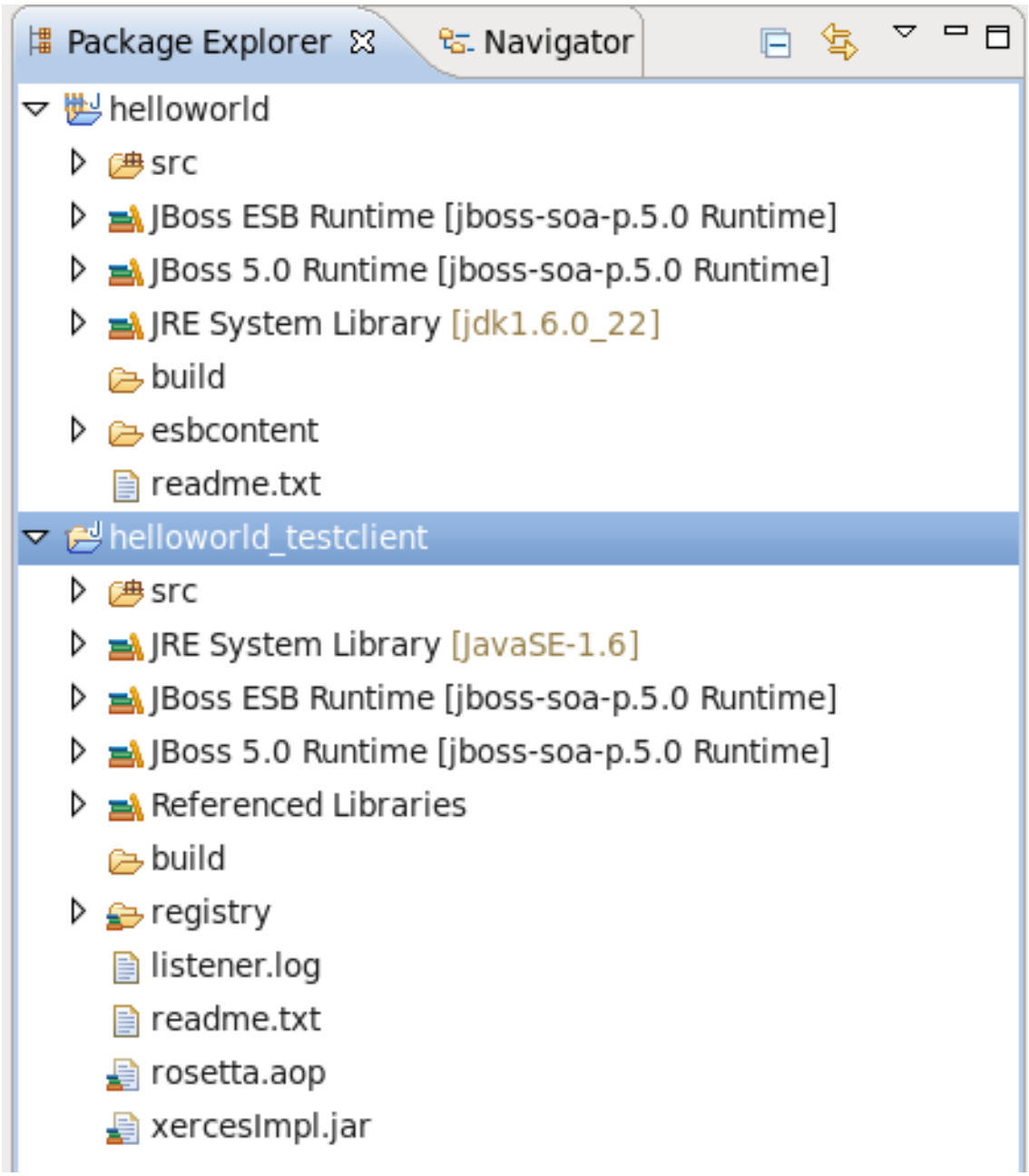
3. **Next** ボタンをクリックすると、リストから ESB プロジェクトの例を選択できるウィザードページが表示されます。

4. すべての ESB サンプルには 2 つのプロジェクトがあります。1 つは ESB プロジェクトで、もう 1 つは ESB プロジェクトのテストに使用される Java プロジェクトです。以下は、すぐに使える例のリストです。各例には、SOA-P サーバーにデプロイされる ESB アプリケーションと、Java アプリケーションとして実行され、デプロイされた ESB アプリケーションと対話するテストクライアントが含まれています。

5. - **JBoss ESB HelloWorld の例** - 基本的な ESB コンポーネントを実行し、ESB ランタイムが適切に設定されていることを証明するために必要な最小限のファイルを示します。
   - **JBoss ESB HelloWorld アクションの例** - 単一の設定から複数のアクション呼び出しを使用する方法を示します。1 つの Action クラスを使用して複数のメソッド呼び出しを行うか、複数の Action クラスを使用できます。
   - **JBoss ESB HelloWorld ファイルアクションの例** - JBoss ESB のファイルゲートウェイ機能の使用方法を示します。特定の拡張子を持つ特定のディレクトリーで見つかったファイルは、処理のためのアクションとともに JMS キューに送信されます。
   - **JBoss ESB Web サービスの consumer1 の例**- ESB アクションで JSR181 Web サービスを使用する方法を示します。
   - **JBoss ESB Web サービスプロデューサーの例** - SOAPProcessor アクションを使用して JBossESB に JSR181 Web サービスエンドポイントをデプロイする方法を示します。
   - **JBoss ESB Smooks CSV -> XML Example**- コンマ区切り値 (CSV) ファイルを XML に変換する方法を示します。
   - **JBoss ESB Smooks XML -> POJO Example**- XML ファイルを Java POJO に変換することで単純な変換を実行する Smooks の使用方法を示します。
   - **JBoss ESB Smooks XML -> XML date-manipulation Example**- JBoss ESB 内でメッセージ変換を手動で定義および適用する方法を示します。
   - **JBoss ESB Smooks XML -> XML Example**- JBoss ESB 内でメッセージ変換を手動で定義および適用する方法の非常に基本的な例。非常に単純な XSLT を SampleOrder.xml メッセージに適用し、前後の XML をコンソールに出力します。

6. 関連する ESB プロジェクトの例を選択し、**Finish** ボタンをクリックします。ESB アプリケーションプロジェクトとテストクライアントプロジェクトが作成されます。

図3.2 HelloWorld の例



7. ESB プロジェクトをデプロイし、テストクラスをローカル Java アプリケーションとして実行して、コンソールビューでテスト結果を確認します。

バグの報告

3.6. JBOSS サーバーモジュールリストへの ESB プロジェクトの追加

1. ESB プロジェクトをデプロイする前に、**Window → Show View → Other → Server → Servers** を選択してサーバービューを開きます。

2. サーバービューに ESB ランタイムがインストールされた JBoss サーバーを作成し、起動します。

3. 新しい JBoss サーバーを右クリックし、**Add and Remove Projects** を選択して、デプロイする ESB プロジェクトを開いているダイアログに追加します。

4. **Finish** ボタンをクリックして、プロジェクトをサーバーに追加します。ESB プロジェクトを Project Explorer ビューからサーバーにドラッグすることもできます。

バグの報告

3.7. ESB プロジェクトのデプロイ

1. ESB プロジェクトを JBoss サーバーモジュールリストに追加した後、JBoss サーバーを右クリックし、**Publish** を選択して、サーバー上でプロジェクトを発行します。デプロイ結果はコンソールビューで確認できます。

2. **Run** および **Debug** オプションは ESB プロジェクトで機能し、ユーザーが指定したサーバーの (再) デプロイを引き起こします。

   フィンガータッチボタンをクリックして、サーバーを再起動せずにプロジェクトをすばやく再起動することもできます。

3. **エクスポート中**
   さらに、ESB プロジェクトを .esb アーカイブとしてデプロイできます。プロジェクトを右クリックし、**Export** を選択します。

   ESB → ESB File を選択し、**Next** をクリックします。

   ESB プロジェクトをファイルシステムにエクスポートします。宛先とターゲットランタイムを選択し、アーカイブに適切な設定を行います。最後に、**Finish** をクリックします。

4. プロジェクトが .esb アーカイブとしてデプロイされました。

バグの報告

3.8. ESB ファイルの作成

1. ESB ファセットが有効になっているプロジェクトを開きます。選択したプロジェクトのメインメニューバーまたはコンテキストメニューで **File → New → Other...** を選択し、新しいダイアログで **ESB → ESB File** をクリックします。

2. **Next** ボタンをクリックすると、フォルダー、名前、およびバージョンを指定するウィザードページが表示されます。たとえば、名前として **jboss-esb.xml** を選択し、選択したプロジェクトフォルダーとデフォルトバージョンを受け入れます。

   注記

   SOA-P バージョン 5.3 の場合、ESB バージョンは 4.11 で、ESB ファイルスキーマバージョンは 1.3.1 です。古い SOA-P サーバーリリースを使用している場合は、古い ESB ファイルスキーマバージョンもサポートされます。

3. デフォルトでは、選択したプロジェクトのフォルダーにファイルが作成されます。保存先のフォルダーを変更する場合は、**Browse** をクリックして別の保存先を選択します。

4. **Finish** をクリックしてファイルを生成します。

バグの報告

3.9. ESB アクションの作成

1. ESB または Javaプロジェクトを開きます。
2. メインメニューまたはコンテキストプロジェクトメニューから **File → New → Other** を選択します。次に、新しいダイアログで **ESB → ESB Action** をクリックします。
3. **Next** をクリックすると、 **New ESB Action** クラス名を指定するウィザードが開きます。パッケージを設定したり、インターフェイスを追加したりすることもできます。
4. JBoss ESB 4.9 では、アクションアノテーションの新しいセットが導入されました。アクションアノテーションの新しいセットは、インターフェイスと抽象クラスの実装の複雑さを隠し、ConfigTree タイプを処理することで、クリーンな ESB アクションの実装を簡単に作成できるようにします。これらのアノテーションを使用する場合は、タイプ **As annotated POJO** を確認してください。それ以外の場合は、タイプ **As AbstractActionPipelineProcessor implementation** を確認してください。

   アノテーションは次のとおりです。

   - @Process
   - @ConfigProperty
   - @Initialize
   - @Destroy
   - @BodyParam
   - @PropertyParam
   - @AttachmentParam
   - @OnSucess
   - @OnException
5. **Finish** をクリックして、 ESB Action クラスを作成します。このクラスは ESB Editor ウィザードで利用できるようになります。

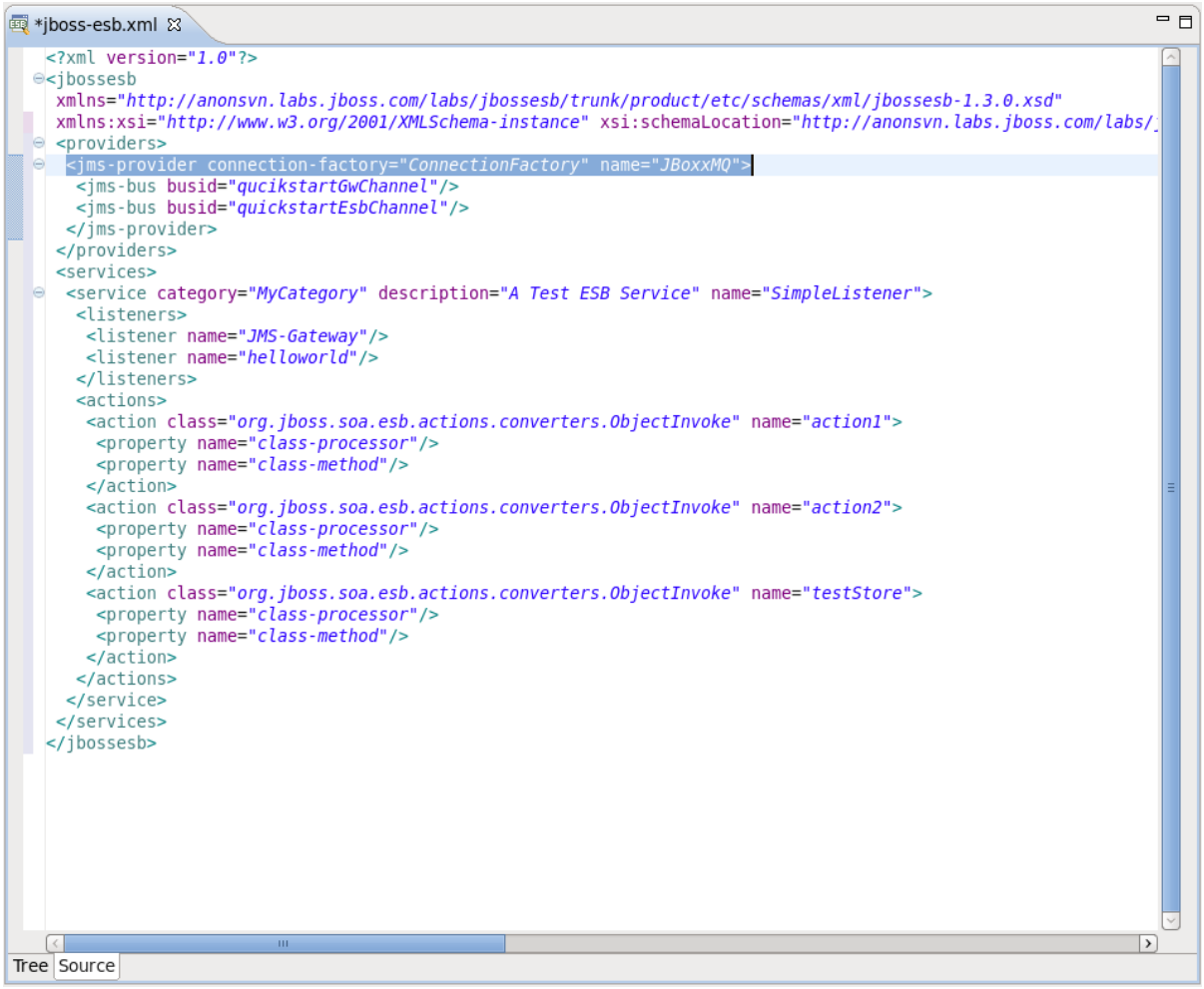
バグの報告

第4章 参照資料

4.1. JBOSS ESB エディター

1. JBoss ESB エディターには、ツリーとソースの 2 つのビューがあります。エディターの下部にある **Source** タブを選択してツリービューからソースビューに簡単に切り替え、ソースビューで作業することができます。

図4.1 ソースビュー

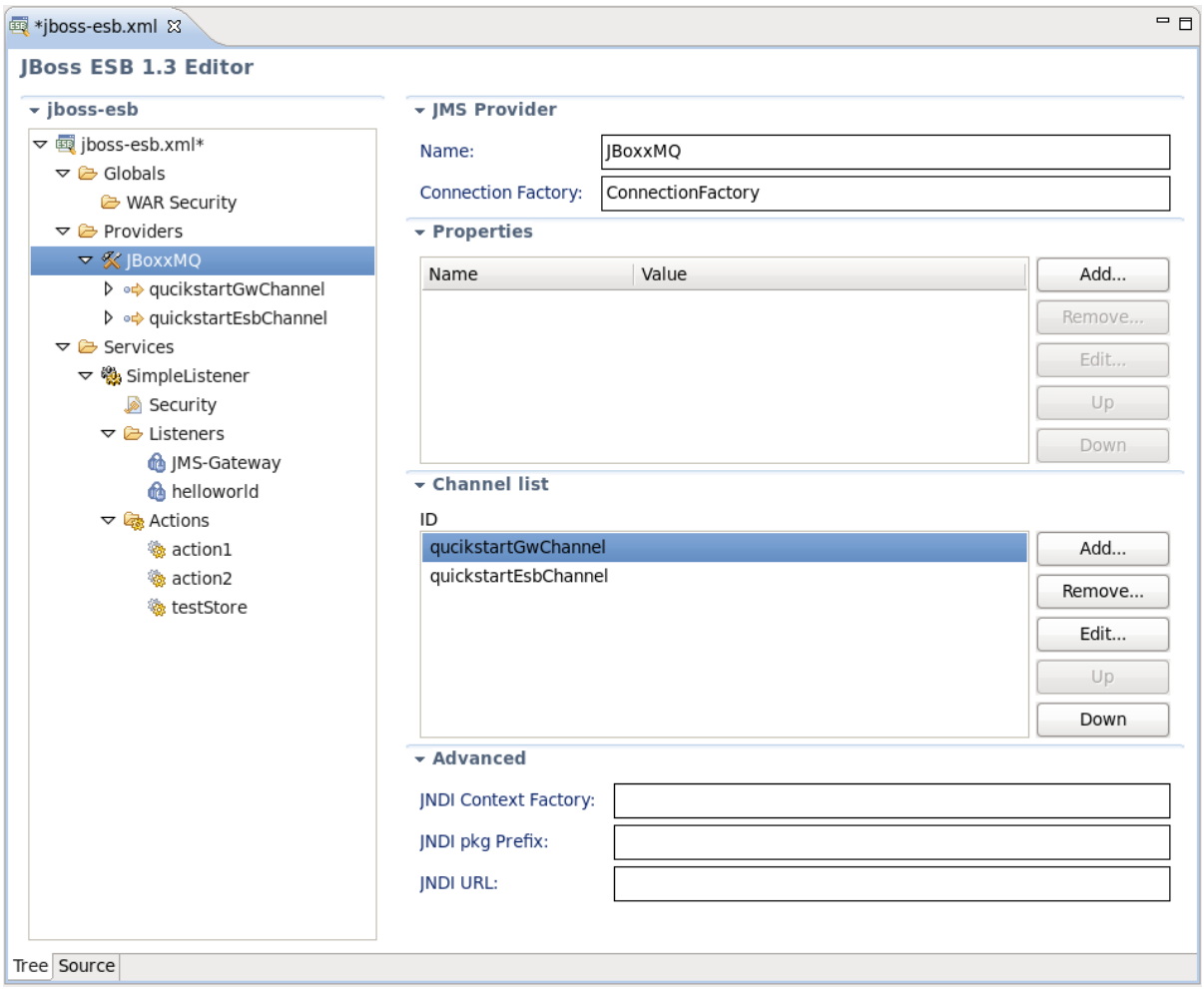


2. どちらのビューも、アウトラインビューと完全に統合されています。アウトラインビューには、ESB ファイルのツリー構造が表示されます。アウトラインビューで任意の要素を選択すると、ツリー/ソースエディターの同じ場所にジャンプするため、アウトラインビューでソースコードをナビゲートできます。

3. 両方のビューは互いに同期されているため、一方のビューで行われた変更は、すぐにもう一方のビューに反映されます。

4. エディターのツリービューには、すべての ESB アーティファクトがツリー形式で表示されます。任意のノードを選択すると、右側に表示されるそのプロパティーを表示および編集できます。一部のプロパティーは、関連するエディターへのリンクとして表されます。

図4.2 ツリー表示



5. 他のファイル (Drools、Groovy、Smooks など) を参照する ESB アクションを編集する場合、フィールドのラベルは、そのファイルの適切なエディターを起動するためのリンクに変わります。

6. 一部のアーティファクト操作の追加、編集、または削除は、ツリービューで直接実行できます。任意のノードを右クリックし、コンテキストメニューで使用可能なアクションの 1 つを選択します。

7. 同じ方法で、または右側にある 追加、編集、および 削除 ボタンのあるフォームを使用して、プロバイダーのチャネルとプロパティーを追加できます。

8. さらに、サービスとその新しいリスナーを追加できます。この方法でフィルターを追加することもできます。

バグの報告

4.2. JBOSS ESB エディターでの新しいカスタムアクションの追加

1. 新しいカスタムアクションを ESB XML ファイルに追加するには、JBoss ESB エディターのツリービューに移動します。

2. **Services** ノードの下の **Actions** ノードを選択し、右クリックして **New → Custom Action** を選択します。

   または、 editor の左側にある **Add...** ボタンをクリックします。

3. ESB アクションウィザードが表示されます。名前と Java クラスを指定します。必要に応じて、アクションのプロセスを指定できます。

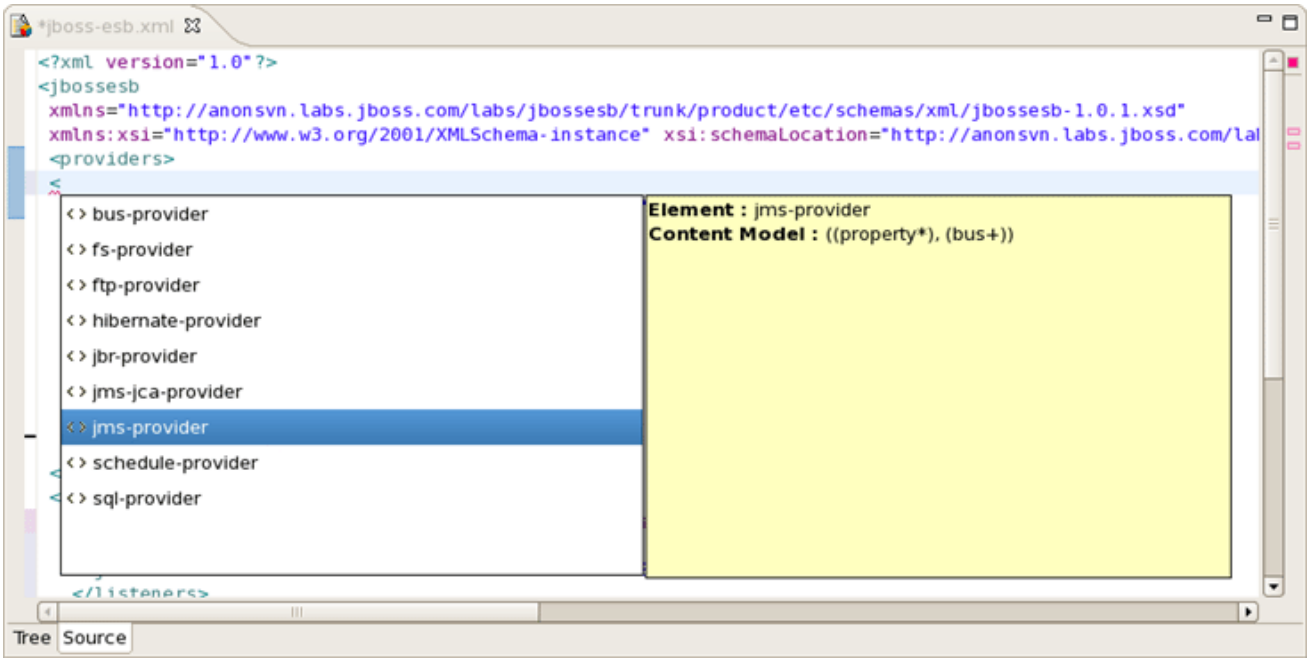
4. アクションを作成すると、**Actions** ノードの下のツリービューに表示されます。左側の Form editor でプレビューと設定の編集ができます。

バグの報告

4.3. ESB XML ファイルのコンテンツアシスト

ソースモードで ESB XML ファイルを操作する場合、 Content Assist は利用可能です。コードステートメントを完成させるのに役立つポップアップヒントが表示されます。利用可能なヒントを表示するには、**Ctrl-Space** を入力します。

図4.3 ESB XML ファイルのコンテンツアシスト

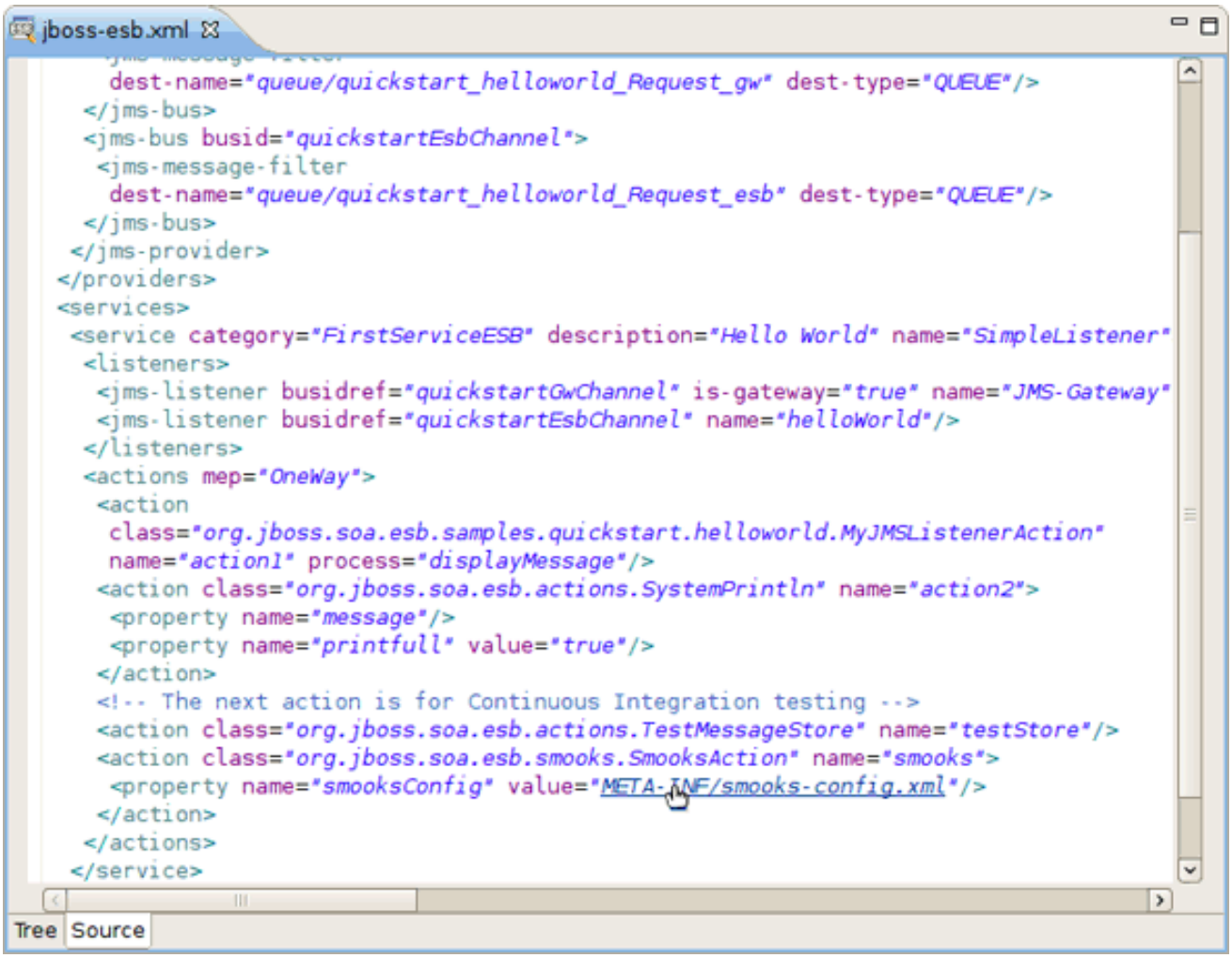


バグの報告

4.4. OPENON FOR ESB XML

ESB ファイルには、ファイル内の複数の参照を利用できる OpenOn 機能が付属しています。**Ctrl** キーを押したままクリックして表示します。

図4.4 OpenOn



OpenOn は、**.xsd .xml** などの拡張子を持つさまざまな種類のファイルに対して実装されます。

バグの報告

第5章 概要

5.1. ESB ツールと追加リソースのまとめ

本書は、JBoss ESB プラグインが JBoss ESB で動作するために提供する機能に関するすべての必要な情報を提供しました。

以下のリンクにアクセスすると、追加情報を見つけることができます。

- JBoss ESB
- JBoss Wiki
- JBoss ESB ドキュメンテーションライブラリー

最新の JBoss Tools/JBoss Developer Studio ドキュメンテーションビルドは、JBoss Tools ナイトリードキュメンテーションページ で入手できます。

バグの報告

付録A 更新履歴

改訂 5.3.1-111.400 2013-10-31 **Rüdiger Landmann**
Rebuild with publican 4.0.0

改訂 5.3.1-111 Wed Feb 20 2013 **CS Builder Robot**
コンテンツ仕様からビルド: 6854、リビジョン: 375405