HPE Operations Orchestration

アクション開発者ガイド

バージョン10.60

ドキュメントリリース日: 2016年5月 (英語版)
ソフトウェアリリース日: 2016年5月

Hewlett Packard Enterprise

ご注意

保証

HPE製品、またはサービスの保証は、当該製品、およびサービスに付随する明示的な保証文によってのみ規定されるものとします。ここでの記載は、追加保証を提供するものではありません。ここに含まれる技術的、編集上の誤り、または欠如について、HPEはいかなる責任も負いません。

ここに記載する情報は、予告なしに変更されることがあります。

権利の制限

機密性のあるコンピューターソフトウェアです。これらを所有、使用、または複製するには、HPEからの有効な使用許諾が必要です。商用コンピューターソフトウェア、コンピューターソフトウェアに関する文書類、および商用アイテムの技術データは、FAR12.211および12.212の規定に従い、ベンダーの標準商用ライセンスに基づいて米国政府に使用許諾が付与されます。

著作権について

商標について

Adobe™は、Adobe Systems Incorporated (アドビシステムズ社) の登録商標です。

Microsoft®およびWindows®は、米国におけるMicrosoft Corporationの登録商標です。UNIX®は、The Open Groupの登録商標です。

本製品には、'zlib' (汎用圧縮ライブラリ) のインタフェースが含まれています。'zlib': Copyright © 1995-2002 Jean-loup Gailly and Mark Adler.

ドキュメントの更新情報

このマニュアルの表紙には、以下の識別情報が記載されています。

- ソフトウェアバージョンの番号は、ソフトウェアのバージョンを示します。
- ドキュメントリリース日は、ドキュメントが更新されるたびに変更されます。
- ソフトウェアリリース日は、このバージョンのソフトウェアのリリース期日を表します。

更新状況、およびご使用のドキュメントが最新版かどうかは、次のサイトで確認できます。**http://h20230.www2.HPE.com/selfsolve/manuals**

このサイトを利用するには、HPE Passportへの登録とサインインが必要です。HPE Passport IDの登録は、次のWebサイトから行なうことができます。**http://h20229.www2.HPE.com/passport-registration.html**

または、HPE Passportのログインページの [**New users - please register**] リンクをクリックします。

適切な製品サポートサービスをお申し込みいただいたお客様は、更新版または最新版をご入手いただけます。詳細は、HPEの営業担当にお問い合わせください。

サポート

HPEソフトウェアサポートオンラインWebサイトを参照してください。https://softwaresupport.HPE.com/
このサイトでは、HPEのお客様窓口のほか、HPEソフトウェアが提供する製品、サービス、およびサポートに関する詳細情報をご覧いただけます。
HPEソフトウェアオンラインではセルフソルブ機能を提供しています。お客様のビジネスを管理するのに必要な対話型の技術サポートツールに、素早く効率的にアクセスできます。HPソフトウェアサポートのWebサイトでは、次のようなことができます。

- 関心のあるナレッジドキュメントの検索
- サポートケースの登録とエンハンスメント要求のトラッキング
- ソフトウェアパッチのダウンロード
- サポート契約の管理
- HPEサポート窓口の検索
- 利用可能なサービスに関する情報の閲覧
- 他のソフトウェアカスタマーとの意見交換
- ソフトウェアトレーニングの検索と登録

目次

HPE OO用の拡張機能の作成

このドキュメントは、JavaおよびNETの開発者がHPE Operations Orchestrationを拡張するためにアクションを開発する際のガイドラインです。

注: Javaまたは.NETに関する知識が必要です。

HPE Operations Orchestrationは、プログラムによって拡張することができます。これにより、サードパーティがHPE Operations Orchestrationに機能を追加して、フロー実行エンジンにコンテンツとして導入することが可能になります。

新しいコンテンツを導入するには、拡張機能を作成してHPE OO Centralにデプロイする必要があります。アクションはJavaまたは.NETで記述します。

HPE Operations Orchestration 10.xでは、拡張機能はプラグインと呼ばれます (以前のバージョンでは、拡張機能はIActionと呼ばれていました)。プラグインは、実行エンジン内で実行されるコードです。このコードによって、分離された独自のクラスパスを定義できます。クラスパスの分離によって、複数の異なるプラグインで競合する依存関係を使用できます。たとえば、プラグインAでは依存関係Xのバージョン1.0を使用し、プラグインBでは同じ依存関係Xの異なるバージョン2.0を使用することが可能になります。クラスパスの競合問題が発生しているかどうかに関係なく、両方のプラグインを同じフローで使用できるようになりました。

プラグインには、1つまたは複数のアクションと、必要なすべての依存関係への参照が含まれます。

HPE Operations Orchestration 10.xには、アクションを開発するための新しい@Actionインタフェースがあります。@Actionはクラス内のメソッドです。詳細については、@Actionの開発を参照してください。

プラグインはすべてJavaで実行されていますが、HPE Operations Orchestrationでは.NETアクションもサポートしています。.NETで記述されたアクションは、ラップしているJavaプラグインによって参照されます。詳細については、.NET拡張機能を参照してください。

注: HPE OO 9.xのIActionインタフェースは現在は非推奨となっています。新しいコンテンツを作成するユーザーは、IActionインタフェースを実装するのではなく、@Actionを記述してください。

@Actionの作成

@Actionは、Mavenプラグインとして作成することをお勧めします。

10.20以前のSDKを使用してプラグインを作成するには、Apache Maven 3.0.3～3.0.5を使用する必要があります。

SDK 10.60用のプラグインを作成するには、Apache Maven 3.2.1を使用する必要があります。

プラグインの作成

このセクションでは、プラグインの作成方法について説明します。

com.HP.oo.sdk:oo-plugin-archetypeというMavenアーキタイプを使用すると、プラグインとStudioプロジェクトのスケルトンを作成できます。

Mavenアーキタイプを使用したプラグイン作成の準備

Mavenのインストール

コンピューターにMavenをインストールすると、システムPATHにbinディレクトリが追加されます。これにより、ファイルシステムの任意の場所からmvnを実行できます。

ローカルMavenリポジトリの作成

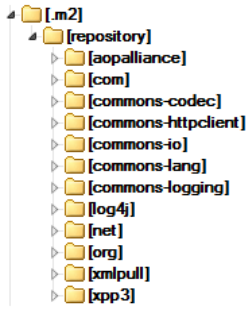
- sdk-dotnet-<バージョン>.zipとsdk-java-<バージョン>.zipを次の場所に展開します。

Windows: `%HOMEPATH%\.m2\repository`

Linux: `$HOME/.m2/repository`

注: これらのファイルはSDKフォルダーのHPE OO ZIPファイルにあります。

次に、ファイルを正しく展開した場合のディレクトリ構造の例を示します。



プラグインのアーキタイプの登録

- コマンドプロンプトを開いて次のコマンドを入力します。

```
mvn archetype:crawl
```

Mavenアーキタイプカタログが`$HOME/.m2/repository`で更新されます。

Mavenアーキタイプを使用したプラグインの作成

サンプルプロジェクトの作成

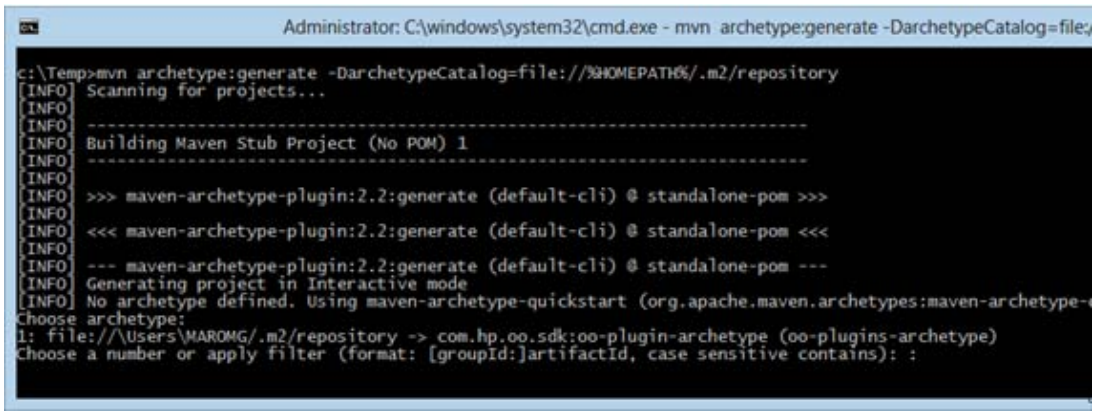
1. サンプルプラグインのプロジェクトを作成するパスに移動し、コマンドラインで次のコマンドを入力します。

```
mvn archetype:generate -DarchetypeCatalog=file://$HOME/.m2/repository
```

注: Windowsの場合、%HOMEPATH%を使用します。

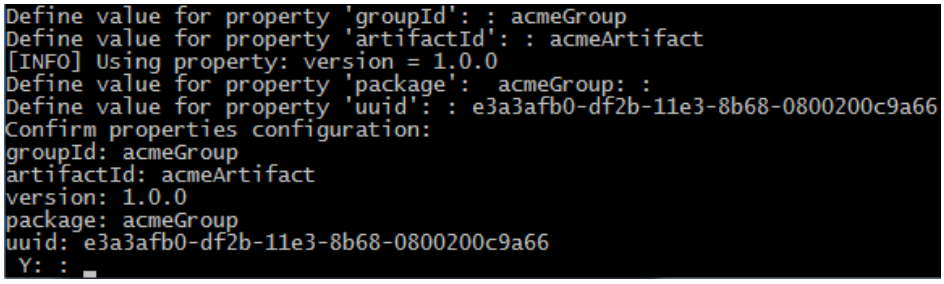
プロジェクトの作成が開始されます。カタログ内で見つかったアーキタイプのリストが表示されます。
com.hp.oo.sdk:oo-plugin-archetypeアーキタイプを表す番号を押し、Enterを押します。

以下の例では、**1**を押します。

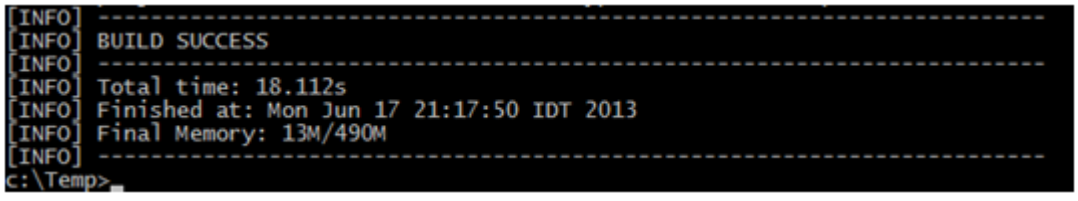


2. アーキタイプの作成中に、次の詳細を入力します。各項目の入力後にEnterを押します。

- groupId: 作成するMavenプロジェクトのグループID。下記の例ではacmeGroupを使用しています。
- artifactId: 作成するMavenプロジェクトのアーティファクトID。下記の例ではacmeArtifactを使用しています。
- package: プロジェクト内のファイルのパッケージ。このオプションのデフォルトはgroupIdと同じです。
- uuid: 生成されたプロジェクトのUUID。下記の例ではランダムに生成されたUUIDを使用しています。



ビルドが終了してプロジェクトが作成されます。



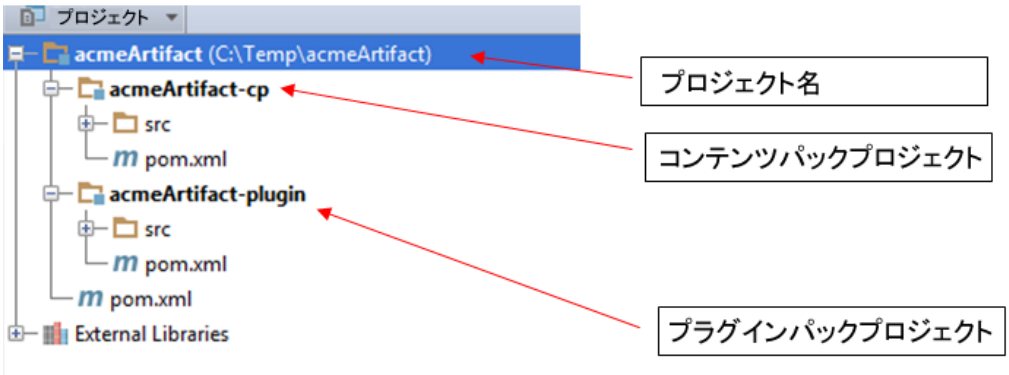
Java IDEでプロジェクトを開く

前のステップでMavenベースのモデルの新しいjavaプロジェクトが作成されました。

Java IDEアプリケーションでプロジェクトを開きます。

プロジェクトには、提供されたIDと同じプレフィックスを持つ2つのモジュールが含まれています。プロジェクトの1つはコンテンツパックプロジェクトであり、ほかのプロジェクトはプラグインプロジェクトです。

例:



親プロジェクト

図の例では、親プロジェクトは**acmeArtifact**と呼ばれます。

デフォルトでは、コンテンツパックとプラグインの2つのモジュールが含まれています。このプロジェクトは、@Action、およびその関連するオペレーションとフローを1つのコンテンツパックにグループ化するためのものです。

たとえば、Office統合を開発している場合、いくつかのプラグインプロジェクト (各Officeバージョンごとに1つずつ) を作成できます。ただし、オペレーションとフローを含む1つのコンテンツパックプロジェクトも可能で、これが推奨されるベストプラクティスです。

プラグインプロジェクト

この例では、プラグインプロジェクトは**acmeArtifact-plugin**と呼ばれます。

このモジュールには、@Actionsが含まれています。このプロジェクト (Mavenで) をビルドしている場合、内部のコードがコンパイルされ、結果のJARファイルをStudio内で開き、オペレーションを内部の@Actionから作成することができます。

このモジュール内には、サンプルの@Actionがあります。これを削除して、独自のものを作成できます。

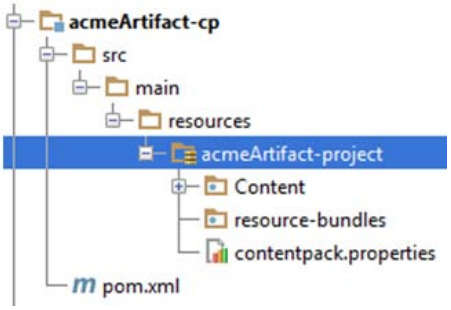
コンテンツパックプロジェクト

この例では、コンテンツパックプロジェクトは**acmeArtifact-cp**と呼ばれます。

このモジュールはコンテンツパックを表します。この中には、依存先の任意のプラグインモジュール (**acmeArtifact-pluginとその依存関係**など)、およびプロジェクト内で定義されているフロー、オペレーション、構成アイテムが含まれています。

Studioを使用したコンテンツパックモジュールの編集

コンテンツパックモジュールには、Studioで開き、編集できるStudioプロジェクトが含まれています。プロジェクトフォルダー (この例では**acmeArtifact-projectフォルダー**) をStudioにインポートすると、プロジェクトを編集できます。

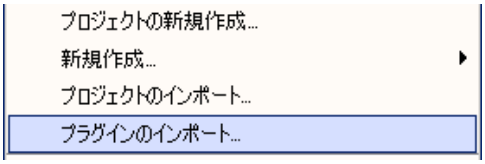


プロジェクト内でフロー、オペレーション、または構成アイテムを作成する場合、Mavenでこのプロジェクトをビルドできます。結果のJARファイルはコンテンツパックであり、Centralにデプロイするか、別のユーザーによってStudioで再度開くことができます。

プラグインモジュールからのプロジェクト内のオペレーションの作成

同じacmeArtifactプロジェクトの一部となっているプラグイン (たとえば、acmeArtifact-plugin) からオペレーションを作成する場合、次の手順を実行します。

1. **acmeArtifact-project**プロジェクトをStudioにインポートします。
2. Mavenを使用して、**acmeArtifact parent**モジュールをビルドします。
3. プラグインとその依存関係を**Studio**にインポートします。



4. OOプラグインとその依存関係のパスは、ローカルのMavenリポジトリ下のacmeArtifact-cp.jarファイルのパスとなります。
5. **Studio**で、新しいオペレーションを作成し、[**オペレーションの作成**] ダイアログボックスでプラグインを選択します。

@Actionの開発

@Actionはクラス内のメソッドであるため、任意のクラスの任意のメソッドにできます。このメソッドは拡張機能とも呼ばれます。

@Actionを使ったオペレーションが実行されると、その@Actionはフローの実行中に呼び出されます。

"Hello World!" 例

メソッドを@Actionとしてマークするには、`@com.HP.oo.sdk.content.annotations.Action`を使ってメソッドに注釈を設定します。以下に、単純な `"Hello World!"`@Action exampleの例を示します。

```
public class MyActions {

     @Action

     public void sayHello() {

         System.out.println("Hello World!");

     }

 }
```

デフォルトでは、作成した@Actionには、@Actionを定義するメソッドと同じ名前が付けられます。"Hello World!" の例では、@Actionの名前はsayHelloです。@Actionの名前はオペレーションの定義で使用されます。オペレーションは、@ActionをStudioとフロー作成者に公開するために使用されます。各オペレーションは特定のgroupId、artifactId、version、および@Action名（GAV+@Action名）を指します。

@Action名をカスタマイズして、メソッド名とは異なる名前を指定することもできます。これを行うには、@Actionの注釈値のパラメーターを使用します。以下のコードで、同じ "Hello World!"@Actionを定義しながら、my-hello-actionという名前を指定します。

```
public class MyActions {

     @Action(“my-hello-action”)

     public void sayHello() {

         System.out.println("Hello World!");

     }

 }
```

@Actionへの引数の引き渡し

@Actionはフローコンテキストに公開され、フローコンテキストのパラメーターを要求できます。フローコンテキストはフローの状態を保持しています。たとえば、2つの数字を加算して結果をコンソールに表示する、次のような@Actionがあるとします。

```
@Action

 public void sum(int x, int y){

     System.out.println(x+y);

 }
```

パラメーターは、コンテキストから名前で取得されます。sumメソッドは、コンテキストから2つの整数パラメーターxとyを要求します。@Actionが呼び出されると、HPE Operations Orchestrationはコンテキストから取得した値xとyを、同じ名前を持つメソッドの引数に割り当てます。

@Actionの場合と同じように、パラメーター名をカスタマイズし、カスタマイズした名前の使用時は値を解決するようにHPE Operations Orchestrationに要求できます。以下の例では、sumメソッドはコンテキストのop1パラメーターを引数xに、op2を引数yに割り当てるよう要求しています。

```
@Action

 public void sum(@Param("op1") int x, @Param("op2") int y){

     System.out.println(x+y);

 }
```

`com.HP.oo.sdk.content.constants`パッケージの下にある`ResponseNames`、`ReturnCodes`、`InputNames`、`OutputNames`の各クラスには、@Actionsで使用可能な一般的な定数が含まれています。たとえば、`HOST`、`USERNAME`、`PASSWORD`、`PORT`などの入力名や、`SUCCESS`、`FAILURE`、`NO_MORE`などのレスポンス名があります。

戻り値

Javaのメソッドと同様に、@Actionでも単一値を返すことができます。返された値は@Actionから返された結果と見なされ、オペレーションでreturn resultとして使用されます。@Actionでは、複数の結果をオペレーションに返すこともできます。これを行うには、Map<String, String> を返します。ここで、キーMapは結果の名前で、関連付けられる値は結果の値です。Map<String, String> を返すことで、@Actionは実行時に複数の出力をオペレーションに渡すことができます。

@Action注釈の追加

@Action注釈は、Studioで新しいオペレーションを生成するために使用します。オペレーションに基づいて@Actionを生成すると、新しいオペレーションの初期属性 (説明、入力、出力、レスポンス) は@Action注釈の定義から取得されます。

プラグインを開発している場合、単一値を返すアクションを正しい注釈を付ける必要があります。注釈では、singleResultKeyという特殊名を持つ出力を宣言する必要があります。この場合、ActionExecutionGoal.SINGLE_RESULT_KEYという役に立つ定数があり、次のように使用できます。

```
@Action(name = "modulo-ten",
     description = "returns the last digit",
     outputs = @Output(ActionExecutionGoal.SINGLE_RESULT_KEY),
     responses = @Response(text = ResponseNames.SUCCESS,
     field = OutputNames.RETURN_RESULT,
     value = "0", matchType = MatchType.ALWAYS_MATCH,
     responseType = ResponseType.RESOLVED)
   )
   public int moduloTen(@Param("number") int number) {
       return number % 10;
   }
```

注: @Action注釈を使用することが重要です。使用しないと、これらの@Actionを作成したオペレーションの使用は難しくなります。

注釈

メタデータの追加とは、関連する注釈とそれらの注釈の属性を、追加または設定することです。以下の項目では、@Action、@Output、@Response、@Paramの各注釈について説明します。

操作

属性:

- value (オプション): @Actionの名前
- description (オプション)
- Output[] (オプション): 出力の配列 (以下を参照)
- Response[] (オプション): レスポンスの配列 (以下を参照)

コメント:

次の2つのオプションを使って@Actionの名前を設定できます。

1. value属性:

```
@Action("aflPing")
public void ping(…)
```

または

```
@Action(value="aflPing")
public void ping(…)
```

2. メソッド名:

```
@Action
public void ping(…)
```

名前は上記の順序でチェックされます。最初にチェックされるのはvalue属性です。この属性が存在しない場合、メソッド名が選択されます。

パラメーター

属性:

- value: 入力の名前
- required (オプション): デフォルトではfalse
- encrypted (オプション): デフォルトではfalse
- description (オプション)

コメント:

これは、@Actionデータだけでなく実行でも重要な注釈です。

入力は、操作に必要なデータをオペレーションまたはフローに渡します。各入力は変数にマッピングされます。フロー、オペレーション、またはステップに入力を作成できます。

Studioでは、入力に対して以下の操作が可能です。

- 特定の値に設定する
- 別のステップで収集された情報から取得する
- フローの実行者がフローの開始時に入力する

詳細については、『HPE OO 10.60 Studioオーサリングガイド』を参照してください。実行機能の詳細については、@Actionへの引数の引き渡しを参照してください。

出力

属性:

- **value**: 出力の名前
- **description** (オプション)

コメント:

Studioのオペレーションで複数の出力を使用するには、@Action自体でこれらの出力を宣言する必要があります。複数の出力に値を割り当てるには、戻り値が`Map<String, String>`の@Actionを作成します。

Studioのオペレーションで1つの出力だけを使用するには、@Action自体で戻り値にこの出力を宣言し、バインディングに`SINGLE_RESULT_KEY`を使用する必要があります。

出力は、オペレーションまたはフローによって生成されるデータです。たとえば、サクセスコード、出力文字列、エラー文字列、障害メッセージなどがあります。

Studioでは、オペレーションには次のような異なる出力があります。

- 未加工結果: 戻されたデータ全体 (リターンコード、データ出力、エラー文字列)。
- プライマリ出力とその他の出力。これらは未加工結果の構成要素です。

レスポンス

属性:

- `text`: レスポンスの各トランジションによって表示されるテキスト
- `field`: 評価対象のフィールド
- `value`: field内の予期される値
- `description`: (オプション)
- `isDefault`: これがデフォルトのレスポンスかどうかを指定します。デフォルト値はfalseです。この属性をtrueに設定できるのは、@Action内の1つのレスポンスだけです。
- `mathType`: 値に対して有効にする比較条件のタイプ。たとえば、(field = `fieldName`, value = `0`, matchType = `COMPARE_GREATER`) と定義すると、このレスポンスはフィールド`fieldName`の値が`0`より大きい場合に選択されます。
- `responseType`: レスポンスのタイプ (Success、Failure、Diagnosed、No_Action、またはResolve_By_Name)。

- `isOnFail`: これが`On-Fail`レスポンスかどうかを指定します。デフォルト値はfalseです。この属性を`true`に設定できるのは、@Action内の1つのレスポンスだけです。
- `ruleDefined`: このレスポンスにルールが定義されているかどうかを指定します。ルールが定義されていないレスポンスは、デフォルトのレスポンスとして使用できます。ルールが定義されていないレスポンスは、1つの@Actionで1つだけ使用できます。
- **コメント:**
- レスポンスは、オペレーションまたはフローにおいて考えられる結果です。レスポンスには次の1つのルールが含まれます。`field`が`value`と一致する

詳細については、『HPE OO 10.60 Studioオーサリングガイド』を参照してください。

@Actionデータ定義の例

```
@Action(value = "aflPing",
         description = "perform a dummy ping",
         outputs = {@Output(value = RETURN_RESULT, description ="returnResult description"),
                    @Output(RETURN_CODE),
                    @Output("packetsSent"),
                    @Output("packetsReceived"),
                    @Output("percentagePacketsLost"),
                    @Output("transmissionTimeMin"),
                    @Output("transmissionTimeMax"),
                    @Output("transmissionTimeAvg")},
         responses = {@Response(text = "success", field = RETURN_CODE, value = PASSED),
                      @Response(text = "failure", field = RETURN_CODE, value = FAILED)})

 public Map<String, String> doPing(
     @Param(value = "targetHost",
            required = true,
            encrypted = false,
            description = "the host to ping") String targetHost,
     @Param("packetCount") String packetCount,
     @Param("packetSize") String packetSize) {
     …
 }
```

拡張機能のテスト

プロジェクトのビルドの一部としての拡張機能のテスト

@Actionは単純なJavaメソッドです。JUnitなどの標準的なJavaテストツールを使ってテストでき、Mavenプロジェクトの標準ライフサイクルフェーズを利用できます。

@Actionはそれ自体が正規のメソッドであるため、HPE Operations Orchestrationのコンポーネントを呼び出す必要はありません。呼び出しは、テストケースのJavaメソッドの直接呼び出しとすることができます。

コマンドラインとは関係のない拡張機能のテスト

プラグインにパッケージ化した拡張機能は、テストの目的でコマンドラインから呼び出すことができます。以下に、@Actionの例を示します。

```
public class TestActions {
     @Action
     public int sum(@Param("op1") int x, @Param("op2") int y){
         return x+y;
     }
 }
```

TestActions クラスが、次の groupId、artifactId、version (GAV) が指定されたプラグインに格納されているとします。
`com.mycompany:my-actions:1.0`

次のように、sum @Actionをコマンドラインから呼び出すことができます。

```
mvn com.mycompany:my-actions:1.0:execute -Daction=sum -Dop1=1 -Dop2=3 -X
```

このコマンドの結果は長いトレースになります。ログメッセージの表示には `-X`オプションが必要です。トレースの終わりあたりで、以下のメッセージが表示されます。

```
[DEBUG] Configuring mojo 'com.mycompany:my-actions:1.0::execute' with basic configurator -->
 [DEBUG]   (f) actionName = sum
 [DEBUG]   (f) session = org.apache.maven.execution.MavenSession@21cfa61c
 [DEBUG] -- end configuration --
 [DEBUG] Action result: action result = 4
```

.NET拡張機能

.NETアクションを使用してコンテンツを作成するには、次の操作を実行する必要があります。

1. バージョン9.xの場合と同じように、必要なアクションの実装が含まれたDLLファイルを作成します。アクションクラスでIActionインタフェースを実装する必要があります。
2. mvn install:install-fileを使用して、作成したDLLを (参照先のライブラリも含め) ローカルのMavenリポジトリにデプロイします。Mavenでビルドしたのではないアーティファクトのインストールの詳細については、http://maven.apache.org/plugins/maven-install-plugin/usage.htmlを参照してください。
3. .NETアクションをラップする、HPE OOのMavenプラグインを生成します。このためには、次の操作を実行します。
   a. **pom.xml**ファイルを作成します。POMのリファレンスについては、http://maven.apache.org/pom.htmlを参照してください。
   b. <dependencies> に、必要なすべてのDLLが含まれたリストを追加します。<type>dll</type>を使用して、DLLのすべてのアーティファクトを定義します。
   c. **pom.xml**ファイルが含まれているフォルダーからmvn installコマンドを実行します。ここでは、Mavenのbinフォルダーがシステムpathに含まれていると見なしています。

生成されたMavenプラグインはターゲットフォルダーに配置され、ローカルのMavenリポジトリにインストールされます。ターゲットフォルダーの場所は、現在のフォルダーに対して相対的な場所になります。

pom.xmlの内容:

```
<?xml version="1.0" encoding="UTF-8"?>
 <project xmlns="http://maven.apache.org/POM/4.0.0"
          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-
4.0.0.xsd">
     <modelVersion>4.0.0</modelVersion>

     <groupId>[my plugin groupId]</groupId>
     <artifactId>[my plugin artifactId]</artifactId>
     <version>[my plugin version]</version>

     <packaging>maven-plugin</packaging>

     <properties>
         <oo-sdk.version>[THE LATEST HPE OO_SDK VERSION]</oo-sdk.version>
         <oo-dotnet.version>[THE LATEST HPE OO_DOTNET VERSION]</oo-dotnet.version>
     </properties>

     <dependencies>
         <!-- required dependencies -->
         <dependency>
             <groupId>com.HP.oo.sdk</groupId>
             <artifactId>oo-dotnet-action-plugin</artifactId>
             <version>${oo-sdk.version}</version>
         </dependency>

         <dependency>
             <groupId>com.HP.oo.dotnet</groupId>
             <artifactId>oo-dotnet-legacy-plugin</artifactId>
             <version>${oo-dotnet.version}</version>
             <type>dll</type>
         </dependency>

         <dependency>
             <groupId>${project.groupId}</groupId>
             <artifactId>IAction</artifactId>
             <version>9.0</version>
             <type>dll</type>
         </dependency>
         <!-- end of required dependencies -->

         <dependency>
             <groupId>[groupId-1]</groupId>
             <artifactId>[artifactId-1]</artifactId>
             <version>[version-1]</version>
```

```
            <type>dll</type>
        </dependency>

        <dependency>
            <groupId>[groupId-2]</groupId>
            <artifactId>[artifactId-2]</artifactId>
            <version>[version-2]</version>
            <type>dll</type>
        </dependency>

...

        <dependency>
            <groupId>[groupId-n]</groupId>
            <artifactId>[artifactId-n]</artifactId>
            <version>[version-n]</version>
            <type>dll</type>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>com.HP.oo.sdk</groupId>
                <artifactId>oo-action-plugin-maven-plugin</artifactId>
                <version>${oo-sdk.version}</version>
                <executions>
                    <execution>
                        <id>generate plugin</id>
                        <phase>process-sources</phase>
                        <goals>
                            <goal>generate-dotnet-plugin</goal>
                        </goals>
                    </execution>
                </executions>
            </plugin>
        </plugins>
    </build>
</project>
```

次の例の設定:

- POMファイルの名前はexample.pom.xmlです。
- 必要な@Actionは**my-dotnet-actions.dll**に格納されています。
- 生成されるMavenプラグインは**com.example:my-dotnet-plugin:1.0**です。

```
<?xml version="1.0" encoding="UTF-8"?>
 <project xmlns="http://maven.apache.org/POM/4.0.0"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-
4.0.0.xsd">


     <modelVersion>4.0.0</modelVersion>
     <groupId>com.example</groupId>
     <artifactId>my-dotnet-plugin</artifactId>
     <version>1.0</version>
     <packaging>maven-plugin</packaging>

     <properties>
         <oo-sdk.version>[THE LATEST HPE OO_SDK VERSION]</oo-sdk.version>
         <oo-dotnet.version>[THE LATEST HPE OO_DOTNET VERSION]</oo-dotnet.version>
     </properties>

     <dependencies>
       <!-- required dependencies -->
         <dependency>
             <groupId>com.HP.oo.sdk</groupId>
             <artifactId>oo-dotnet-action-plugin</artifactId>
             <version>${oo-sdk.version}</version>
         </dependency>
         <dependency>
             <groupId>com.HP.oo.dotnet</groupId>
```

```
                <artifactId>oo-dotnet-legacy-plugin</artifactId>
                <version>${oo-dotnet.version}</version>
                <type>dll</type>
            </dependency>
            <dependency>
                <groupId>${project.groupId}</groupId>
                <artifactId>IAction</artifactId>
                <version>9.0</version>
                <type>dll</type>
            </dependency>
          <!-- end of required dependencies -->
            <dependency>
                <groupId>com.example</groupId>
                <artifactId>my-dotnet-actions</artifactId>
                <version>1.0</version>
                <type>dll</type>
            </dependency>
        </dependencies>

        <build>
            <plugins>
                <plugin>
                    <groupId>com.HP.oo.sdk</groupId>
                    <artifactId>oo-action-plugin-maven-plugin</artifactId>
                    <version>${oo-sdk.version}</version>
                    <executions>
                        <execution>
                            <id>generate plugin</id>
                            <phase>process-sources</phase>
                            <goals>
                                <goal>generate-dotnet-plugin</goal>
                            </goals>
                        </execution>
                    </executions>
                </plugin>
            </plugins>
        </build>
    </project>
```

レガシーアクション

レガシーアクションを使用してコンテンツを作成するには、次の操作を実行する必要があります。

1. バージョン9.xの場合と同じように、必要なアクションの実装が含まれたJARがあることを確認します。アクションクラスでIActionインタフェースを実装する必要があります。
2. mvn **install:install-file**を使用して、JARを (参照先のライブラリも含め) ローカルのMavenリポジトリにデプロイします。Mavenでビルドしたのではないアーティファクトのインストールの詳細については、http://maven.apache.org/plugins/maven-install-plugin/usage.htmlを参照してください。
3. レガシーアクションのライブラリをラップする、HPE OOのMavenプラグインを生成します。このためには、次の操作を実行します。
   a. **pom.xml**ファイルを作成します。POMのリファレンスについては、http://maven.apache.org/pom.htmlを参照してください。
   b. <dependencies> に、必要なすべてのJARが含まれたリストを追加します。
   c. **pom.xml**ファイルが含まれているフォルダーからmvn installコマンドを実行します。ここでは、Mavenのbinフォルダーがシステムpathに含まれていると見なしています。

生成されたMavenプラグインはターゲットフォルダーに配置され、ローカルのMavenリポジトリにインストールされます。ターゲットフォルダーの場所は、現在のフォルダーに対して相対的な場所になります。

pom.xmlの内容:

```
<?xml version="1.0" encoding="UTF-8"?>
 <project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-
4.0.0.xsd">
     <modelVersion>4.0.0</modelVersion>

     <groupId>[my plugin groupId]</groupId>
     <artifactId>[my plugin artifactId]</artifactId>
     <version>[my plugin version]</version>

     <packaging>maven-plugin</packaging>

     <properties>
         <oo-sdk.version>[THE LATEST HPE OO_SDK VERSION]</oo-sdk.version>
         <oo-dotnet.version>[THE LATEST HPE OO_DOTNET VERSION]</oo-dotnet.version>
     </properties>


     <dependencies>
         <!-- required dependencies -->
         <dependency>
             <groupId>com.HP.oo.sdk</groupId>
             <artifactId>oo-legacy-action-plugin</artifactId>
             <version>${oo-sdk.version}</version>
         </dependency>
         <!-- end of required dependencies -->

         <dependency>
             <groupId>[groupId-1]</groupId>
             <artifactId>[artifactId-1]</artifactId>
             <version>[version-1]</version>
         </dependency>

         <dependency>
             <groupId>[groupId-2]</groupId>
             <artifactId>[artifactId-2]</artifactId>
             <version>[version-2]</version>
         </dependency>

 ...

         <dependency>
             <groupId>[groupId-n]</groupId>
             <artifactId>[artifactId-n]</artifactId>
```

```
            <version>[version-n]</version>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>com.HP.oo.sdk</groupId>
                <artifactId>oo-action-plugin-maven-plugin</artifactId>
                <version>${oo-sdk.version}</version>
                <executions>
                    <execution>
                        <id>generate plugin</id>
                        <phase>process-sources</phase>
                        <goals>
                            <goal>generate-legacy-plugin</goal>
                        </goals>
                    </execution>
                </executions>
            </plugin>
        </plugins>
    </build>
 </project>
```

次の例の設定:

- POMファイルの名前は**example.pom.xml**です。
- 必要なアクションは**my-legacy-actions.jar**に格納されています。
- 生成されるMavenプラグインは**com.example:my-legacy-actions:1.0**です。

```
<?xml version="1.0" encoding="UTF-8"?>
 <project xmlns="http://maven.apache.org/POM/4.0.0"
          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-
4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>com.example</groupId>
    <artifactId>my-legacy-actions-plugin</artifactId>
    <version>1.0</version>

    <packaging>maven-plugin</packaging>

    <properties>
        <oo-sdk.version>[THE LATEST HPE OO_SDK VERSION]</oo-sdk.version>
        <oo-dotnet.version>[THE LATEST HPE OO_DOTNET VERSION]</oo-dotnet.version>
    </properties>

    <dependencies>
        <!-- required dependencies -->
        <dependency>
            <groupId>com.HP.oo.sdk</groupId>
            <artifactId>oo-legacy-action-plugin</artifactId>
            <version>${oo-sdk.version}</version>
        </dependency>
        <!-- end of required dependencies -->

        <dependency>
            <groupId>com.example</groupId>
            <artifactId>my-legacy-actions</artifactId>
            <version>1.0</version>
        </dependency>

    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>com.HP.oo.sdk</groupId>
                <artifactId>oo-action-plugin-maven-plugin</artifactId>
                <version>${oo-sdk.version}</version>
                <executions>
                    <execution>
```

```
                        <id>generate plugin</id>
                        <phase>process-sources</phase>
                        <goals>
                            <goal>generate-legacy-plugin</goal>
                        </goals>
                    </execution>
                </executions>
            </plugin>
        </plugins>
    </build>
</project>
```