HP Operations Orchestration

ソフトウェアバージョン: 10.20
Windows および Linux オペレーティングシステム

アクション開発者ガイド

ドキュメントリリース日 : 2014 年 11 月 (英語版)
ソフトウェアリリース日 : 2014 年 11 月 (英語版)

ご注意

保証

HP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、HP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、HP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.

謝辞

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.hp.com/selfsolve/manuals

このサイトを利用するには、HP Passportへの登録とサインインが必要です。HP 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.hp.com/passport-registration.html**

または、HP 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、HPの営業担当にお問い合わせください。

サポート

HP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。**http://www.hp.com/go/hpsoftwaresupport**

このサイトでは、HPのお客様窓口のほか、HP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。

HP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
- HP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

一部のサポートを除き、サポートのご利用には、HP Passportユーザーとしてご登録の上、サインインしていただく必要があります。また、多くのサポートのご利用には、サポート契約が必要です。HP Passport IDを登録するには、次のWebサイトにアクセスしてください。

http://h20229.www2.hp.com/passport-registration.html

アクセスレベルの詳細については、次のWebサイトをご覧ください。

http://h20230.www2.hp.com/new_access_levels.jsp

HP Software Solutions Nowは、HPSWのソリューションと統合に関するポータルWebサイトです。このサイトでは、お客様のビジネスニーズを満たすHP製品ソリューションを検索したり、HP製品間の統合に関する詳細なリストやITILプロセスのリストを閲覧することができます。このサイトのURLはhttp://h20230.www2.hp.com/sc/solutions/index.jsp です。

目次

HP 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 HP 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。

HP 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 HP 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 HP 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。

HP 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。

HP 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」(10ページ)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、HP 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」(15ページ)を参照してください。

注: HP 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。

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。

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

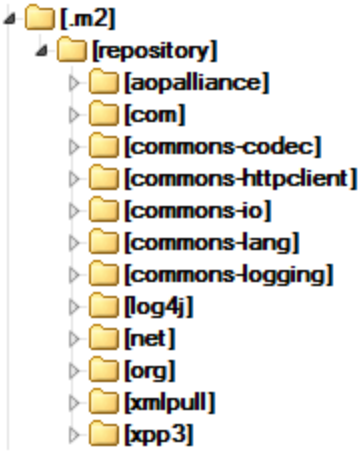
- `sdk-dotnet-<version>.zip` と sdk-java-<version>.zip を次の場所に展開します。

  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 HP 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

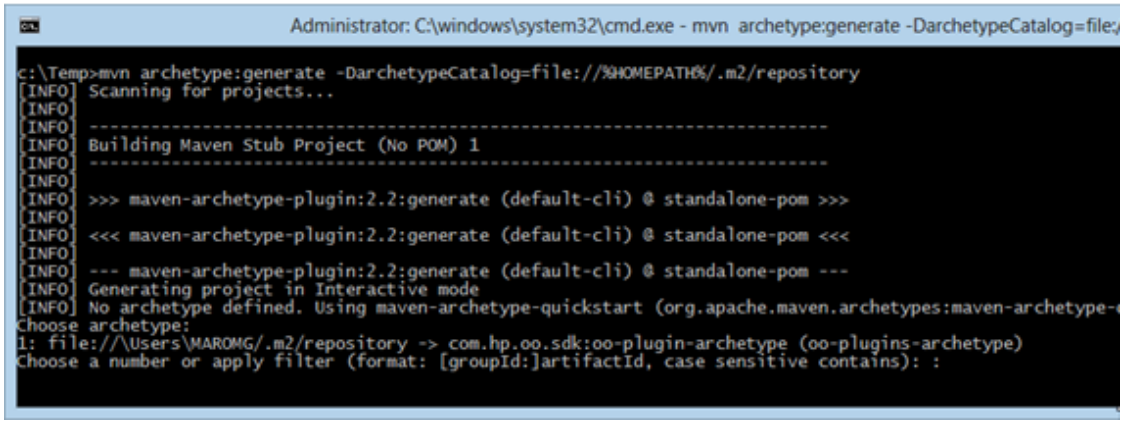
1. サンプルプラグインのプロジェクトを作成するパスに移動し、コマンドラインで次のコマンドを入力します。

   ```
   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。

```
Define value for property 'groupId': : acmeGroup
Define value for property 'artifactId': : acmeArtifact
[INFO] Using property: version = 1.0.0
Define value for property 'package':  acmeGroup: :
Define value for property 'uuid': : e3a3afb0-df2b-11e3-8b68-0800200c9a66
Confirm properties configuration:
groupId: acmeGroup
artifactId: acmeArtifact
version: 1.0.0
package: acmeGroup
uuid: e3a3afb0-df2b-11e3-8b68-0800200c9a66
 Y: : _
```

ビルドが終了してプロジェクトが作成されます。

```
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 18.112s
[INFO] Finished at: Mon Jun 17 21:17:50 IDT 2013
[INFO] Final Memory: 13M/490M
[INFO] ------------------------------------------------------------------------
c:\Temp>_
```

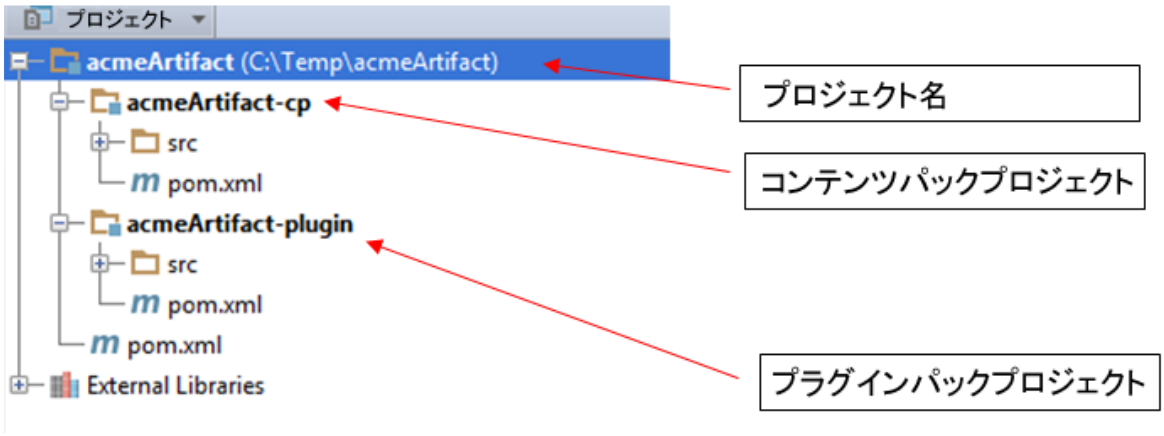
Java IDE でプロジェクトを開く

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。

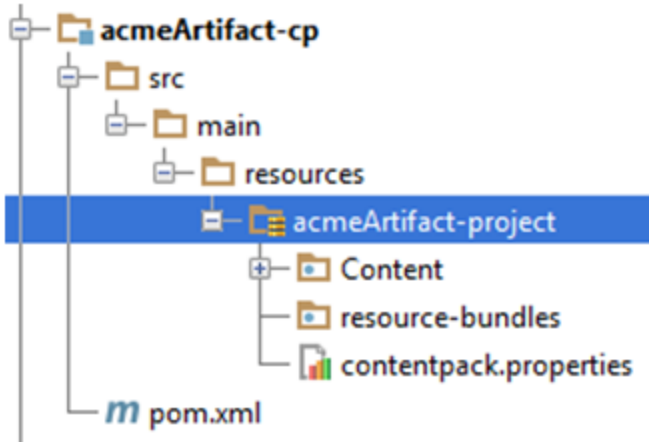
コンテンツパックプロジェクト

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** 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。



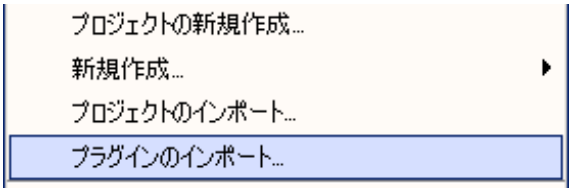
プロジェクト内でフロー、オペレーション、または構成アイテムを作成する場合、Maven でこのプロジェクトをビルドできます。結果の JAR ファイルはコンテンツパックであり、Central にデプロイするか、別のユーザーによって Studio で再度開くことができます。

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-plugin** モジュールをビルドします。

3. プラグインを Studio にインポートします。



プラグインのパスは、ローカルの Maven リポジトリ下 の **acmeArtifact-plugin.jar** ファイルのパスとなります。

4. 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、HP 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 HP 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`、`@Param`、`@Output`、`@Response` 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、『HP OO 10.00 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」(11ページ)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、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です。

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。

詳細については、『HP OO 10.00 Studio オーサリングガイド』を参照してください。

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、『HP OO 10.00 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、HP 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 @Action の実装が含まれた DLL ファイルを作成します。@Action 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、HP 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 HP OO_SDK VERSION]</oo-sdk.version>
        <oo-dotnet.version>[THE LATEST HP OO_DOTNET VERSION]</oo-dotnet.version>
    </properties>

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

        <dependency>
            <groupId>com.hp.oo</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</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>2.190</oo-sdk.version>
        <oo-dotnet.version>1.30</oo-dotnet.version>
    </properties>

    <dependencies>
      <!-- required dependencies -->
        <dependency>
            <groupId>com.hp.oo</groupId>
            <artifactId>oo-dotnet-action-plugin</artifactId>
            <version>${oo-sdk.version}</version>
        </dependency>
        <dependency>
            <groupId>com.hp.oo</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</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、HP 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 HP OO_SDK VERSION]</oo-sdk.version>
        <oo-dotnet.version>[THE LATEST HP OO_DOTNET VERSION]</oo-dotnet.version>
    </properties>


    <dependencies>
        <!-- required dependencies -->
        <dependency>
            <groupId>com.hp.oo</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</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>2.190</oo-sdk.version>
        <oo-dotnet.version>1.30</oo-dotnet.version>
    </properties>

    <dependencies>
        <!-- required dependencies -->
        <dependency>
            <groupId>com.hp.oo</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</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>
```


Java™
COMPATIBLE