Magic xpi
コネクタビルダー

OUTPERFORM THE FUTURE™

イントロダクション

Magic xpi コネクタビルダーは Magic xpi で使用される全機能を備えたプロフェッショナル グレードのコネクタを開発者が作成できる新機能です。

新しいビルダーで開発されたコネクタには、次の機能があります。:

- カスタム設定と組込みデータマッパー(フラットファイル, JSON, XML)を備えたステップ。
- メソッド インターフェイス。
- カスタムリソースとサービス、検証ボタンと追加可能な 3 つのアクションボタン。
- カスタム設定画面または静的設定画面を備えたトリガー。
- ポーリング、外部およびエンドポイント トリガーがサポートされています。
- 外部およびエンドポイント トリガーは、同期または非同期の動作をさせることができます。
- 1 行のコードで設定 UI 用の式エディタ、変数選択、エラー/メッセージダイアログボックスを組み込むことができます。
- Java、.NET および Magic xpa がランタイムのテクノロジーとしてサポートされます。
- 設定 UI の実装に.NET テクノロジーが使用されます。(WinForms or WPF).

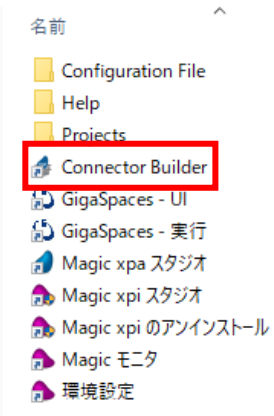
次のドキュメントでは、Magic xpi コネクタビルダーを使用して新しいダイナミックコネクタを作成する方法について説明します。 さらに、トリガーについても説明します。

ダイナミック ステップの作成

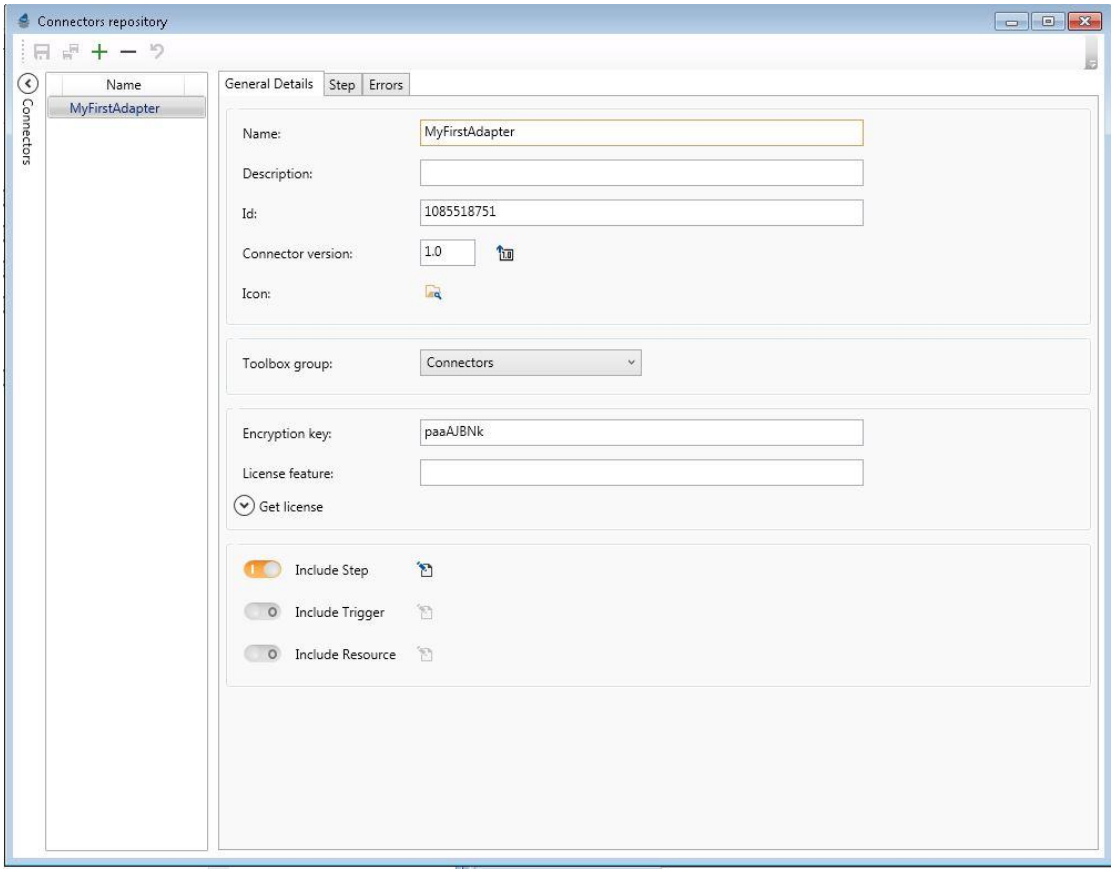
コネクタを作成する最初のステップは、コネクタ ビルダー ユーティリティを使用してコネクタを定義することです。 このユーティリティが、リソース、UI コード、ランタイムコード、エラー、アイコンなど、コネクタの様々な部分を全て取り纏めています。

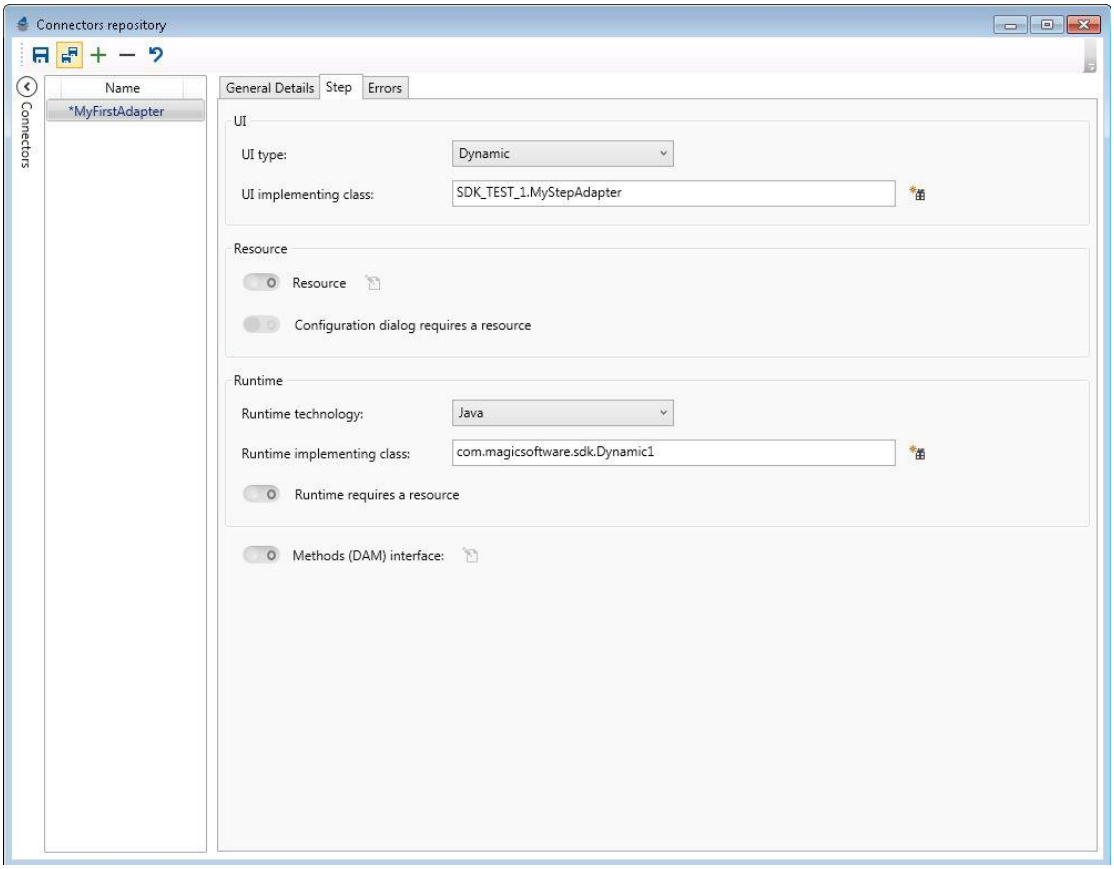
コネクタビルダーを使用したコネクタの定義

1. Magic xpi スタートメニューまたはデスクトップフォルダからコネクタビルダー(Connector Builder)ユーティリティを開きます。
2. 開いたウィンドウの左上で新規コネクタ作成ボタンをクリックする。



3. コネクタ名、オプションアイコン、説明を定義します。
4. **UI タイプ(UI Type)をダイナミック(Dynamic)**とします。.
5. 以下の構文で **UI 実装クラス(UI Implementing class)**を指定します:
   <ネームスペース>.<クラス名>
   - この例では: **SDK_TEST_1.MyStepAdapter**
6. コネクタのローカルエージェントと互換性を持たせるには、[リソースを含める]トグルボタンをオンにしてから、[リソース]タブで[ローカルエージェントの互換性]トグルボタンをオンにします。 デフォルトでは、このオプションはオフになっています。
7. ランタイムテクノロジー(Runtime technology)に Java 或いは .NET を選択します。
8. ランタイム実装クラス(Runtime Implementing class)を定義します。
   - Java の場合: com.magicsoftware.sdk.Dynamic1
   - .NET の場合: SDK_TEST_1.MyStepAdapterRT
   - Magic xpa の場合 : MyConnectorName.ecf
9. ウィンドウの左上隅にある[すべてのコネクタの変更を保存]ボタンをクリックして、ビルダーを閉じます。





この設定で以下のパスにコネクタ フォルダが作成されます。: **<Magic xpi インストール先>\Runtime\addon_connectors**.

コネクタフォルダは先に指定したコネクタ名と同一になります。

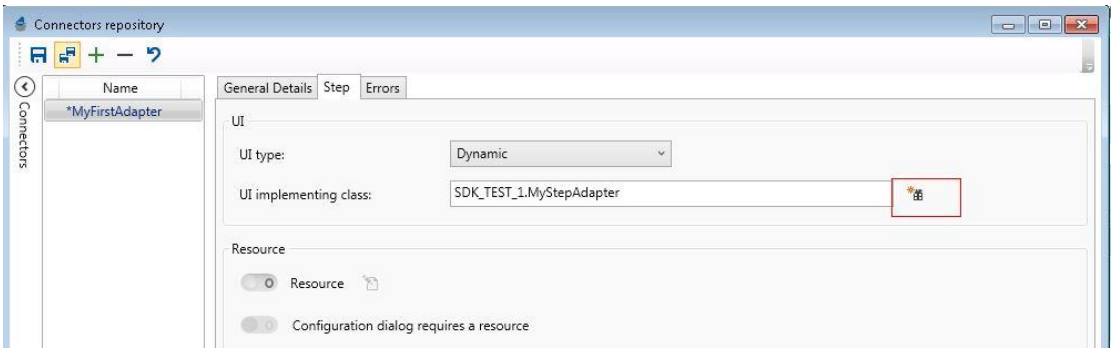
コネクタ フォルダには幾つかの重要なロケーションが存在します。:

- **ui\lib** – このフォルダには UI の dll が配置されます。
- **runtime\java\lib** – このフォルダには、サーバ実行用に作成されたランタイム jar ファイルが配置されます。
- **runtime\dotnet \lib** – このフォルダには、サーバ実行用に作成されたランタイム dll ファイルが配置されます。
- **untime\magic xpa\lib** – このフォルダには、サーバ実行用に作成されたランタイム ecf ファイルが配置されます。

スタジオ用設定 UI の作成

コネクタの UI 部分の開発を開始する最も簡単な方法は、あらかじめ定義されたテンプレートからソースコードを生成することです。

UI クラス名の横にある **UI プロジェクト生成(Generate UI Project)** ユーティリティを使用します。このユーティリティは、全ての参照、実装されたインターフェース、およびいくつかの使用例が事前設定されたサンプルプロジェクトを生成します。



ユーティリティでは.NET 名とその場所を尋ねられます。このネームスペースは先に入力した **UI 実行クラス(UI implementing class)**から取得されます。

*** 付録 A では IDE で UI プロジェクトを手動で設定する手順が説明されています。

プロジェクトをビルドしたら、プロジェクトの dll を**<コネクタ名>\ui\lib** フォルダにコピーする必要があります。

IUserComponent インターフェイス説明

コネクタの UI 部分のステップ アダプタ クラスには、実装する必要のあるいくつかのメソッドが含まれています。:

CreateDataObject() メソッド

このメソッドは、コネクタ設定に使用する一連のプロパティを保持するオブジェクトを戻します。このオブジェクトは、Magic xpi プロパティのセットを保持するクラスインスタンスです。(このタイプのクラスの例は、生成されたテンプレートで見つけることができます)。

プロパティは文字や数値といった Magic プリミティブでも構いませんが、**変数**や**式**を使用することができます。

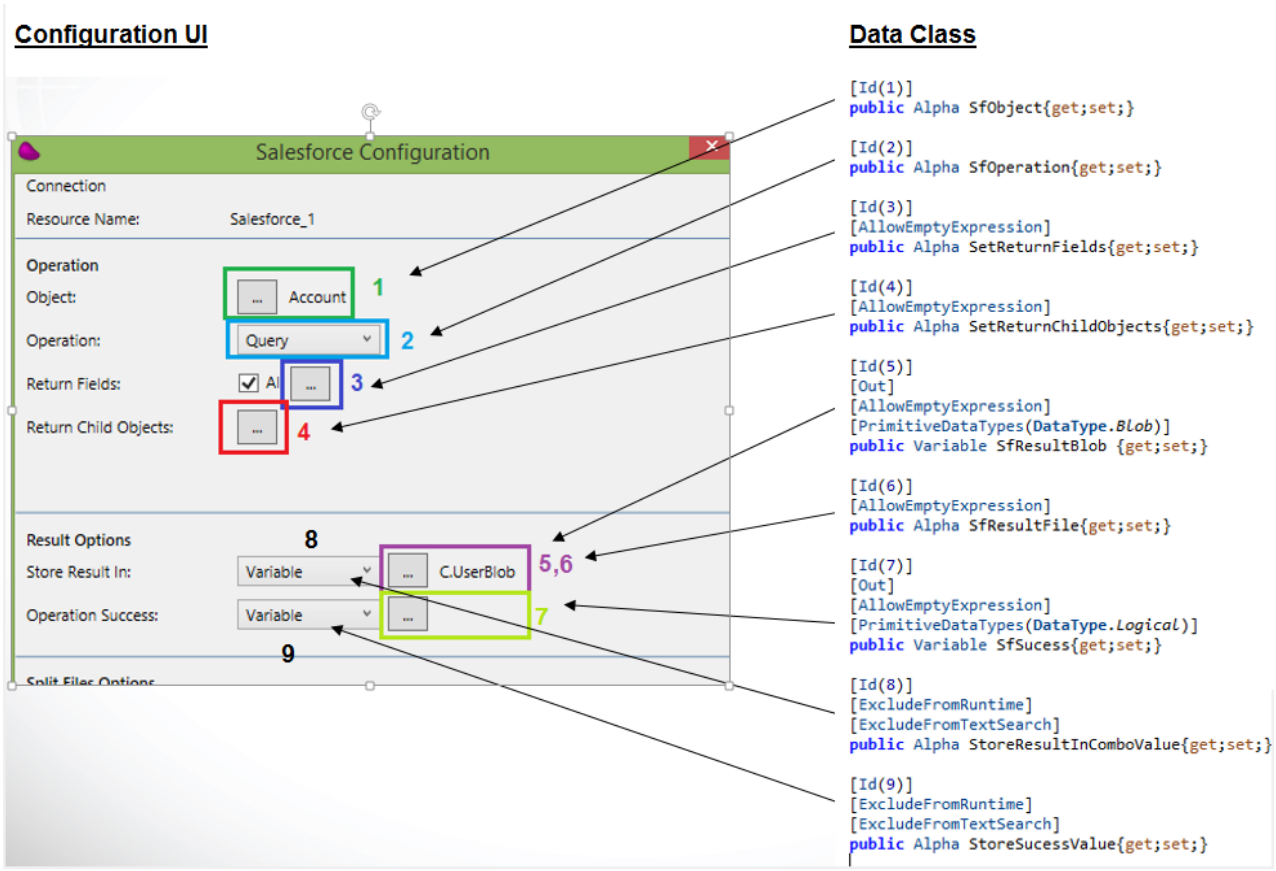
各プロパティに対して、その動作を制御する一連のアノテーションを定義できます。たとえば、型変数のプロパティに対して、方向を定義できます。つまり、実行時に、このプロパティにマップされた変数は、その値をコネクタに渡すだけで[In]、コネクタの実行後に新しい値[Out]または[InOut]の両方を更新することができます。もう一つの例は文字プロパティのエンコーディング設定です。

ステップでは、基本的な Magic タイプ(文字、論理、数値、日付、時刻)と式タイプはステップに値を渡すことしかできません。これは、その方向が常に IN であることを見します。IN はデフォルトの方向なので、これらの型の方向アノテーションを定義する必要はありません。

Magic タイプの変数は、ステップへの入力データとステップの結果を保持する変数の両方として使用できるため、変数タイプを以下のように定義することができます。:

- [In] – 変数データは実行時にステップに渡されます。例えば、選択した変数が C.UserString の場合、C.UserString 内のデータは実行モジュールのソースコードで使用できます。
- [Out] – ステップの実行時のロジックでこの変数の値を更新することができます。例えば、選択された変数が C.UserString である場合、ステップが終了すると、C.UserString はステップのロジックにより更新されたデータを保持します。
- [InOut] –変数データは実行時にステップに渡され、ステップが終了するとステップのロジックでこの変数にデータを更新します。

Salesforce コネクタを例に、設定プロパティを予想されるデータクラスにマップします。:



- Object (1) – ボタンをクリックすると、Salesforce への接続が開き、すべてのオブジェクトのリストが要求され、ダイアログボックスにリストされ、ユーザが必要なオブジェクトを選択できるようになります。 このフィールドには、選択したオブジェクト名の静的な文字列値が格納されます。この値は、object プロパティから実行時に必要な値です。
- Operation (2) – このフィールドは、オブジェクトを選択したときに取得されたメタデータの一部を使用して、オブジェクトのサポートされている操作：オペレーションを一覧表示します。 フィールドには、静的な文字列値又は操作：オペレーションを表す静的数値(1,2,3..)が格納されます。この値は、実行時に Operation プロパティから必要な値です。
- Return fields (3) – ボタンをクリックすると、Salesforce への接続が開き、オブジェクトフィールドの一覧が表示されますユーザは必要なフィールドを複数選択できます。 このフィールドには、静的文字列値とオブジェクトフィールドのリストが格納されます。この値は、実行時に Return fields フィールドのプロパティから必要な値です。このプロパティは必須ではないので、アノテーション[AllowEmptyExpression]を使用して、このプロパティに値が設定されていない場合、チェッカーがエラーを発生しないようにします。

- Return child objects (4) – ボタンをクリックすると、Salesforce への接続が開き、子オブジェクトの一覧が表示されます。ユーザはオブジェクトを複数選択できます。 このフィールドには、静的文字列値と子オブジェクト名のリストが格納されます。この値は、実行時に Return child objects プロパティから必要な値です。アノテーション[AllowEmptyExpression]を使用して、このプロパティに値が設定されていない場合にチェッカーがエラーを発生させないようにします。
- Store result in > 変数選択 (5) – このプロパティは、実行時にステップ結果で更新される変数の名前を保持する必要があります。方向を示す[Out]のアノテーションを追加します。これは変数であるため、チェッカーとランタイムの両方に対して型を定義する必要があります。そのためには、別のアノテーション[PrimitiveDataTypes(DataType)]を追加します。また、ファイルまたは変数があるので、このプロパティにアノテーション[AllowEmptyExpression]が表示されるので、どちらのオプションも必須として設定できません。
- Store result in > ファイル選択 (6) – このプロパティは、実行時にステップ結果で更新されるファイルパスを保持する必要があります。これはファイルパスなので、コネクタは実行時にこの値を取得しなければなりません。したがって、このプロパティの方向は IN です。実行時に実際のコネクタコードによってファイルが作成されます。また、ファイルまたは変数があるので、このプロパティのアノテーション[AllowEmptyExpression] は必須として設定できません。そのうちの 1 つが値を持つことを確認することは、ui レベルまたは check()メソッドを使用して、チェッカープロセスにロジックを追加することができます。
- Operation Success > 変数選択 (7) – このプロパティは、OUT 方向の変数名を保持する必要があります。この定義は、変数の型が BLOB ではなく論理的であることを除けば、Store result in と同様です。
- Store result in コンボボックス (8) と Operation Success コンボボックス (9) – データクラスを使用すると、開発者は実行時に関連するプロパティを格納するだけでなく、設定時に関連するプロパティも格納することができます。Salesforce の設定画面では、コンボボックスの値を保存したいので、コネクタを再設定すると、以前に選択したユーザと同じ値がコンボボックスに表示されます。この例では、次の 2 つの重要なアノテーションがあります。:

  1. [ExcludeFromRuntime] – このアノテーションは、このプロパティ値をランタイムコードに渡さないことを示します。

  2. [ExcludeFromTextSearch] – このアノテーションは、このプロパティをスタジオのテキスト検索から除外する必要があることを示します。コンボボックスの値の結果は、テキスト検索結果ペインに表示されません。

configure() メソッド

configure()メソッドは、コネクタの設定 UI のメインエントリポイントです。このメソッドから開発者は次のことを行います。:

- カスタム設定 UI を開きます。
- Salesforce に接続してオブジェクトのリストを取得するなど、カスタム構成ロジックを実行します。
- サーバに接続するための資格情報など、リソースのプロパティを取得します。
- ユーザが設定した値でデータオブジェクトを更新します。
- どのスキーマを開くか(JSON、XML、フラットファイル)を定義します。
- 設定が変更され、保存する必要があるかどうかを定義します(スタジオでのダーティ表示)。
- 組込ユーティリティ (テキスト検索、チェッカー、クロスリファレンス) からの移動要求の結果として、どのプロパティを重点的に実行するかを指定します。

configure() – bool 値を戻します

- **True** が返された場合:
  - 設定が OK で終了したことを意味します。
  - **GetSchema()**が呼び出され、指定したスキーマでデータマッパーが開きます。
  - **configure()**メソッドが True を返し、**configurationChanged** が **True** に設定されている場合、データマッパーはダーティな状態で開かれており、保存する必要があることを示します。
- **False** が返された場合:
  - 設定が **Cancel** で終了したことを意味します。
  - メソッドは、データマッパーを開かずに、データクラスに加えられた変更を保存せずに終了します。

configurationChanged 動作

前述ように、このプロパティはコネクタのダーティ表示を制御します。**configurationChanged** が True に設定されている場合は、何かが変更されたことを意味します。

ダーティ状態になるためには、**configure()**メソッドは **True** を返す必要があります。

GetSchema() メソッド

このメソッドは、configure()メソッドが True を返した後に呼び出されます。このメソッドは、SchemaInfo クラスオブジェクトを返す必要があります。SchemaInfo クラスには、次の 3 つの派生実装があります。:

- XMLSchemaInfo – XML データマッパーの送り先を表し、XSD、エンコーディング、ルート要素などを定義するための専用のプロパティを備えています。
- FlatFileSchemaInfo – フラット ファイル データマッパーの送り先を表し、デリミタ、エンコーディングおよび実際のフラット ファイル構造を定義するための専用のプロパティを持っています。
- JSonSchemaInfo – JSON データマッパーの送り先を表し、JSON スキーマ、エンコーディングなどを定義するための専用のプロパティを備えています。

リソース関連のメソッド

動的ステップは、リソースを使用して構成することができます。リソースは、専用のビルダーを使用してコネクタビルダー ユーティリティで定義されます。

標準のリソースフィールドに加えて、検証ボタンと 3 つのアクションボタンを定義することもできます。

検証 ボタンと アクション は、次のメソッドを使用して実装されます。:

- public bool ValidateResource(IReadOnlyResourceConfiguration resouceData, out string errorMsg) – このメソッドは、リソースで 検証 ボタンがクリックされたときに呼び出されます。このメソッドには、Resource プロパティのオブジェクトの読み取り専用コピーがあり、有効である場合は True を、有効でない場合はエラーメッセージを返します。
- public void InvokeResourceHelper(string helperID, IResourceConfiguration resouceData) –このメソッドは、リソース内でアクションボタンの 1 つがクリックされたときに呼び出されます。このメソッドは、クリックされたボタンの名前と更新可能な Resource プロパティのオブジェクトを受け取ります。このメソッドでは、通常、開発者は追加のダイアログボックスを開き、Resource の値の一部を更新します。 リソースを更新する方法の例は、UI テンプレート内にあります。

Check() メソッド

このメソッドはスタジオのチェッカーによって呼び出され、開発者はカスタムチェッカーの結果をコネクタに追加できます。たとえば、ユーザがステップの結果を保持する変数を選択したか、ファイルパスを指定したかなどを確認します。

デバッグ/デザイン スタジオの状態への応答

Magic xpi スタジオには 2 つの主要な状態があります。:

- デザイン状態 – 開発を行うスタジオの初期状態です。
- デバッグ状態 – プロジェクトをデバッグするときのスタジオの状態です。 この状態では、ダイアログボックスの動作が異なるため、実行中のプロジェクトのソースを変更できません。

コネクタビルダーの UI 部分を作成する場合は、標準のスタジオ ダイアログボックスに準拠し、ユーザが変更を行わないようにすることをお勧めします。通常は、ユーザが構成を表示できるようにしながら、OK ボタンを無効にするだけで十分です。

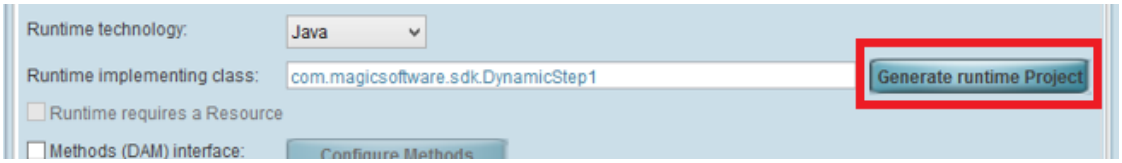
スタジオの状態は、IsStudioInDesignMode という Utils System Properties コレクションのプロパティとして config()メソッドで使用できます。このプロパティは、値として True または False の文字列を返します。

このプロパティにアクセスするには、次のコード行を追加します。:

```
utils.GetSystemProperty("IsStudioInDesignMode");
```

ランタイム実装の作成

コネクタのランタイム部分の開発を開始する最も簡単な方法は、コネクタビルダー ユーティリティからテンプレートプロジェクトを生成することです。



定義したランタイムクラスの横にある**ランタイムプロジェクトの生成(Generate runtime Project)**ボタンを使用して、ユーティリティを開きます。このユーティリティでは、.NET、Java および Magic xpa プロジェクト名とその場所を入力します。このプロジェクトのネームスペースまたはパッケージ名は、前に入力した**ランタイム実装クラス(Runtime implementing class)**プロパティから取得されます。

この例では Java での実装を説明します。

1. Eclipse で新しく Java プロジェクトを作成します。
2. プロジェクト名を設定します。
3. **「既存のソースからプロジェクトを作成」**ラジオボタンを選択します。
4. **「ランタイムプロジェクトの生成」**ユーティリティを使用して生成したプロジェクトへのパスを選択します。

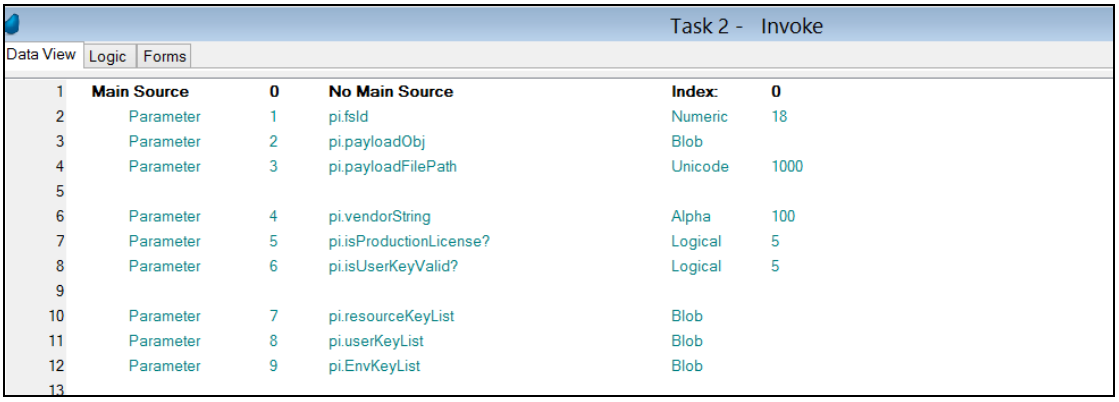
5. **完了**ボタンをクリックして、プロジェクトの作成を終了します。
6. Eclipse でプロジェクトに移動し、クラス(コネクタビルダーで指定した名前)を開きます。

クラスは IStep インターフェイスを実装し、invoke()という 1 つのメソッドを含んでいます。

invoke()メソッドで提供される StepGeneralParams 型のオブジェクトには、以下のものが含まれます。:

a. データマッパー ペイロード : ペイロードのバイト配列を取得するには、getPayloadOBject()または getPayloadFile()を使用します。ペイロード(ファイルまたは BLOB)の場所は、スキーマ UI 設定の以下の.NET コードで制御します。: xmlSchemaInfoLocal.DataDestinationType = 0/1 (0=Blob and 1=file)
b. getUserProperties().get("property name");を使用して UI(データクラス)で定義されたプロパティ。
c. 特定のユーザープロパティを取得すると、以下の操作を実行できます。:
    i. 次を使用し、値を取得する。: getValue().toString()
    ii. 新しい値を設定します(データクラスで変数 Out 型として定義されている場合)。:
        - setAlpha(String value)
        - setBlob(Byte [] args)
        - setDate(Date d)
        - setLogical(Boolean b)
        - setNumeric(Double d)
        - setTime(String t)
d. リソースのプロパティ取得 : getResourceObject()の使用。
e. ライセンス詳細の取得。
f. コネクタにライセンスが設定されている場合は、その詳細を取得します。
g. Magic.ini ファイルで定義されているエンコーディング、プロジェクトの場所、インストール場所、コネクタの場所などの環境設定を取得します。

この例では、xpa 実装を使用します。
xpa スタジオを開きます。 その中で、ConnectorBuilder によって生成された xpa ソリューションを開きます。
次のパラメータを持つ Invoke という名前のプログラムがあります。



1	**Main Source**	**0**	**No Main Source**	**Index:**	**0**
2	Parameter	1	pi.fsId	Numeric	18
3	Parameter	2	pi.payloadObj	Blob	
4	Parameter	3	pi.payloadFilePath	Unicode	1000
5					
6	Parameter	4	pi.vendorString	Alpha	100
7	Parameter	5	pi.isProductionLicense?	Logical	5
8	Parameter	6	pi.isUserKeyValid?	Logical	5
9					
10	Parameter	7	pi.resourceKeyList	Blob	
11	Parameter	8	pi.userKeyList	Blob	
12	Parameter	9	pi.EnvKeyList	Blob	
13					

pi.payloadObj という名前の 2 番目のパラメーターにはデータマッパーペイロードが含まれ、pi.payloadFilePath という名前の 3 番目のパラメーターにはペイロードの場所が含まれます。

次の関数を使用して、ユーザープロパティを設定および取得できます。

- **getUserPropertyValueByKey(PropertyName)**
  この関数は、ユーザープロパティの値を返します。
- **setUserPropValueA(PropertyName,Value)**
  この関数は、Alpha タイプのユーザプロパティの値を設定します。
- **setUserPropValueN(PropertyName,Value)**
  この関数は、数値型のユーザープロパティの値を設定します。
- **setUserPropValueD(PropertyName,Value)**
  この関数は、日付タイプのユーザープロパティの値を設定します
- **setUserPropValueT(PropertyName,Value)**
  この関数は、時間タイプのユーザープロパティの値を設定します
- **setUserPropValueL(PropertyName,Value)**
  この関数は、論理型のユーザープロパティの値を設定します。
- **setUserPropValueB(PropertyName,Value)**
  この関数は、Blob タイプのユーザープロパティの値を設定します

次の関数を使用して、リソースプロパティの値を取得できます。

getResourceValueByKey(ResourceParameterName)

パラメータ pi.vendorString、pi.isProductionLicense？, pi.isuserkeyValid？ は、ライセンスの詳細に関する情報を提供します。

以下の関数を使用して、環境設定を取得できます。

getEnvSettings(EnvPorperty)

pi.resourceKeyList パラメーターは、リソースパラメーター名を提供し、pi.userKeyList パラメーターは、ユーザープロパティを提供します。 また、pi.envKeyList パラメーターは、環境の詳細を提供します。 各パラメーターには、コンマで区切られたプロパティ値があります。

トリガーの作成

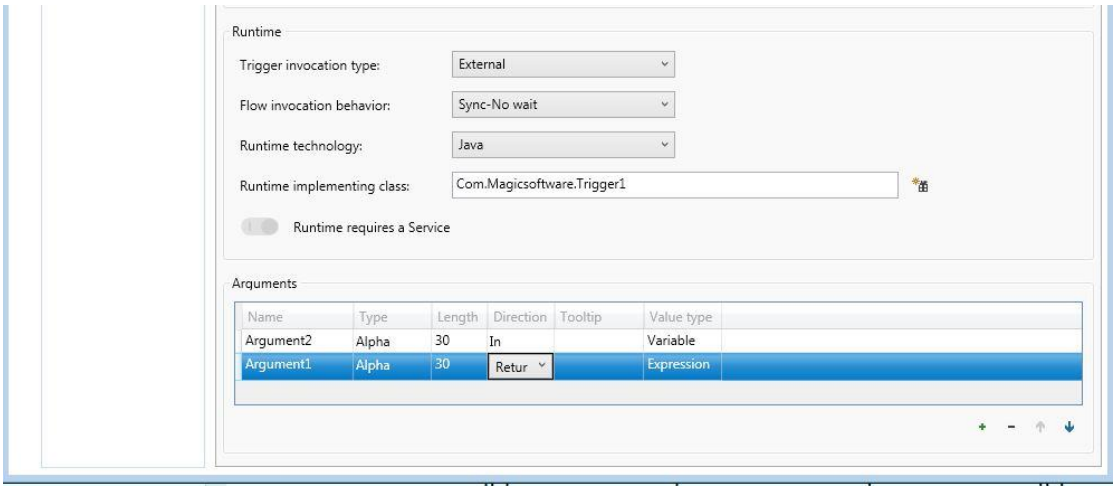
静的 UI を使用した外部トリガーの作成

この例では静的 UI を使用して外部トリガーを作成します。

外部トリガーとは、フローの呼び出しが、ポーリングメカニズムではなく、トリガー自体によって制御されることを意味します。 外部トリガーは、コールバックを含む実装に適しています。 例えば、メッセージを受け取ったメッセージングインフラストラクチャはコールバックメソッドを呼び出し、このメソッドからフローが呼び出されます。

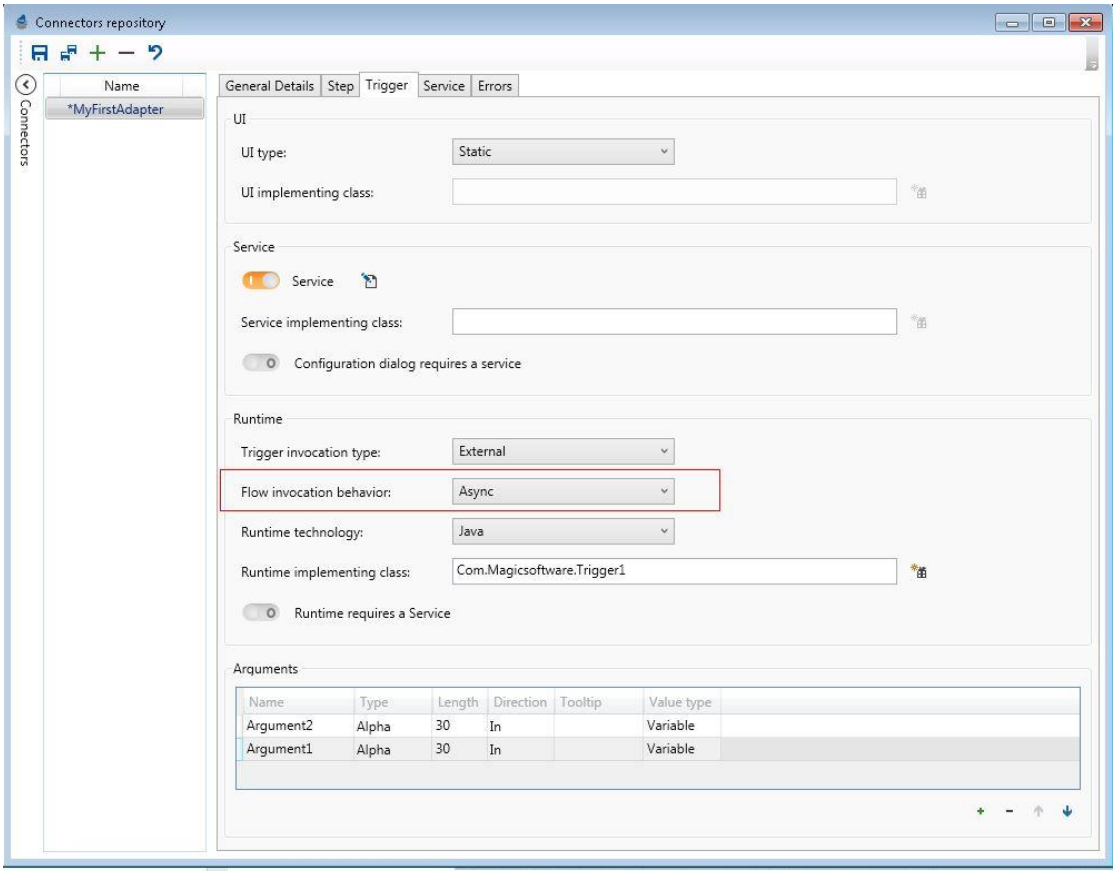
静的 UI とは、UI がコネクタビルダーから次のような方向で構成された単純なテーブルであることを意味します。:

1. In – トリガーランタイムコードからフローに渡されるプロパティを表します。これらのプロパティは、呼び出されたフロー内の変数にマップされる必要があるため、ピックリストタイプとして定義されます。
2. Return – フローから返された値を表し、トリガーに戻されます。静的 UI を使用する場合、その型が変数または式のいずれかである場合、単一の戻り値のみが可能です。



後で、動的 UI について説明するときに、構成にのみ使用されるデータクラスに別の種類のプロパティを渡すことができることがわかります。このタイプの変数には特別なアノテーションがあり、load () メソッドが呼び出されたときに実行時に一度評価され、フローに渡されることはありません。

トリガーの設定



フロー起動動作(Flow invocation behavior) コンボボックスに注意してください。外部トリガーの場合、以下の呼出タイプを使用できます。:

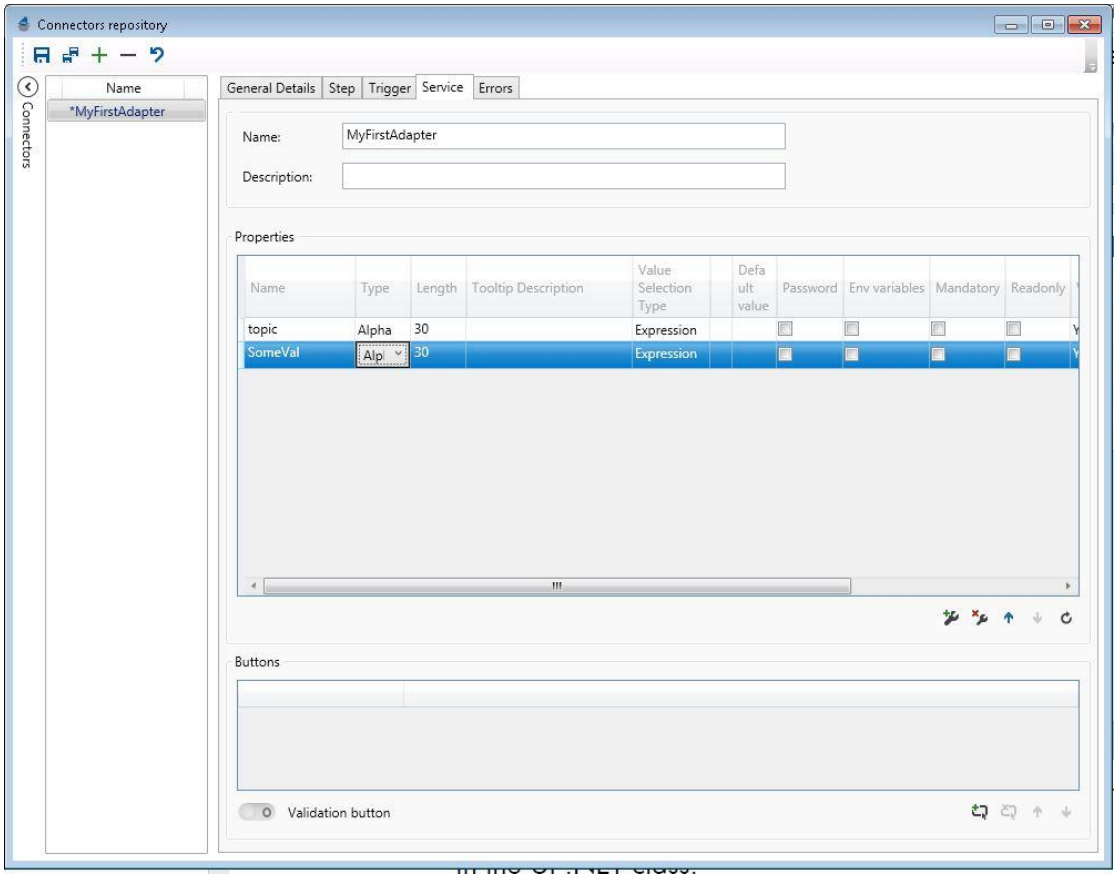
- **同期-待機(Sync-wait)** – フロー呼び出しは同期的で、フローが定義された最大インスタンス値内にある場合、メッセージは待機します。
- **同期-非待機(Sync-no wait)** – フロー呼び出しは同期していますが、フローが定義された Max インスタンス内にある場合はメッセージは待機せず、エラーが発生した場合にはすぐに戻ります。
- **非同期(Async)** – フロー呼び出しは非同期です。 invoke メソッドは、メッセージがスペースに入ったら返され、フローが結果を返すのを待機しません。

同期起動は、TCP トリガーなどの戻り値を持つトリガーに適しています。

非同期起動は、戻り値のないトリガー(コールバック メソッドを持つメッセージングトリガーなど)に適しています。

サービスの設定

トリガーはサービスで設定することができます。コネクタビルダーでサービス定義を作成します。



標準のサービスフィールドに加えて、検証ボタンと 3 つのアクションボタンを定義することができます。検証およびアクションボタンの実装は、UI .NET クラスにあります。

動的インターフェイスを持つトリガーの場合、実装メソッドは既に **IUserTriggerComponent** インターフェイスの一部になっています。

静的インターフェイスを持つトリガーの場合、ユーザは、サービスメソッドのみを含む専用インターフェイスを実装する必要があります。このインターフェイスを実装するクラス名は、コネクタビルダーの **Service implementing class(サービス実装クラス)**プロパティで定義されます。

メソッドは以下の通りです:

- public bool ValidateService(IReadOnlyServiceConfiguration serviceData, out string errorMsg) – このメソッドは、サービスで 検証 ボタンがクリックされると呼び出されます。 このメソッドには、サービスプロパティのオブジェクトの読み取り専用コピーがあり、有効な場合は True を返し、無効の場合は False およびエラーメッセージを返します。

- public void InvokeServiceHelper(string helperID, IServiceConfiguration serviceData) –このメソッドは、サービスでアクションボタンの 1 つがクリックされると呼び出されます。 このメソッドは、クリックされたボタンの名前と更新可能なサービス プロパティのオブジェクトを受け取ります。 このメソッドから、開発者は通常、追加のダイアログボックスを開き、いくつかのサービス値を更新します。

外部トリガー – 動的 UI

UI コードの実装は、以下に示す相違点を持つ動的ステップと非常によく似ています。

データ クラス

[TriggerIn] – このアノテーションを使用して、トリガーの実行時ロジックからフローに渡る値を表します。プロパティの型は変数である必要があります。

[TriggerReturn] – このアノテーションは、フローからトリガーに戻る値を表すために使用します。この種類のプロパティは、変数または式のいずれかになります。

[UseForConfiguration] – このアノテーションを使用して、特定のプロパティが設定にのみ使用されることを示します。これらのプロパティは、load ()メソッドへの読み込み時にトリガーに渡され、フロー自体に渡されません。

アダプタ クラス

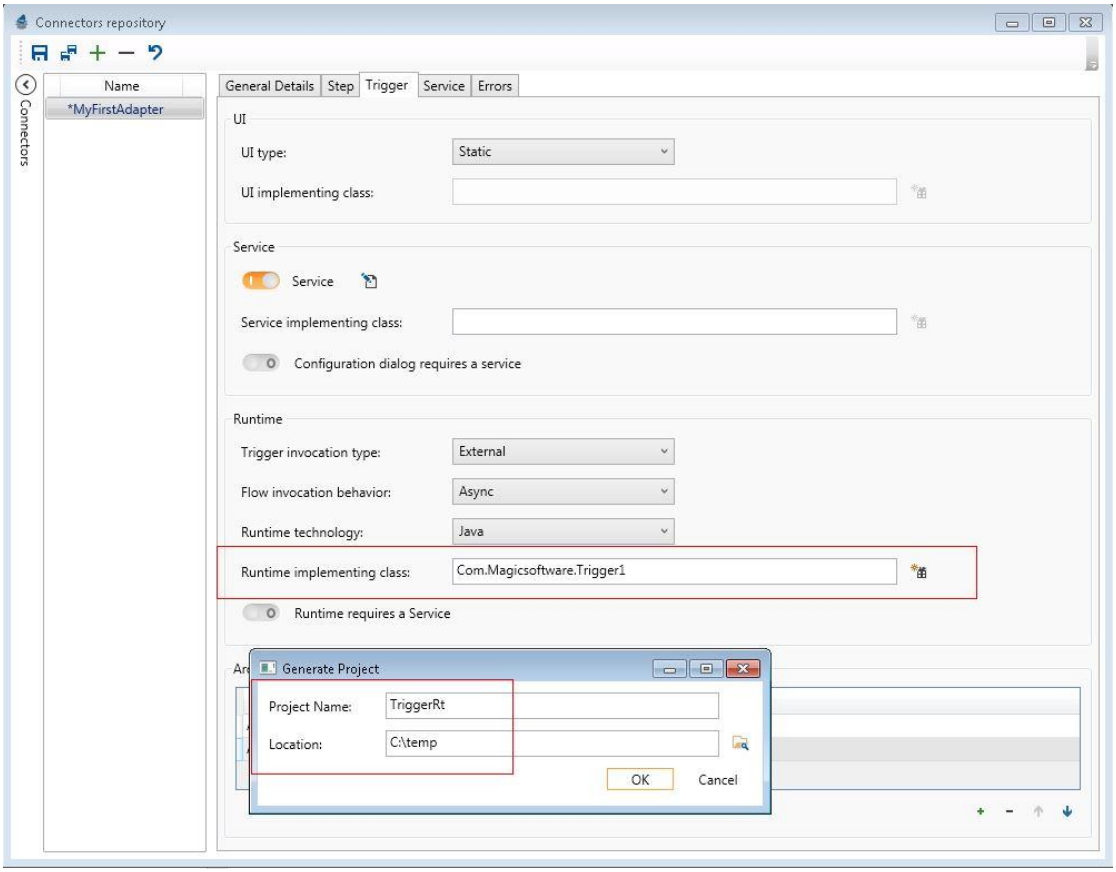
1. クラスは、IUserTriggerComponent インターフェイスを実装してエクスポートする必要があります。
2. Configure メソッドは動的ステップに非常によく似ていますが、Resource オブジェクトではなく Service オブジェクトに含まれます。
3. トリガーはデータマッパを使用しないため、スキーマ関連のメソッドはありません。

実装例

```
using System;
using System.Windows.Forms;
using MagicSoftware.Integration.UserComponents.Interfaces;
using MagicSoftware.Integration.UserComponents;
using System.ComponentModel.Composition;
using DynamicTrigger1;

namespace DynamicTrigger1
{
    [Export(typeof(IUserTriggerComponent))]
     class TriggerAdapter  : IUserTriggerComponent
    {
        public TriggerAdapter()
        {
        }

        #region IUserTriggerComponent implementation
        public object CreateDataObject()
        {
            return new TriggerDataClass();
        }
        public bool? Configure(ref object dataObject, ISDKStudioUtils utils, IReadOnlyServiceConfiguration
serviceData, object NavigateTo, out bool configurationChanged)
        {
            configurationChanged=true;
             TriggerDataClass triggerData = new TriggerDataClass();
            if (dataObject is TriggerDataClass)
                triggerData = (dataObject as TriggerDataClass);
            else
                dataObject = triggerData; // Set the reference to a new instance of the data class
            trigger2 dialog = new trigger2(triggerData,utils); // Open UI form
            DialogResult dr= dialog.ShowDialog();

            if(dr.Equals(DialogResult.OK))
            {
                configurationChanged=dialog.configHasChanged();
                return true;
            }

            return false;

        }
        public ICheckerResult Check(ref object data, IReadOnlyServiceConfiguration serviceData)
        {
            return null;
        }
        public bool ValidateService(IReadOnlyServiceConfiguration serviceData, out string errorMsg)
        {
            errorMsg="";
            return true;
        }
        public void InvokeServiceHelper(string helperID, IServiceConfiguration serviceData)
        {

        }
        #endregion
    }
}
```

外部トリガー – ランタイム

トリガーのランタイム部分の開発を開始する最も簡単な方法は、コネクタビルダーからテンプレートプロジェクトを生成することです。



定義されたランタイムクラスの横にある **ランタイムプロジェクトの生成(Generate runtime Project)** ボタンを使用して、ユーティリティを開きます。ユーティリティでは、.NET、Java および Magic xpa のプロジェクト名とその場所を入力します。このプロジェクトのネームスペースおよびパッケージ名は、先に入力した**ランタイム実装クラス(Runtime implementing class)**プロパティから設定されます。

この例では Java での実装を説明します。

1. Eclipse で、新しい Java プロジェクトを作成します。
2. プロジェクト名を設定します。
3. **既存のソースからプロジェクトを作成(Create project from existing source)**ラジオボタンを選択します。

4. **実行時プロジェクトの生成(Generate runtime Project)**ユーティリティで生成したプロジェクトのパスを選択します。
5. **完了(Finish)**をクリックし、プロジェクトの作成を終了します。
6. Eclipse でプロジェクトに移動し、クラス(コネクタビルダーで設定した名前)を開きます。
7. クラスは **IExternalTrigger** インターフェイスを実装し、次のメソッドを含みます。:

    - **load(TriggerGeneralParams triggergeneralparams, FlowLauncher flowlauncher)** – このメソッドは、トリガーのメインエントリポイントです。 このメソッドは、トリガーをロードするときにのみ呼び出されます。**FlowLauncher** オブジェクトは、開発者がフローを呼び出せるようにするロジックを保持します。**TriggerGeneralParams** オブジェクトは、トリガーに渡されたサービス、リソース、およびその他の設定を保持します。その時点から、ユーザは Magic xpi サーバが必要なときに disable()メソッドと enable()メソッドを呼び出せるように、新しいスレッドで独自のロジックを実装する必要があります。
    - **disable()** – フローを無効にする時に呼び出します。トリガーの無効化ロジックを実装するのは開発者の責任です。
    - **enable()** – フローを有効にする時に呼び出します。トリガーの有効化ロジックを実装するのは開発者の責任です。
    - **Unload()** – エンジンをシャットダウンする時に呼び出します。

*** FlowLauncher オブジェクトを使用してフローを呼び出す場合、呼び出しタイムアウトは既定で 90 秒に設定されます。**FlowLauncher** クラスの次のメソッドを使用して、タイムアウトを取得/設定することができます。:

public void setTimeOut(int timeoutSec)

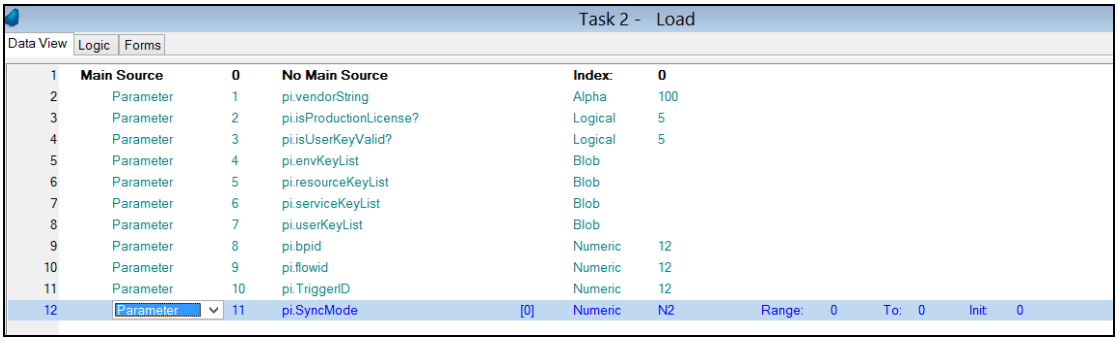
public int getTimeOut()

タイムアウト値を 0(Zero)に設定すると、タイムアウトはなくなります。

この例では、xpa 実装を使用します。

xpa スタジオを開きます。 その中で、ConnectorBuilder によって生成された xpa ソリューションを開きます。

Load という名前のプログラムがあります。 このプログラムは、トリガーの主要なエントリポイントです。 トリガーのロード時に 1 回だけ呼び出され、次のパラメーターを提供します。



次の関数を使用して、ユーザーとリソースのプロパティを取得および設定できます。

- getUserPropertyValueByKey(PropertyName)
- setUserPropValueA(PropertyName,Value)
- setUserPropValueN(PropertyName,Value)
- setUserPropValueD(PropertyName,Value)
- setUserPropValueT(PropertyName,Value)
- setUserPropValueL(PropertyName,Value)
- setUserPropValueB(PropertyName,Value)
- getResourceValueByKey(ResourceParameterName)

pi.envKeyList パラメーターは、環境の詳細を提供します。 pi.resourceKeyList パラメーターは、リソースパラメーター名を提供します。 pi.serviceKeyList パラメーターは、サービスパラメーター名を提供します。 pi.userKeyList パラメーターは、ユーザープロパティを提供します。 各パラメーターには、コンマで区切られたプロパティ値があります。

これ以降、ユーザーはカスタムロジックを実装できます。

着信引数をマップされた変数にマップするには、ユーザーはイベント MapInArguments を呼び出す必要があります。

フローを呼び出すには、ユーザーはイベント TriggerInvokeFlow を使用する必要があります。

return 引数をマップするには、ユーザーはイベント MapReturnArgument を呼び出す必要があります。

詳細については、コールプログラムのサンプルプロジェクトを参照してください。

監視トリガー

監視トリガーは、外部システムに対して事前に定義された間隔でデータを明示的に監視(ポーリング)するトリガーであり、データが見つかると、フローを呼び出します。監視トリガーの例としては、数秒ごとに新しい電子メールをチェックする Email トリガーと、指定した間隔毎にフォルダをスキャンする Directory Scanner トリガーがあげられます。

UI

トリガーの UI 部分は、外部トリガーのものと非常によく似ています。動的インターフェイスでは、同じ IUserTriggerComponent インターフェイスが使用されます。唯一の違いは、トリガーが使用できるプロパティのタイプです。監視トリガーは常に非同期です。フローが結果を返すのを待機しているクライアントはありません。したがって、データクラスの動的インターフェイスに対して定義できるプロパティには、次の 2 種類しかありません。:

[TriggerIn] – トリガーコードから渡された値をフローに渡します。

[UseForConfiguration] – 値は load()メソッドに渡され、フローに渡されることはありません。

静的 UI では、フローに渡された値を変数にマップする必要があるため、ピックリストタイプのみを選択できます。

ランタイム

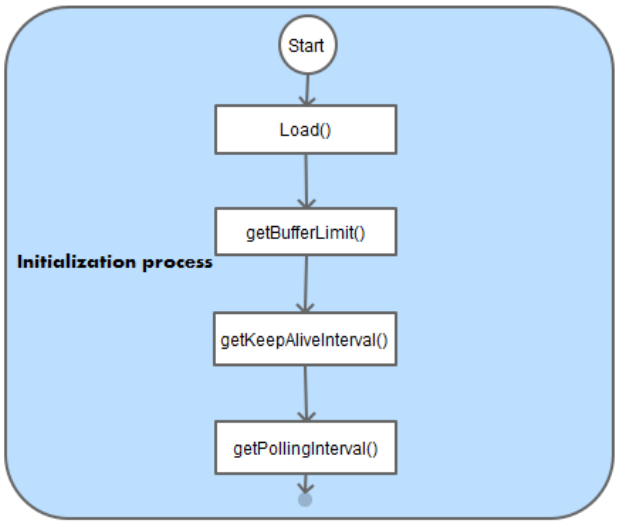
監視トリガーのランタイム部分は、外部トリガーと比較して少し複雑になります。監視トリガーでは、Magic xpi エンジンは、バッファサイズとキープアライブ間隔を強制確保するために、あらかじめ定義された間隔でトリガーを起動します。

監視トリガーで使用できる機能は次のとおりです。:

1. トリガーの監視(ポーリング)間隔を制御する – サーバはこの間隔でトリガーを呼び出します。
2. トリガーのキープアライブ間隔を制御する – トリガーは、応答しなかった場合に再起動されます。
3. バッファの制御 – トリガーによって作成されたメッセージの量がバッファ サイズを超えた場合、サーバはトリガーを呼び出しません。
4. 非常に大きなペイロードに対処するために、ペイロードを複数のフロー呼び出しに分割することができます。|

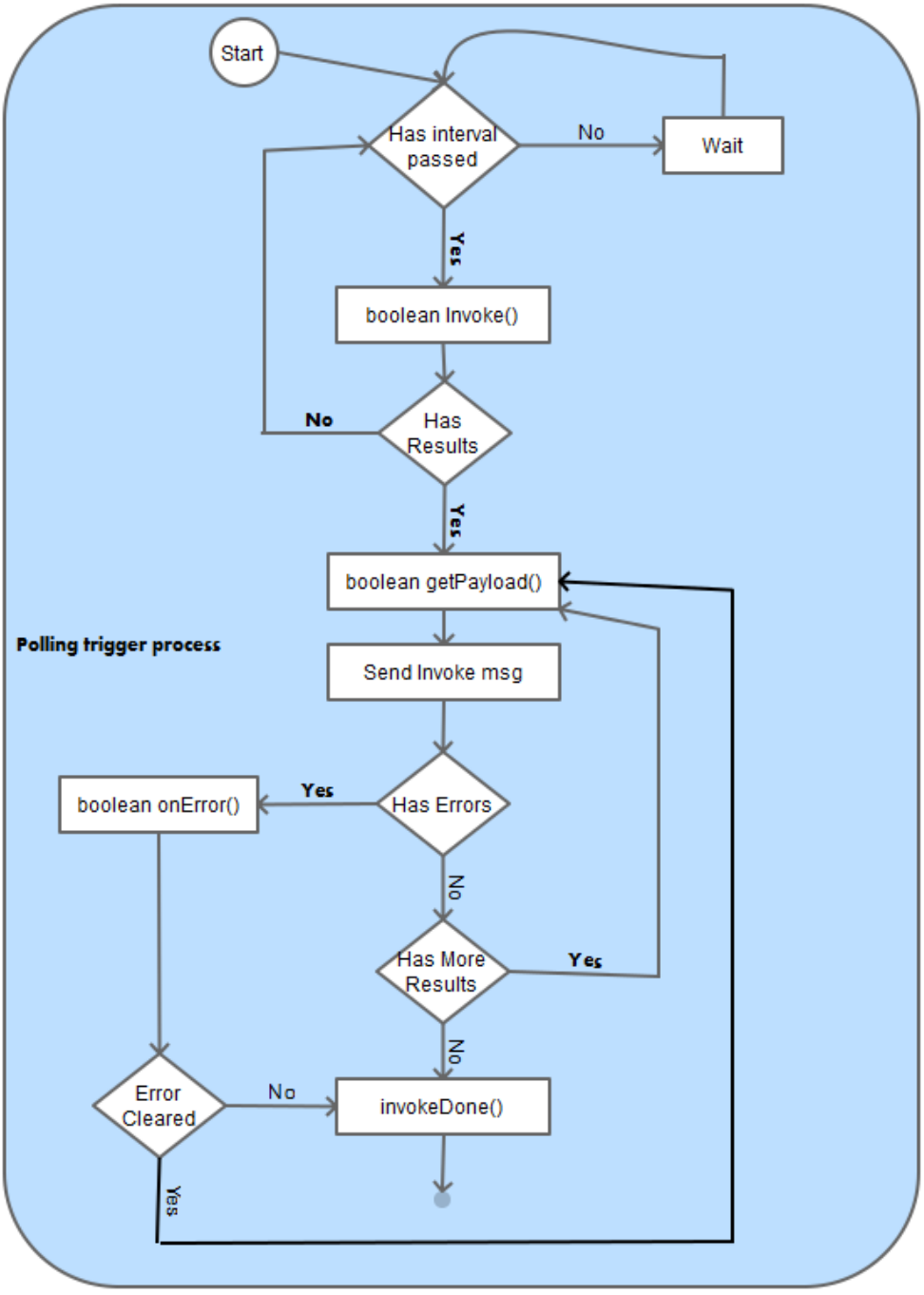
初期化

この部分は、Load()メソッドを構成オブジェクトとともに呼び出し、getBufferLimit()、getKeepAliveInterval()および getPollingInterval()を順番に呼び出します。 開発者はこれらのメソッドの実装を提供する必要があります。



監視シーケンス

この部分では、invoke()メソッドが各間隔毎に呼び出されます。 このフローを次の図で説明します。:



boolean Invoke() – このメソッドから、開発者はターゲットシステムにアクセスし、結果をポーリングする必要があります。このメソッドが True を返すと、呼び出しが結果とともに終了したことを示します(たとえば、新しい電子メールが見つかりました)。メソッドが False を返す場合、結果がないことを示します(例えば、新しい電子メールが見つからないなど)。

boolean getPayload(HashMap<String, Object> payload) –このメソッドは、invoke () メソッドが True を返すと呼び出されます。**ペイロードオブジェクトは** ref によって渡され、開発者は、フローに渡される必要がある引数を使用してデータを作成しなければなりません。

getPayload()メソッドを使用すると、開発者はペイロードの分割を実装できます。

メソッドが False を返すと、フロー リクエスト メッセージがフローに送信され、この呼び出しサイクルが終了します。 サイクルの終わりに、サーバは invokeDone()メソッドを呼び出して、開発者がキャッシュされた結果を解放できるようにします。

このメソッドが True を返すと、それ以上の結果があり、ペイロードが分割されたことを示します。 次に、サーバはフロー リクエスト メッセージを送信し、getPayload()を再度呼び出して次の分割を取得します。

getPayload()メソッドでエラーが発生した場合、onError()メソッドがサーバによって呼び出されます。 このメソッドの結果が True の場合、エラーは無視され、次の分割を取得するために getPayload()メソッドが再度呼び出されます。 結果が False の場合、サーバは invokeDone()メソッドを呼び出します。

エンドポイント トリガー

エンドポイントトリガーは、Magic xpi 4.6 で導入された新しいタイプのトリガーです。 このタイプのトリガーは、Magic xpi 以外でライフサイクルが管理されます。

このようなトリガーの例として、HTTP トリガーを挙げることができます。このトリガーは IIS によって管理され、要求が到着すると、prgname、appname トリガー名、およびエンドポイント名によって、到着している要求とそれを処理する特定のプロジェクト/フロー/トリガーの組み合わせが一意に決まります。

この新しいタイプのトリガーでは、Magic xpi は API クライアント ライブラリーを提供し、エンドポイント トリガーの開発者は以下の事を行うことができます。:

1. 使用可能なライセンス機能のスペースを確認
2. 一意のエンドポイント ID 値と名前/値のペア(呼び出しは同期または非同期である可能性があります)を指定して、フローを呼び出します。

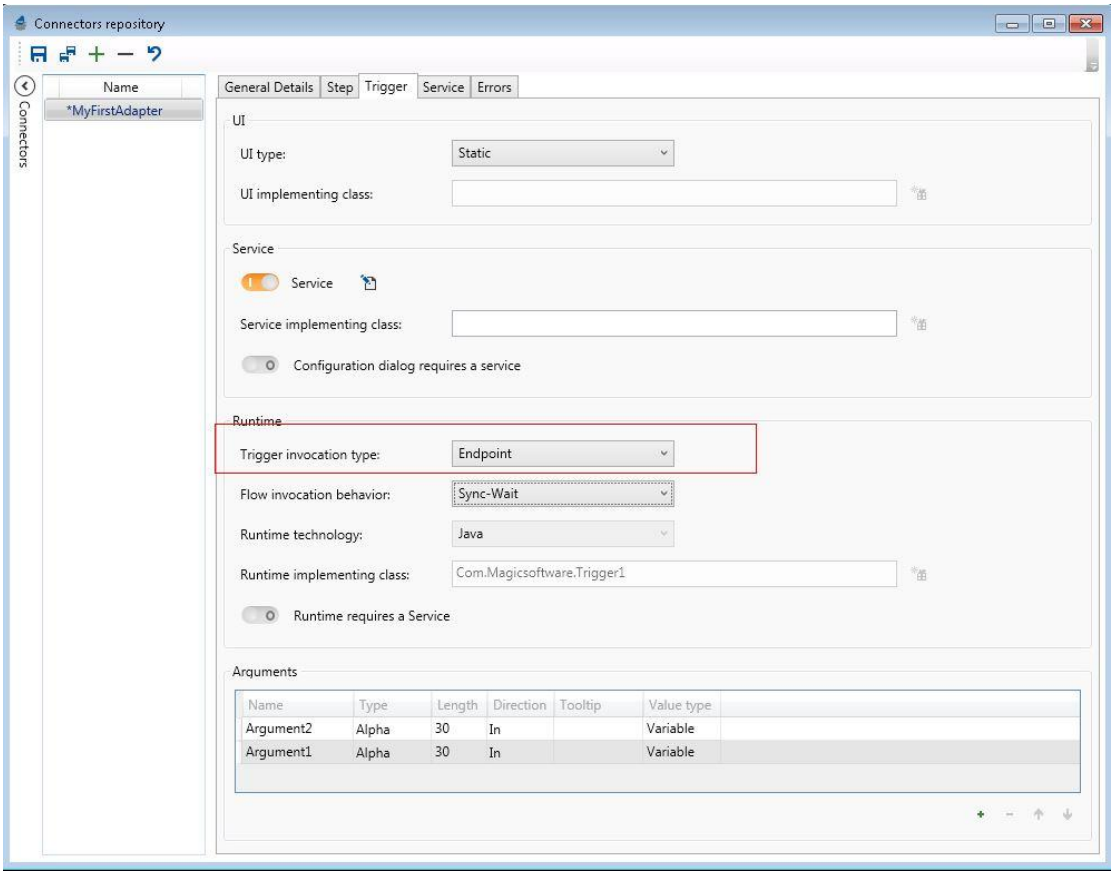
エンドポイント トリガーを使用する場合は、リクエストを特定のプロジェクト/フロー/トリガーの組み合わせに一致させるために固有の ID を使用する必要があります。 この一意の ID はエンドポイントと呼ばれます。

コネクタビルダーの設定

コネクタビルダーの**トリガー(Triggers)**タブで、**トリガー起動タイプ(Trigger invocation type)**を**エンドポイント(Endpoint)**に設定します。

このタイプのトリガーは、動的 UI と静的 UI の両方をサポートします。

エンドポイント トリガーには Magic xpi によって管理されるランタイムコードがないため、ランタイム実装クラスを提供するオプションはありません。



静的 UI

エンドポイント トリガーの静的 UI は、追加されたエンドポイント フィールドを除き、外部トリガーの静的 UI に似ています。トリガーをフローのトリガーエリアにドラッグする場合、開発者は実行時にこのトリガー インスタンスを識別する一意のエンドポイントを指定する必要があります。エンドポイント トリガーからフローを呼び出すときに、同じエンドポイントを送信する必要があります。

動的 UI

動的 UI の場合、コードは外部または監視トリガーのコードに似ています。 唯一の違いは、DataClass で定義する必要のあるエンドポイントの追加です。

まず、クラスレベルで、どのプロパティがエンドポイント値を保持しているかを宣言するために、アノテーションを使用する必要があります。:

```
[EndPointProperty("MyTriggerEndpoint")]
public class MyData
{
```

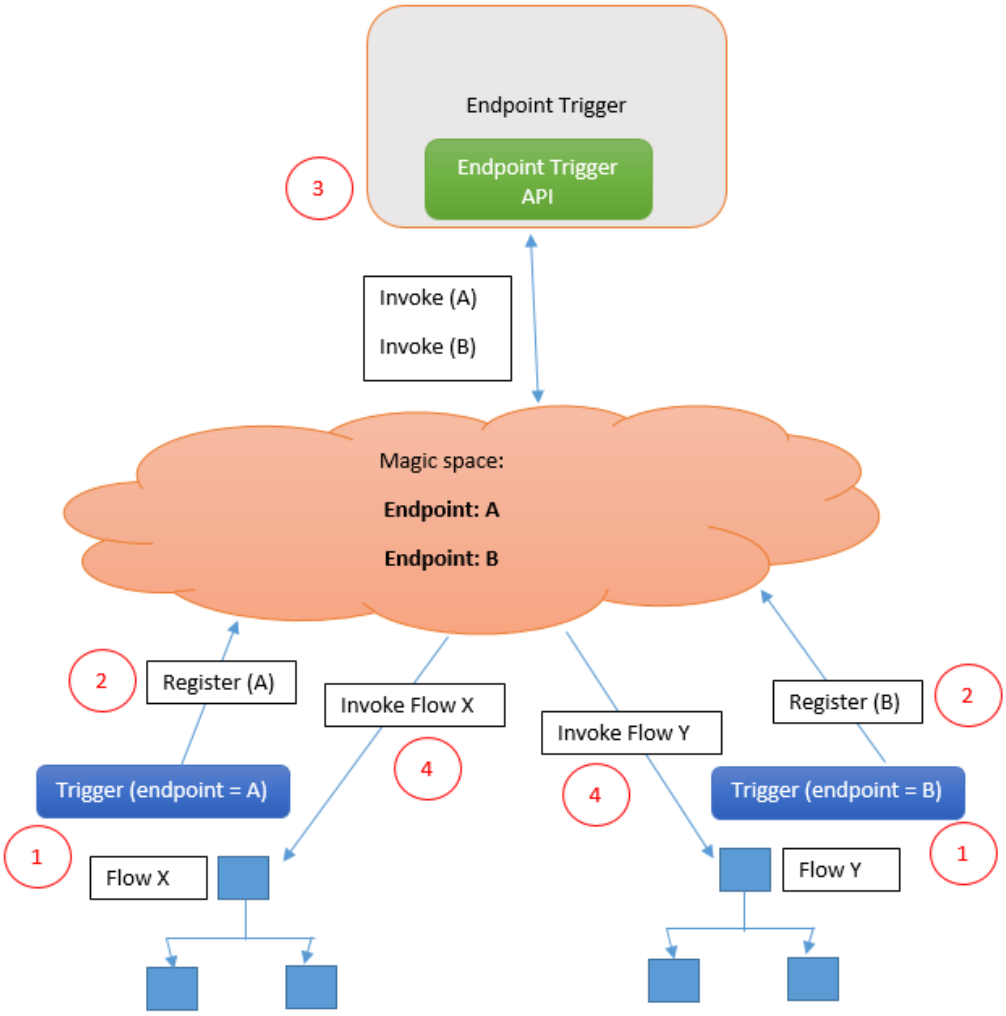
次に、このプロパティをクラスメンバーの 1 つとして定義する必要があります。プロパティは、文字型の IN プロパティである必要があります。:

```
//This TriggerEndpoint property is only relevant for Endpoint triggers. The property must hold the unique endpoint of the trigger
 [Id(4)]
 public Alpha MyTriggerEndpoint { get; set; }
```

このエンドポイントプロパティは、ユーザがスタジオでトリガーを設定した時に値が設定されていなければなりません。 そうでない場合、チェッカーでエラーが発生します。

コネクタビルダー ユーティリティから生成されるテンプレートには、既にエンドポイント トリガーに必要なプロパティが含まれています。

オペレーション図



1. 設計時 – トリガーがフローにドラッグされ、一意のエンドポイントが定義されます。

2. 実行時 – プロジェクトが開始されると、全てのエンドポイント トリガーが一意のエンドポイント値とプロジェクト/BP/フロー/TriggerID のアドレス指定によりスペースに登録されます。

3. 実行時 – トリガーは invoke メソッドを呼び出し、固有のエンドポイント ID とトリガーが期待する引数を渡します。 この操作の結果は、スペース内のエンドポイント トリガー オブジェクトです。

4. 実行時 – スペース内の処理ユニット(PU)は、新しいオブジェクトで定義されたエンドポイントと登録済みのエンドポイントとの間で照合を試みます。一致するものが見つかった場合、PU は、特定のプロジェクト/BP/フロー/トリガーID とトリガーによって送信されたプロパティのアドレスを持つフロー リクエスト メッセージを作成します。このプ

ロジェクトに属するワーカーは、フローリクエストメッセージを受け取り、フローを実行します。

Magic xpi エンドポイント トリガーAPI の使用

Magic xpi は、トリガー インスタンスを呼び出すためにエンドポイント トリガー実装で使用されるクライアント ライブラリを提供します。

必要なすべてのファイルを含む Standalone Invoker.zip ファイルは、Runtime \Support フォルダ内にあります。

API を使用する場合は、API が全ての参照を見つけることができるようにフォルダ構造を保持する必要があります。

mgreq.ini ファイルは、スペースの LookupLocators と LookupGroup を使用して設定する必要があります。

トリガー コードの例

次のサンプル Java コードは、Endpoint_Unique_Id エンドポイントの呼び出しを示しています。 さらに、このコードは、特定のライセンスがロードされたかどうか(トリガーMYTRG)、およびサーバ ライセンスがプロダクションライセンスであるかどうかをトリガーが確認する方法を示しています。

IDE を設定する際は、Standalone Invoker.zip ファイルで使用できる Java および lib フォルダのすべての jar をクラスパスに追加する必要があります。 さらに、log4j jar ファイルを追加する必要があります。log4j jar ファイルは zip には含まれていません。

```
package com.magicsoftware.pm.TriggerXpi;

import java.io.IOException;
import java.util.HashMap;
import com.magicsoftware.xpi.sdk.standalone.MagicxpiProxy;
import com.magicsoftware.xpi.sdk.standalone.MagicxpiProxy.ConnectionStatus;
import com.magicsoftware.xpi.sdk.standalone.license.ComponentLicense;
import com.magicsoftware.xpi.server.messages.StandaloneResponse;

public class Trigger1 {

          /**
           * @param args
           */
          public static void main(String[] args) {
                    try {

                              // create the proxy instance with connection properties from the
                              // provided mgreq.ini
                              MagicxpiProxy magicxpiproxy = new MagicxpiProxy(
                                                  "C:/temp/mgreq.ini");

                              // connect the proxy to the space
                              ConnectionStatus connectionstatus = magicxpiproxy.Connect();
                              if (connectionstatus.equals(ConnectionStatus.CONNECTED)) {
                                        // check if Magic xpi was loaded with the provided license
                                        // feature
                                        ComponentLicense componentlicense = magicxpiproxy
                                                            .getLicense("MYTRG");
                                        if (componentlicense != null
                                                            && componentlicense.isValidLicense()) {
                                                  System.out.println("Vendor String = "
                                                                      + componentlicense.getVendorString());
                                                  System.out.println("is production license = "
                                                                      + componentlicense.isProduction());
                                                  System.out.println("Is Valid license = "
                                                                      + componentlicense.isValidLicense());
                                        }
                                        // building the payload that will be sent when triggering the
                                        // flow
```

```
                                        // The keys and values should match the properties defined for
                                        // the Endpoint trigger
                                        HashMap args = new HashMap();
                                        args.put("inputvalue", "My Input value");
                                        // double d = 2D;
                                        // args.put("_InNumeric", Double.valueOf(d));

                                        // Calling the trigger with the trigger's unique endpoint and
                                        // the trigger payload
                                        // the invoke operation timeout is taken from the mgreq.ini
                                        StandaloneResponse standaloneresponse = magicxpiproxy.invoke(
                                                        "Endpoint_Unique_Id", args);
                                        if (standaloneresponse != null) {
                                                // Getting the variables returning from the flow - only
                                                // relevant for sync invocation
                                                HashMap responseVars = standaloneresponse.getVariables();
                                                String result1 = new String((byte[]) responseVars
                                                                .get("resultvalue"));
                                                System.out.println(result1);
                                        }

                                } else {
                                        // connection has failed
                                }
                        } catch (IOException ioexception) {
                                ioexception.printStackTrace();
                        } catch (Exception exception) {
                                exception.printStackTrace();
                        }
                }
        }
```

付録 A – ステップ UI プロジェクトの手動作成

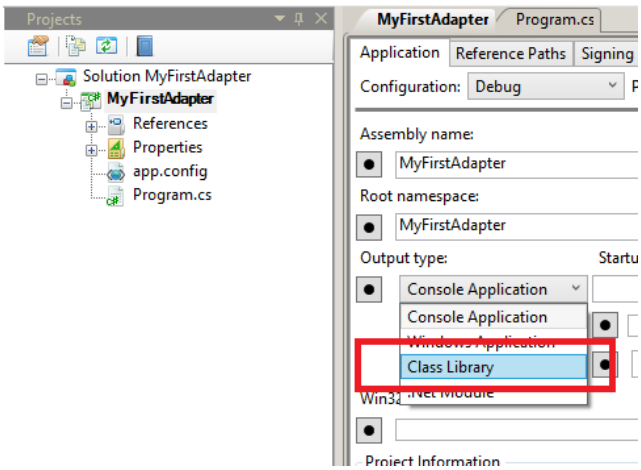
以下の手順は、C#プロジェクト用の IDE を作成/構成するために必要な手作業の手順です。 他の方法としては、UI クラス名の隣にある **UI プロジェクト生成(Generate UI project)**ユーティリティを使用する方法があります。

手動ステップ

SharpDevelop を C# IDE として使用します。ここからダウンロードします。:

http://www.icsharpcode.net/OpenSource/SD/Download/

1. SharpDevelop を開き、**SDK_TEST_1** という名前の新しいソリューションを作成します。
2. テンプレートとして **Console Application** を選択し、**OK** をクリックします。
3. プロジェクトが作成されたら、プロジェクトのプロパティを開き、**出力タイプ(Output type)**を**クラスライブラリ(Class Library)**に変更します。



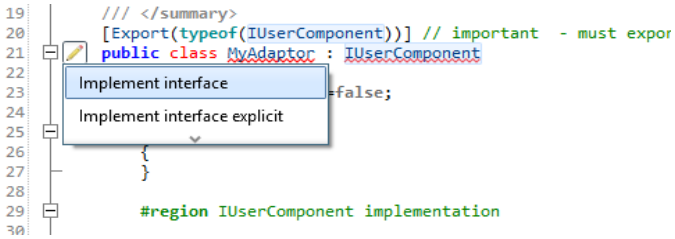
4. 既存の **program.cs** を削除します。
5. **MyStepAdapter**.という名前の新たなクラスを追加します。

ここでリファレンスを追加します。:

6. **GAC** タブで以下の項目を追加します。:
    a. System.Data.DataSetExtensions
    b. System.ComponentModel.Composition
7. **.NET Assembly Browser** タブから:
    a. **<Magic xpi root>\Studio\Extensions\Application** を参照します。

b. Select MagicSoftware.Integration.UserComponents.dll を選択します。

8. IUserComponent インターフェイスを実装します。 (public class SDK_TEST_1 : IUserComponent
9. クラスをエクスポートします: [Export(typeof(IUserComponent))].
10. 以下のインポートを追加します:
    a. using MagicSoftware.Integration.UserComponents.Interfaces;
    b. using MagicSoftware.Integration.UserComponents;
    c. using System.ComponentModel.Composition;
11. インターフェイス メソッドを実装します。(IDE には、必要なすべてのメソッドの空のスケルトンを作成するためのショートカットがあります。)



12. 実装されていない例外呼び出しを、自動作成したままにしないようにしてください。この呼び出しをそのままにすると、スタジオ例外が発生します。

Magic Software Enterprises について

Magic Software Enterprises (NASDAQ: MGIC) empowers customers and partners around the globe with smarter technology that provides a multi-channel user experience of enterprise logic and data.
We draw on 30 years of experience, millions of installations worldwide, and strategic alliances with global IT leaders, including IBM, Microsoft, Oracle, Salesforce.com, and SAP, to enable our customers to seamlessly adopt new technologies and maximize business opportunities.
For more information, visit www.magicsoftware.com.

OUTPERFORM THE FUTURE™