# Red Hat Fuse 7.12

## Fuse Online でのアプリケーションの統合

異なるアプリケーションやサービス間でデータを共有する統合を作成する

Last Updated: 2023-11-25

# Red Hat Fuse 7.12 Fuse Online でのアプリケーションの統合

異なるアプリケーションやサービス間でデータを共有する統合を作成する

## 法律上の通知



## 概要

Fuse Online はインテグレーションをサービスとして提供します。

# 目次

# 前書き

Red Hat Fuse は、分散型のクラウドネイティブな統合ソリューションで、選択できる多くのアーキテクチャー、デプロイメント、およびツールを提供します。Fuse Online は Red Hat の web ベースの Fuse ディストリビューションです。Syndesis は Fuse Online のオープンソースプロジェクトです。Fuse Online は OpenShift Online、OpenShift Dedicated、および OpenShift Container Platform で実行されます。

本ガイドでは、Fuse Online の Web インターフェイスを使用してアプリケーションを統合するための情報および手順を提供します。本書では以下の内容を取り上げます。

- 1章*Fuse Online の概要*
- 2章 *インテグレーション作成の準備方法*
- 3章*統合するアプリケーションへのコネクション*
- 4章 *インテグレーションの作成*
- 5章*REST API 呼び出しによってトリガーされるインテグレーションの作成*
- 6章*HTTP リクエスト (Webhook) によってトリガーされるインテグレーションの作成*
- 7章 *インテグレーションデータを次のコネクションのフィールドにマッピング*
- 8章 *インテグレーションの管理*
- 9章*Fuse Online のカスタマイズ*

サンプルインテグレーションを作成して Fuse Online を使用する方法の詳細は、Fuse Online サンプルインテグレーションのチュートリアル を参照してください。

サポートを利用するには、Fuse Online の左ナビゲーションパネルで **Support** をクリックするか、右上の  をクリックしてから **Support** を選択 します。

# 多様性を受け入れるオープンソースの強化

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

# 第1章 FUSE ONLINE の概要

Fuse Online では、アプリケーションやサービスからのデータを取得し、必要な場合はそのデータで操作することができます。さらに、データを全く異なるアプリケーションやサービスに送信することもできます。そのためにコーディングは必要ありません。

Fuse Online の概要は以下のトピックで取り上げます。

- 「Fuse Online の仕組み」
- 「Fuse Online の対象ユーザー」
- 「Fuse Online を使用する利点」
- 「Fuse Online コンストラクトの説明」

## 1.1. FUSE ONLINE の仕組み

Fuse Online は、コードを作成せずに複数の異なるアプリケーションやサービスの統合を可能にする Web ブラウザーインターフェイスを提供します。また、複雑なユースケースで必要な場合にコードを導入できる機能も提供します。

Fuse Online では、異なるアプリケーション間でのデータ転送を有効にできます。たとえば、ビジネスアナリストは Fuse Online を使用して、お客様がメンションされているツイートをキャプチャーし、Twitter から取得したデータを利用して Salesforce アカウントを更新できます。別の例として、株式取引の提案を行うサービスが挙げられます。Fuse Online を使用すると、対象株式の売買に関する提案をキャプチャーし、この提案を株式移転を自動化するサービスに転送することができます。

シンプルなインテグレーションを作成および実行するため主なステップは次のとおりです。

1. 統合する各アプリケーションへのコネクションを作成します。
2. 最初のコネクションを選択します。これは、別のアプリケーションと共有するデータが含まれるアプリケーションへのコネクションです。
   または、HTTP リクエストを受け入れるタイマーまたは Webhook でインテグレーションを開始できます。
3. 最後のコネクションを選択します。これは、最初のコネクションからデータを受け取ってインテグレーションを完了するアプリケーションへのコネクションです。
4. 最初のコネクションのデータフィールドを最後のコネクションのデータフィールドにマップします。
5. インテグレーションに名前を付け、必要に応じて説明、ラベル、およびカスタム環境変数を指定します。
6. **Publish** をクリックし、インテグレーションの実行を開始します。

この他のインテグレーションの 1 つに API プロバイダーインテグレーションがあります。API プロバイダーインテグレーションでは、REST API クライアントはインテグレーションの実行をトリガーするコマンドを呼び出しできます。API プロバイダーインテグレーションを作成および実行するには、OpenAPI 3 (または 2) ドキュメントを Fuse Online にアップロードします。このドキュメントによって、クライアントで呼び出しできるオペレーションが指定されます。オペレーションを実行する、コネクションやステップ (データマッパーステップやフィルターステップなど) のフローを、各オペレーショ

ンに対して指定および設定します。シンプルなインテグレーションごとに 1 つのプライマリーフローがあります。API プロバイダーインテグレーションの場合は、オペレーションごとに 1 つのプライマリーフローがあります。

Fuse Online のダッシュボードでは、インテグレーションを監視および管理できます。実行しているインテグレーションを確認でき、インテグレーションを開始、停止、および編集できます。

## 1.2. FUSE ONLINE の対象ユーザー

Fuse Online の対象ユーザーは、異なるアプリケーション間でデータを共有するためにコードを作成したくないビジネスユーザー (財務、人事、マーケティングなどの関係者) です。このようなユーザーは、さまざまな SaaS (Software-as-a-Service) を使用することにより、ビジネス要件、ワークフロー、および関連データについて理解します。

ビジネスユーザーは Fuse Online を使用して以下を行うことができます。

- 所属企業の名前がメンションされたツイートをキャプチャーし、それが知らないソースからのツイートである場合は Salesforce 環境で新しい連絡先を作成します。
- Salesforce のリード更新を特定し、SQL ストアドプロシージャーを実行して、関連するデータベースを最新の状態に維持します。
- AMQ ブローカーが受け取るオーダーをサブスクライブし、カスタム API でこれらのオーダーを操作します。
- Amazon S3 バケットからデータを取得し、Dropbox フォルダーに追加します。

これは、ビジネスユーザーがコードを作成せずに行える例の一部です。

## 1.3. FUSE ONLINE を使用する利点

Fuse Online には多くの利点があります。

- コードを作成せずに異なるアプリケーションやサービスからデータを統合します。
- パブリッククラウドの OpenShift Online や、オンサイトの OpenShift Container Platform でインテグレーションを実行できます。
- ビジュアルデータマッパーを使用して、あるアプリケーションのデータフィールドを別のアプリケーションのデータフィールドにマップします。
- オープンソースソフトウェアの利点をすべて活用します。機能の拡張やインターフェイスのカスタマイズが可能です。統合するアプリケーションやサービスのコネクターを Fuse Online が提供しない場合は、開発者が必要なコネクターを作成できます。

## 1.4. FUSE ONLINE コンストラクトの説明

Fuse Online を使用するには、コネクター、コネクション、アクション、ステップ、およびフローを使用してインテグレーションを作成します。これらのコンストラクトの基本を理解すると便利です。

Fuse Online の各インストールは Fuse Online 環境と呼ばれます。Red Hat が Fuse Online 環境をインストールおよび管理する場合、OpenShift Online または OpenShift Dedicated 上で実行されます。ユーザーが Fuse Online 環境をインストールおよび管理する場合、通常は OpenShift Container Platform で実行されますが、OpenShift Dedicated で実行することもできます。

### インテグレーション

Fuse Online には、シンプルなインテグレーションと API プロバイダーインテグレーションがあります。

シンプルなインテグレーションは、Fuse Online が実行する順序付けされたステップです。これには、以下が含まれます。

- アプリケーションに接続し、インテグレーションを開始するステップ。このコネクションは、インテグレーションが操作する初期データを提供します。後続のコネクションは追加のデータを提供できます。
- アプリケーションに接続し、インテグレーションを完了するためのステップ。このコネクションは、以前のステップから出力されたデータをすべて受け取り、インテグレーションを終了します。
- 最初と最後のコネクションの間にアプリケーションに接続する任意の追加ステップ。追加のコネクションがどのインテグレーションステップにあるかによって、追加のコネクションは以下のいずれかまたはすべてを実行できます。
  - 操作するインテグレーションの追加データを提供します。
  - インテグレーションデータを処理します。
  - 処理結果をインテグレーションに出力します。
- コネクションとアプリケーション間のデータで操作する任意のステップ。通常、前のコネクションのデータフィールドを次のコネクションが使用するデータフィールドにマップするステップがあります。

API プロバイダーインテグレーションは、OpenAPI スキーマを提供した REST API サービスをパブリッシュします。REST API クライアントからの呼び出しによって、API プロバイダーインテグレーションの実行がトリガーされます。呼び出しは、REST API が実装するすべてのオペレーションを呼び出すことができます。シンプルなインテグレーションには実行のプライマリーフローが 1 つあります。API プロバイダーインテグレーションの場合は、オペレーションごとに 1 つのプライマリーフローがあります。各オペレーションフローは、インテグレーションの作成時にそのオペレーションのフローに追加したステップに応じて、アプリケーションに接続し、データを処理します。各オペレーションフローは、呼び出しがインテグレーションの実行をトリガーしたクライアントに指定する応答を返して終了します。

### コネクター

Fuse Online はコネクターのセットを提供します。コネクターは、データの取得元また送信先となる特定のアプリケーションを示します。各コネクターは、その特定のアプリケーションへのコネクションを作成するためのテンプレートです。たとえば、Salesforce コネクターを使用して、Salesforce へのコネクションを作成します。

接続するアプリケーションは、OAuth プロトコルを使用してユーザーを認証することがあります。この場合、Fuse Online 環境をそのアプリケーションにアクセスできるクライアントとして登録します。登録は、アプリケーションのコネクターに関連付けられます。OAuth を使用する各アプリケーションに 1 度だけ特定の Fuse Online 環境を登録する必要があります。登録は、コネクターから作成する各コネクションまで適用されます。

Fuse Online が必要なコネクターを提供しない場合、開発者が必要なコネクターを作成できます。

### コネクション

インテグレーションを作成する前に、データの取得元また送信先となる各アプリケーションまたはサー

ビスへのコネクションを作成する必要があります。コネクションを作成するには、コネクターを選択して、設定情報を追加します。たとえば、インテグレーションで AMQ ブローカーに接続するには、AMQ コネクターを選択してコネクションを作成し、プロンプトにしたがって接続するブローカーと、コネクションに使用するアカウントを特定します。

コネクションは、作成されたコネクターの特定のインスタンスの 1 つです。1 つのコネクターから複数のコネクションを作成できます。たとえば、AMQ コネクターを使用して、各コネクションが異なるブローカーにアクセスする 3 つの AMQ コネクションを作成できます。

シンプルなインテグレーションを作成するには、インテグレーションを開始するコネクションとインテグレーションを終了するコネクションを選択します。また任意で、追加のアプリケーションにアクセスするためのコネクションを 1 つ以上選択します。各オペレーションフローに 1 つ以上のコネクションを追加して API プロバイダーインテグレーションを作成できます。任意の数のインテグレーションおよびオペレーションフローが同じコネクションを使用できます。特定のインテグレーションまたはフローは、同じコネクションを複数回使用できます。

詳細は、統合するアプリケーションへのコネクション を参照してください。

### アクション

インテグレーションでは、各コネクションは必ず 1 つのアクションを実行します。インテグレーションの作成時に、フローに追加するコネクションを選択し、コネクションが実行するアクションを選択します。たとえば、Salesforce コネクションをフローに追加するときに、Salesforce アカウントの作成、Salesforce アカウントの更新、および Salesforce の検索が含まれ、それに限定されないアクションのセットから選択します。

一部のアクションには追加の設定が必要で、Fuse Online は必要な情報を要求します。

### ステップ

シンプルなインテグレーションは、順序付けされたステップのセットです。API プロバイダーインテグレーションでは、各オペレーションフローが順序付けされたステップのセットになります。

各ステップはデータで操作します。ステップによっては、Fuse Online 外部のアプリケーションまたはサービスに接続しながらデータ上で操作するものもあります。これらのステップはコネクションです。コネクションの間に、Fuse Online のデータで操作する他のステップがある場合があります。通常、ステップのセットには、前のコネクションで使用されたデータフィールドをフローの次のコネクションで使用されるデータフィールドにマップするステップが含まれます。シンプルなインテグレーションの最初のコネクション以外では、各ステップは前のステップから受け取るデータで操作します。

Fuse Online は、コネクションの間でデータを操作するため、以下を行うステップを提供します。

- あるアプリケーションのデータフィールドを別のアプリケーションのデータフィールドにマップします。
- データをフィルターし、処理中のデータがユーザー定義の基準と合ったときのみインテグレーションが継続されるようにします。
- レコードのコレクションを個別のレコードに分割して、Fuse Online が各レコードに対して 1 度後続のステップをイテレーティブに実行するようにします。
- 個別のレコードをコレクションに集計し、Fuse Online がコレクションに対して 1 度後続のステップを実行するようにします。
- Freemaker、Mustache、または Velocity テンプレートにデータを挿入して、同等で一貫性のある出力を生成します。
- Fuse Online が自動的に提供するデフォルトのログ以外の情報をログに記録します。

カスタムステップを提供するエクステンションをアップロードすると、Fuse Online に組み込まれない方法でコネクションの間のデータを操作できます。Fuse Online エクステンションの開発 を参照してください。

**フロー**

フローは、インテグレーションが実行する順序付けされたステップのセットです。

シンプルなインテグレーションには、1つのプライマリーフローがあります。API プロバイダーインテグレーションの場合は、REST API が定義するオペレーションごとに1つのプライマリーフローがあります。各オペレーションのプライマリーフローは、そのオペレーションの呼び出しを処理するステップのセットです。

プライマリーフローには条件付きフローを追加できます。インテグレーションは、指定した条件を評価し、関連付けられたフローを実行するかどうかを決定します。

フローでは、各ステップは前のステップから出力されたデータで操作できます。フローに必要なステップを判断するには、インテグレーションの計画に関する注意点 を参照してください。

# 第2章 インテグレーション作成の準備方法

インテグレーションを作成するためのワークフローを計画および理解すると、必要に応じたインテグレーションの作成に役立ちます。インテグレーション作成の準備に関する情報は、以下を参照してください。

- 「インテグレーションの計画に関する注意点」
- 「シンプルなインテグレーションを作成するための一般的なワークフロー」
- 「Salesforce からデータベースへシンプルなインテグレーションを作成するためのワークフローの例」

## 2.1. インテグレーションの計画に関する注意点

インテグレーションを作成する前に、以下を検討してください。

インテグレーションの実行をトリガーする方法。

- タイマーを設定して、指定した間隔で実行をトリガーするか。
- HTTP リクエストを送信するか。
- アプリケーションに接続してデータを取得するか。
  - そのアプリケーションで、データを取得するアクションを何がトリガーするか。たとえば、Twitter からデータを取得して開始するインテグレーションは Twitter のメンションでトリガーすることが可能です。
  - 対象となるデータフィールド。
  - このアプリケーションにアクセスするために Fuse Online が使用するクレデンシャル。
- オペレーションのフローの実行をトリガーする REST API 呼び出しをクライアントが実行できるよう、REST API サービスをパブリッシュするか。
  - サービスの OpenAPI スキーマがすでに定義されているか。
  - 定義されていない場合、サービスがどのオペレーションを定義するか。

シンプルなインテグレーションを終了するには、以下を検討します。

- データを受信するアプリケーションがあるか、または情報をインテグレーションのログに送信するか。
- アプリケーションにデータを送信する場合、インテグレーションが実行するアクションは何か。
- 対象となるデータフィールド。
- このアプリケーションにアクセスするために Fuse Online が使用するクレデンシャル。

フローのステップのセットでは、以下を検討します。

- 他のアプリケーションにアクセスする必要があるか。アクセスする必要がある他のアプリケーションに対して、以下を検討します。

- フローが接続する必要のあるアプリケーション。
- コネクションが実行するアクション。
- 対象となるデータフィールド。
- コネクションがこのアプリケーションに接続するために使用するクレデンシャル。

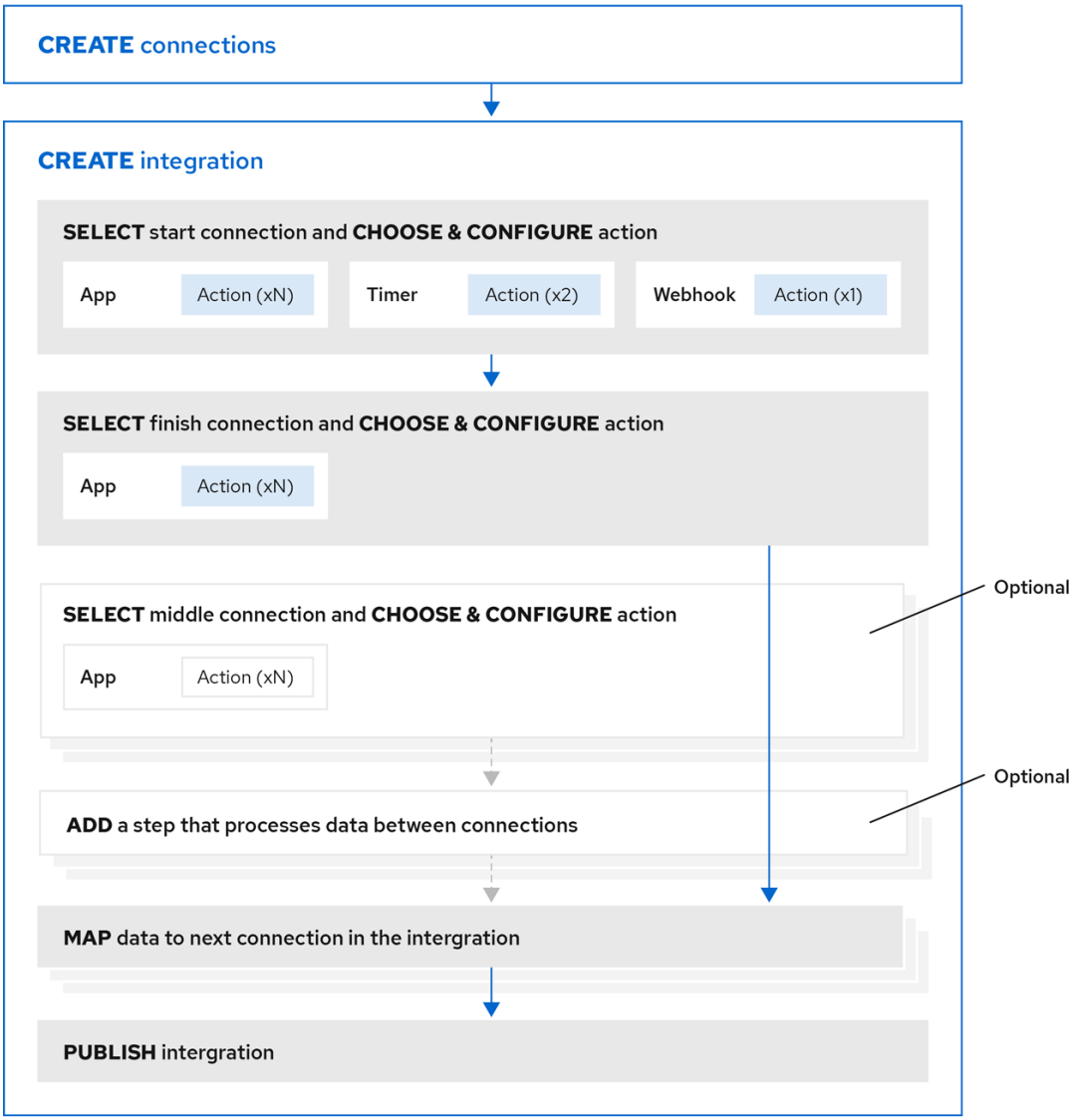
- フローがコネクション間のデータで操作する必要があるか。以下に例を示します。
  - フローは操作するデータをフィルターするか。
  - フィールド名がソースアプリケーションとターゲットアプリケーションで異なるか。異なる場合はデータマッピングが必要です。
  - フローがコレクションで操作するか。操作する場合、フローがデータマッパーを使用してコレクションを処理できるか、またはフローがコレクションを個別のレコードに分割する必要があるか。フローはレコードをコレクションに集約する必要があるか。
  - テンプレートはデータを一貫した形式で出力するのに便利であるか。
  - 処理中のメッセージに関する情報をインテグレーションのログに送信するか。
  - フローはカスタマイズされた方法にてデータで操作する必要があるか。
- インテグレーションデータの内容に応じて実行フローを変更する必要があるか。よって、条件付きフローが必要であるか。

## 2.2. シンプルなインテグレーションを作成するための一般的なワークフロー

Fuse Online コンソールにログインしたら、統合するアプリケーションへのコネクションを作成できます。統合する各アプリケーションと、OAuth プロトコルを使用する各アプリケーションでは、Fuse Online をそのアプリケーションのクライアントとして登録します。登録する必要のあるアプリケーションには以下が含まれます。

- Dropbox
- Google アプリケーション (Gmail、カレンダー、およびスプレッドシート)
- Salesforce
- SAP Concur
- Twitter

これらのアプリケーションを登録すると、シンプルなインテグレーションを作成するためのワークフローは次のようになります。



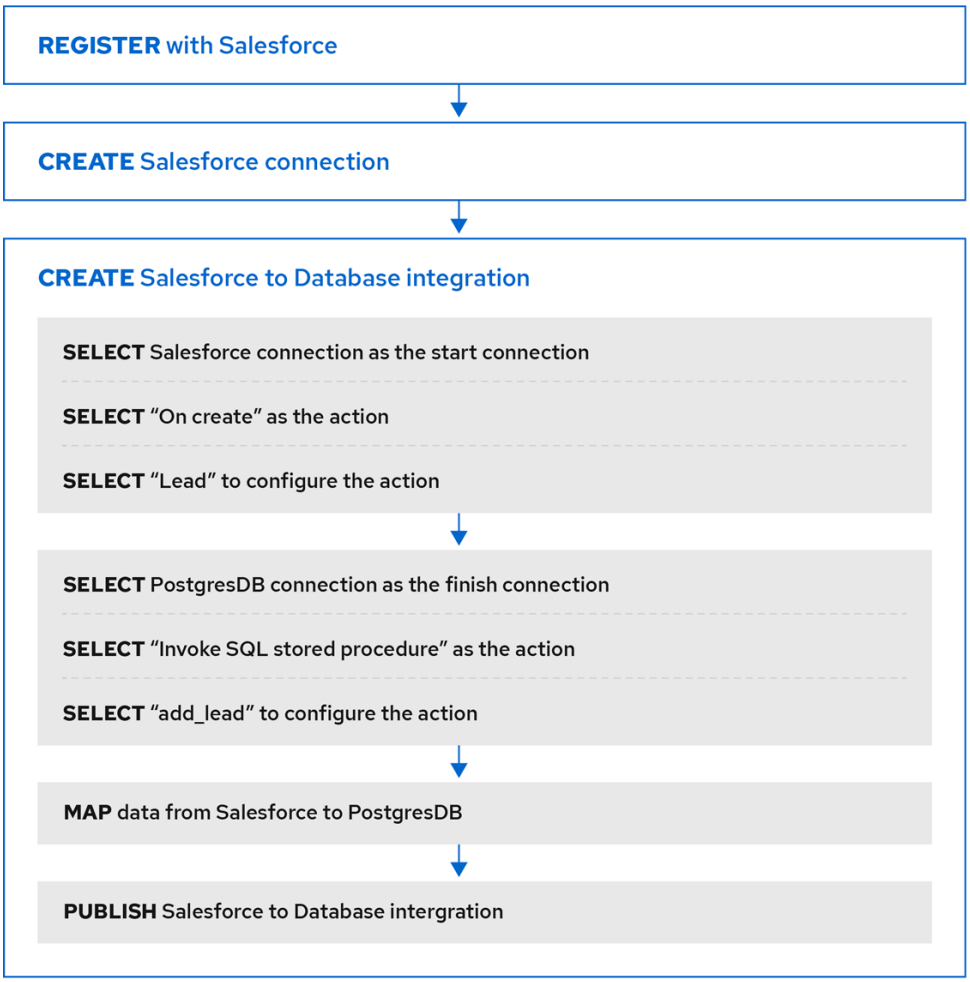

### その他のリソース

API プロバイダーインテグレーションを作成する利点、概要、およびワークフロー

## 2.3. SALESFORCE からデータベースへシンプルなインテグレーションを作成するためのワークフローの例

Fuse Online を使用してシンプルなインテグレーションを作成するためのワークフローを理解するには、サンプルインテグレーションのチュートリアル の手順にしたがってサンプルインテグレーションを作成するのが最も効果的です。

以下の図は、Salesforce からデータベースへのサンプルインテグレーションを作成するためのワークフローを表しています。




インテグレーションをパブリッシュした後、インテグレーションを実行する準備ができていれば Fuse Online ダッシュボードのインテグレーション名の横に **Running** が表示されます。

### その他のリソース

API プロバイダークイックスタートインテグレーションの例のインポートおよびパブリッシュ

# 第3章 統合するアプリケーションへのコネクション

統合するアプリケーションに接続するための主なステップは次のとおりです。

1. 統合する各アプリケーションまたはサービスへのコネクションを作成します。
2. 統合する各アプリケーションへのコネクションを持つインテグレーションを作成します。

コネクションの作成手順は、アプリケーションまたはサービスごとに異なります。各種のコネクションを作成し、特定のインテグレーション向けに設定するための詳細は、Fuse Online のアプリケーションおよびサービスへの接続 を参照してください。

コネクションの一般的な情報は以下を参照してください。

- 「Fuse Online からアプリケーションへのコネクションの作成」
- 「承認取得のための一般的な手順」
- 「コネクションの検証」
- 「インテグレーションへのコネクションの追加」
- 「コネクション情報の表示および編集方法」
- 「カスタムコネクターからコネクションを作成」

## 3.1. FUSE ONLINE からアプリケーションへのコネクションの作成

コネクションを作成するには、接続するアプリケーションのコネクターを選択し、入力フィールドに値を入力して、そのアプリケーションへのコネクションを設定します。提供する必要のある設定の詳細は、アプリケーションごとに異なります。コネクションの設定後、このコネクションと同じアプリケーションへの別のコネクションを区別できる名前を付けます。任意で、コネクションの詳細を指定できます。

同じコネクターを使用して、そのアプリケーションへのコネクションをいくつでも作成することができます。たとえば、AMQ コネクターを使用して 3 つのコネクションを作成することが可能です。各 AMQ コネクションは異なるブローカーを指定することが可能です。

例については、以下を参照してください。

- AMQ コネクションの作成
- HTTP および HTTPS コネクションの作成
- Slack コネクションの作成

## 3.2. 承認取得のための一般的な手順

インテグレーションでは、アクセス要求の認証に OAuth プロトコルを使用するアプリケーションに接続する場合があります。これには、そのアプリケーションにアクセスするために Fuse Online のインストールを登録する必要があります。登録により、Fuse Online インストールから該当するアプリケーションへのすべてのコネクションが承認されます。たとえば、Fuse Online インストールを Salesforce に登録する場合、Fuse Online インストールから Salesforce へのすべてのコネクションは、登録によって提供された同じ Salesforce クライアント ID と 同じ Salesforce クライアントシークレットを使用します。

Fuse Online 環境ごとに、OAuth を使用するアプリケーション 1 つにつき Fuse Online をクライアントとする 1 つの登録のみが必要です。この登録により、複数のコネクションを作成でき、コネクションごとに違うユーザークレデンシャルを使用できます。

接続する OAuth アプリケーションによって特定のステップは異なりますが、登録は常にクライアント ID とクライアントシークレットで Fuse Online 環境を提供します。アプリケーションによっては、クライアント ID およびクライアントシークレットに別のラベルを使用することがあります。たとえば、Salesforce はコンシューマーキーとコンシューマーシークレットを生成します。

OAuth アプリケーションによっては、登録が提供するクライアント ID およびクライアントシークレットを追加するため、Fuse Online が **Settings** ページにエントリーを提供することがあります。これに該当するアプリケーションを確認するには、Fuse Online の左パネルで **Settings** をクリックします。

**前提条件**

- Fuse Online の **Settings** ページに、OAuth プロトコルを使用してアクセスを承認するアプリケーションのエントリーがあります。

**手順の概要**

1. Fuse Online の **OAuth Application Management** ページで、Fuse Online を登録するアプリケーションのエントリーを展開します。これにより、クライアント ID およびクライアントシークレットのフィールドが表示されます。
2. **OAuth Application Management** ページの上部付近に表示されている **During registration, enter this callback URL:** の URL をクリップボードにコピーします。
3. 別のブラウザータブで、登録するアプリケーションの Web サイトに移動し、クライアント ID とシークレットを取得するのに必要なステップを実行します。これらのステップの 1 つで、Fuse Online 環境のコールバック URL を入力する必要があります。クリップボードにコピーした URL を 2 つ目のステップで貼り付けます。
4. Fuse Online の **Settings** ページにクライアント ID とクライアントシークレットを貼り付け、設定を保存します。

**その他のリソース**

- **Settings** ページにエントリーがあるアプリケーションを登録する例:
  - Fuse Online を Salesforce クライアントアプリケーションとして登録
  - Fuse Online を Twitter クライアントアプリケーションとして登録
- Fuse Online の **Settings** ページにエントリーのないアプリケーションに登録する例:Fuse Online を Dropbox クライアントとして登録
- OAuth プロトコルを使用するアプリケーションへのアクセスを許可するカスタムコネクターの使用に関する情報:カスタムコネクターからコネクションを作成

## 3.3. コネクションの検証

OAuth を使用するアプリケーションにアクセスするために Fuse Online の承認を取得した後、そのアプリケーションへのコネクションを 1 つ以上作成できます。OAuth アプリケーションへのコネクションを作成する場合、Fuse Online はそのコネクションを検証し、承認が取得されていることを確認します。いつでもコネクションを再度検証し、承認が有効であることを確認できます。

OAuth アプリケーションによっては、有効期限のあるアクセストークンを付与することがあります。アクセストークンの期限が切れた場合、アプリケーションに再接続して新しいアクセストークンを取得できます。

OAuth を使用するコネクションを検証するか、OAuth アプリケーションの新しいアクセストークンを取得するには、以下を実行します。

1. 左側のパネルで **Connections** をクリックします。
2. 検証するコネクション、または新しいアクセストークンを取得するコネクションをクリックします。
3. コネクションの詳細ページで、**Validate** または **Reconnect** をクリックします。

検証または再接続に失敗した場合は、アプリケーションまたはサービスプロバイダーを確認し、アプリケーションの OAuth キー、ID、トークン、またはシークレットが有効であるか判断します。アイテムが期限切れになったり、取り消された可能性があります。

OAuth アイテムが無効になったり、期限切れや取り消しになった場合は、新しい値を取得し、アプリケーションの Fuse Online 設定に貼り付けます。コネクションが検証されなかったアプリケーションの登録については、Fuse Online のアプリケーションおよびサービスへの接続 の手順を参照してください。更新された設定が導入されたら、上記の手順にしたがって更新されたコネクションを検証します。検証に成功し、このコネクションを使用しているインテグレーションが実行されている場合は、インテグレーションを再起動します。インテグレーションを再起動するには、停止してから開始します。

検証に失敗し、再接続が失敗したものの、サービスプロバイダーですべてが有効であると見られる場合は、アプリケーションで Fuse Online 環境を登録してから、コネクションを再作成します。Fuse Online は再作成時にコネクションを検証します。コネクションを再作成し、コネクションを使用しているインテグレーションがある場合、インテグレーションを編集して古いコネクションを削除し、新しいコネクションを追加する必要があります。インテグレーションが実行されている場合、これを停止してから再起動する必要があります。

## 3.4. インテグレーションへのコネクションの追加

コネクションをシンプルなインテグレーションやオペレーションフローに追加する場合、Fuse Online はコネクションのアプリケーション接続時に実行できるアクションのリストを表示します。必ず 1 つのアクションを選択する必要があります。実行中のインテグレーションでは、各コネクションは選択したアクションのみを実行します。たとえば、Twitter コネクションをインテグレーションの最初のコネクションとして追加した場合、Twitter で Twitter ハンドルがメンションされたツイートを監視する **Mention** アクションを選択することがあります。

一部のアクションを選択すると、アクションを設定する 1 つまたは複数のパラメーターの指定を要求されます。たとえば、Salesforce コネクションをインテグレーションに追加し、**On create** アクションを選択する場合、リードや取引先責任者など対象となるオブジェクトの種類を示す必要があります。

## 3.5. コネクション情報の表示および編集方法

コネクションの作成後、Fuse Online は内部識別子をコネクションに割り当てます。この識別子は変更しません。コネクションの名前、説明、または設定値を変更でき、Fuse Online は同じコネクションとして認識します。

コネクションに関する情報を表示および編集する方法は 2 つあります。

- 左側のパネルで **Connections** をクリックし、詳細を表示するコネクションをクリックします。

- 左側のパネルで **Integrations** をクリックし、インテグレーションを表示してその Summary ページを確認します。インテグレーションのフローダーイアグラムでは以下を行います。
  - シンプルなインテグレーションの場合は、コネクションアイコンをクリックしてそのコネクションの詳細を表示します。
  - API プロバイダーインテグレーションの場合は  をクリックし、インテグレーションのオペレーションリストを表示します。詳細を表示するコネクションが含まれるフローがあるオペレーションをクリックします。

**Connection Details** ページで、編集するコネクションのフィールドの横にある  をクリックして、そのフィールドを編集します。コネクションによっては、設定フィールドの下にある **Edit** をクリックして設定値を変更します。値を変更したら必ず **Save** をクリックしてください。

実行中のインテグレーションで使用されるコネクションを更新する場合、インテグレーションを再パブリッシュする必要があります。

アクセスを許可する OAuth プロトコルを使用するアプリケーションへのコネクションでは、コネクションが使用するログインクレデンシャルを変更することはできません。アプリケーションに接続し、異なるログインクレデンシャルを使用するには、新しいコネクションを作成する必要があります。

## 3.6. カスタムコネクターからコネクションを作成

カスタムコネクターを定義するエクステンションをアップロードした後、カスタムコネクターを使用できます。Fuse Online が提供するコネクターを使用してコネクションを作成する方法と同様に、カスタムコネクターを使用してコネクションを作成します。

カスタムコネクターは、OAuth プロトコルを使用するアプリケーションのカスタムコネクターであることがあります。このようなコネクターからコネクションを作成する前に、コネクターのアプリケーションにアクセスするための Fuse Online 環境を登録する必要があります。これは、コネクターのアプリケーションのインターフェイスで行います。Fuse Online 環境を登録する方法の詳細は、アプリケーションによって異なります。

たとえば、カスタムのコネクターを使用して Yammer へのコネクションを作成する場合に、Yammer 内に新規アプリケーションを作成して、Fuse Online 環境を登録する必要があります。登録によって、Fuse Online の Yammer クライアント ID と Fuse Online の Yammer クライアントシークレットの値が提供されます。Fuse Online 環境から Yammer へのコネクションは、この 2 つの値を提供する必要があります。

アプリケーションによっては、この 2 つの値にコンシューマー ID やコンシューマーシークレットなどの異なる名前を使用する場合があるため注意してください。

Fuse Online 環境を登録した後に、アプリケーションへのコネクションを作成できます。コネクションを設定するとき、クライアント ID およびクライアントシークレットを入力するためのパラメーターが必要になります。これらのパラメーターを利用できない場合は、エクステンションの開発者に連絡し、クライアント ID とクライアントシークレットの指定を可能にする更新済みのエクステンションを依頼する必要があります。

# 第4章 インテグレーションの作成

計画と準備が完了したら、インテグレーションを作成します。Fuse Online の Web インターフェイスでは、**Create Integration**をクリックすると、手順にしたがってインテグレーションを作成できます。

**前提条件**

- インテグレーションの計画に関する注意点
- 作成するインテグレーションの種類に応じて、以下を行います。
  - シンプルなインテグレーションを作成するための一般的なワークフロー を理解します。
  - API プロバイダーインテグレーションを作成するための一般的なワークフロー を理解します。

インテグレーションを作成するための情報および手順は以下を参照してください。

- 「インテグレーション作成の準備」
- 「インテグレーション実行トリガーの代替」
- 「シンプルなインテグレーションを作成する一般的な手順」
- 「インテグレーションの実行をトリガーするためタイマーコネクションを追加」
- 「データがコレクションにある場合のインテグレーションの動作」
- 「コネクション間のステップの追加」
- 「実行フローを決定するためのインテグレーションデータの評価」
- 「データマッパーステップの追加」
- 「基本のフィルターステップの追加」
- 「高度なフィルターステップの追加」
- 「テンプレートステップの追加」
- 「カスタムステップの追加」

## 4.1. インテグレーション作成の準備

インテグレーション作成の準備を開始するには、インテグレーションの計画に関する注意点 に記載されている質問への回答を参照します。インテグレーションを計画したら、インテグレーションの作成前に以下を行う必要があります。

1. 接続するアプリケーションが OAuth プロトコルを使用するかどうかを判別します。OAuth を使用する各アプリケーションでは、Fuse Online をそのアプリケーションへのアクセスが許可されるクライアントとして登録します。OAuth プロトコルを使用するアプリケーションには以下が含まれます。
   - Dropbox
   - Google アプリケーション (Gmail、カレンダー、およびスプレッドシート)

- Salesforce
- SAP Concur
- Twitter

2. 接続するアプリケーションが HTTP Basic 認証を使用するかどうかを判別します。HTTP Basic 認証を使用する各アプリケーションにアクセスするためのユーザー名およびパスワードを特定します。コネクションを作成する際に、この情報を提供する必要があります。
3. 統合するアプリケーションごとにコネクションを作成します。

**その他のリソース**

- 承認取得のための一般的な手順
- Fuse Online からアプリケーションへのコネクションの作成

## 4.2. インテグレーション実行トリガーの代替

インテグレーションの作成時、インテグレーションの実行をトリガーする方法はインテグレーションの最初のステップによって決定されます。インテグレーションの最初のステップは以下のいずれかになります。

- **アプリケーションまたはサービスへのコネクション**。特定のアプリケーションやサービスのコネクションを設定します。以下に例を示します。
  - Twitter へのコネクションはツイートを監視でき、ツイートに指定のテキストが含まれている場合にシンプルなインテグレーションの実行をトリガーできます。
  - Salesforce へのコネクションは、新しいリードが作成されたときにシンプルなインテグレーションの実行をトリガーできます。
  - AWS S3 へのコネクションは、特定のバケットを定期的にポーリングでき、バケットにファイルが含まれる場合にシンプルなインテグレーションの実行をトリガーできます。
- **Timer**.Fuse Online は、シンプルなインテグレーションの実行を指定した間隔でトリガーします。これは、簡単なタイマーまたは **cron** ジョブになります。
- **Webhook**.クライアントは、HTTP **GET** または **POST** リクエストを Fuse Online が公開する HTTP エンドポイントに送信できます。リクエストはシンプルなインテグレーションの実行をトリガーします。
- **API プロバイダー**。API プロバイダーインテグレーションは、REST API サービスから開始します。この REST API サービスは、API プロバイダーインテグレーションの作成時に提供する OpenAPI 3 (または 2) ドキュメントによって定義されます。API プロバイダーインテグレーションをパブリッシュした後、Fuse Online は REST API サービスを OpenShift にデプロイします。インテグレーションエンドポイントにネットワークアクセスできるクライアントは、インテグレーションの実行をトリガーできます。

## 4.3. シンプルなインテグレーションを作成する一般的な手順

Fuse Online では手順にしたがってシンプルなインテグレーションを作成できます。最初のコネクション、最後のコネクション、途中のコネクション (任意)、およびその他のステップを選択するよう促されます。インテグレーションが完了したら、パブリッシュして稼働するか、保存して後でパブリッシュすることができます。

API プロバイダーインテグレーションを作成する手順の詳細は、「API プロバイダーインテグレーションの作成」 を参照してください。

**前提条件**

- インテグレーションのステップが計画済みである必要があります。
- このインテグレーションで接続する各アプリケーションまたはサービスへのコネクションが作成済みである必要があります。

**手順**

1. Fuse Online の左パネルで **Integrations**をクリックします。
2. **Create Integration** をクリックします。
3. 最初のコネクションを追加および設定します。
    a. **Choose a connection**ページで、インテグレーションを開始するために使用するコネクションをクリックします。このインテグレーションの稼働時に、Fuse Online はこのアプリケーションに接続し、インテグレーションが操作するデータを取得します。
    b. **Choose an action**ページで、このコネクションが実行するアクションを選択します。使用できるアクションはコネクションごとに異なります。
    c. アクションを設定するページで、フィールドに値を入力します。
    d. 任意で、コネクションにデータタイプの指定が必要な場合は、**Next** をクリックしてアクションの出入力タイプを指定するよう要求されます。
    e. **Next** をクリックして、最初のコネクションを追加します。

    アプリケーションに接続する代わりに、指定した間隔でインテグレーションの実行をトリガーするタイマーや、HTTP リクエストを許可する Webhook を最初のコネクションとすることが可能です。

    最初のコネクションの選択および設定後、Fuse Online は最後のコネクションを選択するよう要求します。
4. 最後のコネクションを選択および設定します。
    a. **Choose a connection**ページで、インテグレーションを完了するために使用するコネクションをクリックします。このインテグレーションの稼働中に、Fuse Online はインテグレーションが操作するデータでこのアプリケーションに接続します。
    b. **Choose an action**ページで、このコネクションが実行するアクションを選択します。使用できるアクションはコネクションごとに異なります。
    c. アクションを設定するページで、フィールドに値を入力します。
    d. 任意で、コネクションにデータタイプの指定が必要な場合は、**Next** をクリックしてアクションの出入力タイプを指定するよう要求されます。
    e. **Next** をクリックして、最後のコネクションを追加します。

アプリケーションに接続する代わりに、最後のコネクションはインテグレーションが処理したメッセージに関する情報をインテグレーションのログに送信できます。これには、Fuse Online が最後のコネクションの選択を要求したときに **Log** を選択します。

5. 必要に応じて、最初のコネクションと最後のコネクションとの間に 1 つ以上のコネクションを追加します。コネクションごとにアクションを選択し、必要な設定詳細を入力します。

6. 必要に応じて、コネクション間のインテグレーションデータで操作する 1 つ以上のステップを追加します。コネクション間のステップの追加 を参照してください。

7. インテグレーションビジュアライゼーションで、  アイコンを見つけます。この警告は、このコネクションの前にデータマッパーステップが必要なことを示しています。必要なデータマッパーステップを追加します。

8. 必要なデータマッパーステップを追加すると、次の理由により、  アイコンが引き続き表示されることがあります (編集プロセス中にいつでも表示される可能性があります)。

   - ソースステップの 1 つが出力を変更しました
   - ターゲットステップの入力がマッパーの出力と互換性がありません
   - ソースステップの 1 つがありません
   - 対象のステップがありません
     このシナリオでは、この警告は、ステップ 7 で追加されたデータマッパーステップを編集する必要があることを示しています。

9. インテグレーションに必要なステップがすべて含まれている場合は、インテグレーションの実行を開始するかどうかに応じて **Save** または **Publish** をクリックします。

10. **Name** フィールドに、このインテグレーションを別のインテグレーションと区別する名前を入力します。

11. 必要に応じて、**Description** フィールドに説明を入力します。たとえば、このインテグレーションが実行する内容の説明を入力できます。

12. 必要に応じて、**ラベル** フィールドで、1 つ以上のラベルを統合に追加します。ラベルは、後で OpenShift でフィルタリングおよび選択するために、統合 (または他の OpenShift リソース) に適用できるキーと値のペアのタグです。たとえば、OpenShift 管理者ユーザーは、実行中の Pod またはデプロイメントのリストをラベルでフィルタリングできます。

13. 必要に応じて、統合用に 1 つ以上のカスタム環境変数を追加します。これらの環境変数を使用して、**SERVER_MAX_HTTP_HEADER_SIZE** などの Spring Boot 設定オプションを設定できます。インテグレーションを保存する際に、Fuse Online にこれらの環境変数を設定しても、たとえば、OpenShift Web コンソールインターフェイスなどを介して手動で設定された他の環境設定が変更されたり、影響を受けたりしないことに注意してください。

14. 必要に応じて、インポートしたライブラリーエクステンションの一覧から、インテグレーションに関連付けるライブラリーエクステンションを 1 つ以上選択できます。リストで表示して選択できるように、ライブラリー **.jar** ファイルを Fuse Online エクステンションとしてインポートする必要があります。
    ライブラリーエクステンションの詳細は、ライブラリーエクステンションの開発方法 を参照してください。

15. インテグレーションの実行を開始する準備ができたら、**Save and publish** をクリックします。

Fuse Online にはインテグレーションの概要が表示されます。Fuse Online がパブリッシュの処理中であることが分かります。インテグレーションの状態が **Running**になるまでに多少時間がかかる可能性があります。

インテグレーションをパブリッシュしない場合は、**Save** クリックします。Fuse Online はインテグレーションを保存し、フロービジュアライゼーションを表示します。編集を続行できます。または、ページの上部のパンくずリストで **Integrations** をクリックし、インテグレーションの一覧を表示します。保存してもインテグレーションがパブリッシュされない場合は、**Stopped** がインテグレーションのエントリーに表示されます。

## 4.4. インテグレーションの実行をトリガーするためタイマーコネクションを追加

指定のスケジュールに応じてシンプルなインテグレーションの実行をトリガーするには、タイマーコネクションをシンプルなインテグレーションの最初のコネクションとして追加します。タイマーコネクションをフローの途中やフローの最後に追加することはできません。

**手順**

1. Fuse Online で左側にある **Integrations** をクリックします。
2. **Create Integration** をクリックします。
3. **Choose a connection** ページで **Timer** をクリックします。
   Fuse Online は **Timer** コネクションを提供するため、タイマーコネクションを作成する必要はありません。
4. **Choose an action** ページで **Cron** または **Simple** を選択します。
   - **cron** タイマーには、インテグレーションの実行をトリガーするスケジュールを指定する **cron** 式が必要です。
   - 期間とその時間単位の指定を要求されます (例:**5 seconds**、**1 hour**)。使用可能な単位はミリ秒 (milliseconds)、秒 (seconds)、分 (minutes)、時間 (hours)、および日 (days) です。
5. 追加するタイマーのタイプに応じて、**cron** 式または選択した時間単位の期間を入力します。
6. **Next** をクリックして **Timer** コネクションをインテグレーションの最初のコネクションとして追加します。

## 4.5. データがコレクションにある場合のインテグレーションの動作

コネクションは、すべて同じタイプの複数の値が含まれるコレクションを返すことがあります。コネクションがコレクションを返すと、フローは以下を含む複数の方法でコレクションで操作できます。

- 各ステップを 1 度、コレクションに実行します。
- 各ステップを 1 度、コレクションの各要素に実行します。
- 一部のステップを 1 度、コレクションに実行し、他のステップを 1 度、コレクションの各要素に実行します。

フローのコレクションでの操作方法を決定するには、フローが接続するアプリケーション、それらのアプリケーションがコレクションに対応できるかどうか、およびフローが達成することを知っている必要があります。その後、以下の情報を使用して、コレクションを処理するフローにステップを追加できま

す。

- 「データ型とコレクションについて」
- 「コレクションの処理」
- 「データマッパーを使用したコレクションの処理」
- 「分割ステップの追加」
- 「集約ステップの追加」
- 「フローでコレクションを処理する例」

### 4.5.1. データ型とコレクションについて

データマッパーはソースフィールドとターゲットフィールドを表示し、必要なフィールド間のマッピングを定義します。

データマッパーでは、フィールドは以下のいずれかになります。

- 単一の値を格納する **プリミティブ** タイプ。プリミティブタイプの例には、**boolean**、**char**、**byte**、**short**、**int**、**long**、**float**、および **double** があります。プリミティブタイプは単一のフィールドであるため拡張できません。
- 異なるタイプの複数のフィールドで設定される **コンプレックス** タイプ。コンプレックスタイプの子フィールドを設計時に定義します。データマッパーではコンプレックスタイプを拡張して子フィールドを表示できます。

各タイプのフィールド (プリミティブおよびコンプレックス) をコレクションとすることもできます。コレクションは、複数の値を持つことができる単一のフィールドです。コレクションのアイテム数はランタイム時に決定されます。設計時、データマッパーではコレクションは  で示されます。コレクションがデータマッパーインターフェイスで拡張可能かどうかは、タイプにより決定されます。コレクションがプリミティブタイプの場合、拡張できません。コレクションがコンプレックスタイプの場合、データマッパーを展開し、コレクションの子フィールドを表示できます。各フィールドをマップ元またはマップ先とすることができます。

以下に例を示します。

- **id** はプリミティブタイプのフィールド (**int**) です。実行時に、社員は **ID** を 1 つだけ持てます。たとえば、**ID=823** のみを持てます。そのため、**ID** はコレクションではないプリミティブタイプです。データマッパーでは、**ID** は拡張できません。
- **email** はプリミティブタイプのフィールド (文字列) です。実行時に、社員は複数の **email** の値を持つことができます。たとえば、**email<0>=aslan@home.com** および **email<1>=aslan@business.com** を持つことができます。そのため、**email** はコレクションでもあるプリミティブタイプです。データマッパーは、  を使用して、**email** フィールドがコレクションであることを示しますが、**email** はプリミティブタイプであるため (子フィールドがない)、拡張できません。
- **employee** は、**ID** や **email** を含む複数の子フィールドを持つ複雑なオブジェクトフィールドです。会社には多くの従業員がいるため、起動時に **employee** はコレクションでもあります。設計時に、データマッパーは  を使用して **employee** がコレクションであることを示します。**employee** フィールドは、子フィールドが含まれるコンプレックスタイプであるため、拡張可能です。

### 4.5.2. コレクションの処理

フローがコレクションを処理する最も簡単な方法は、データマッパーを使用してソースコレクションにあるフィールドをターゲットコレクションにあるフィールドにマップすることです。多くのフローでは、これだけが必要になります。たとえば、フローはデータベースから社員のレコードのコレクションを取得し、それらのレコードをスプレッドシートに挿入します。データマッパーステップは、データベースコネクションと Google スプレッドシートコネクションの間でデータベースフィールドを Google スプレッドシートフィールドにマップします。ソースとターゲットはコレクションであるため、Fuse Online がフローを実行すると、Google スプレッドシートコネクションを 1 度呼び出します。この呼び出しで、Fuse Online はレコードを繰り返し処理し、スプレッドシートが適切に入力されます。

フローによっては、コレクションを個別のオブジェクトに分割する必要がある場合があります。たとえば、データベースに接続し、特定の日付までに割り当てられた休暇を取らないと休暇が失効してしまう社員のコレクションを取得する場合など考えられます。その後、フローはこれらの各社員にメール通知を送信する必要があります。このフローでは、データベースコネクションの後に分割ステップ (split step) を追加します。その後、社員のレコードのソースフィールドを、メッセージを送信する Gmail コネクションのターゲットフィールドにマップする、データマッパーステップを追加します。Fuse Online がフローを実行すると、データマッパーステップと Gmail コネクションを社員ごとに 1 度実行します。

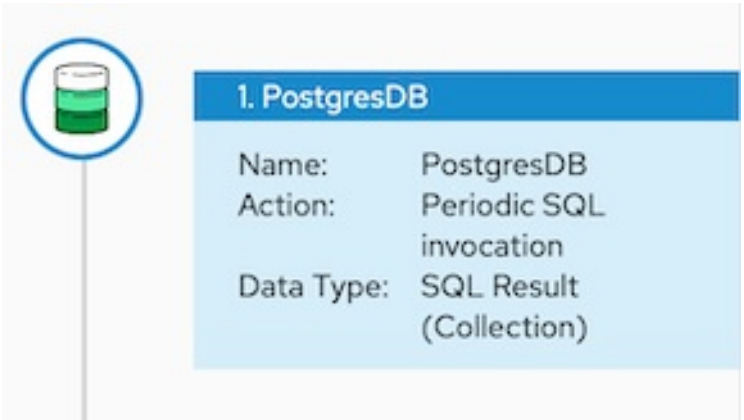
場合によっては、フローのコレクションを分割し、フローがコレクションの各要素に一部のステップを 1 度実行した後、フローをコレクションで再度操作したいことがあります。前述の例について考えてみましょう。Gmail コネクションでメッセージを各従業員に送信した後に、通知済みの従業員一覧をスプレッドシートに追加すると仮定します。このシナリオでは、Gmail コネクションの後に、集約の手順を追加して、従業員名のコレクションを作成します。次に、ソースコレクションのフィールドをターゲット Google スプレッドシートコネクションのフィールドにマップするデータマッパーステップを追加します。Fuse Online がフローを実行すると、新しいデータマッパーステップと Google スプレッドシートコネクションをコレクションに 1 度実行します。

これが、フローのコレクションを処理する最も一般的なシナリオになります。ただし、より複雑な処理も可能です。たとえば、コレクションの要素自体がコレクションである場合、分割および集約ステップを他の分割および集約ステップ内で入れ子にすることができます。

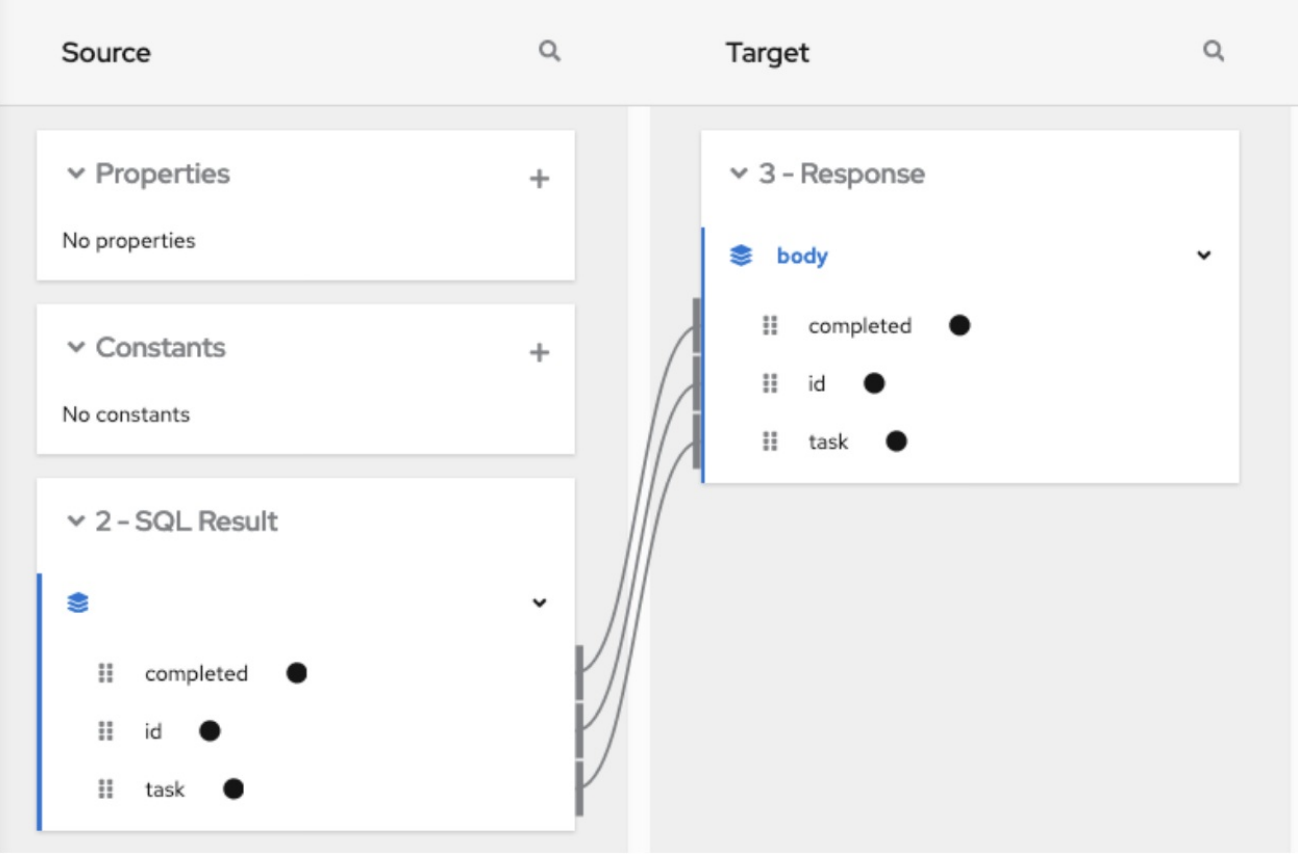
### 4.5.3. データマッパーを使用したコレクションの処理

フローでは、ステップがコレクションを出力し、フローの後続のコネクションはコレクションを入力として想定する場合、データマッパーを使用してフローがどのようにコレクションを処理するかを指定できます。

ステップがコレクションを出力すると、フロービジュアライゼーションはステップの詳細で **Collection** を表示します。以下に例を示します。



データマッパーステップを、コレクションを提供するステップの後およびマッピングを必要とするステップの前に追加します。フローでこのデータマッパーステップが必要な場所は、フローの他のステップによって異なります。以下のイメージは、ソースコレクションフィールドからターゲットコレクションフィールドへのマッピングを示しています。




ソースおよびターゲットパネルで、データマッパーは を表示し、コレクションを示します。

コレクションがコンプレックスタイプの場合、データマッパにコレクションの子フィールドが表示されます。各フィールドをマップ元またはマップ先とすることができます。

ソースフィールドが複数のコレクションで入れ子になっている場合、以下の条件の 1 つを満たすターゲットフィールドにマップできます。

- ターゲットフィールドは、ソースフィールドと同じ数のコレクションで入れ子になっています。たとえば、以下のマッピングが許可されます。
  - /A<>/B<>/C → /D<>/E<>/F
  - /A<>/B<>/C → /G<>/H/I<>/J
- ターゲットフィールドは 1 つのコレクションでのみ入れ子になっています。たとえば、以下のマッピングが許可されます。
  /A<>/B<>/C → /K<>/L

  この場合、データマッパーで深さ優先アルゴリズムが使用され、ソースのすべての値が反復処理されます。データマッパーによって、ソース値は発生順に単一のターゲットコレクションに配置されます。

以下のマッピングは許可されません。

/A<>/B<>/C cannot-map-to /M<>/N/O<>/P<>/Q

Fuse Online でフローが実行されると、ソースコレクション要素が繰り返し処理され、ターゲットコレクション要素が入力されます。1つ以上のソースコレクションフィールドをターゲットコレクションまたはターゲットコレクションフィールドにマップする場合、ターゲットコレクション要素にはマップされたフィールドのみの値が含まれます。

ソースコレクションまたはソースコレクションのフィールドをコレクションではないターゲットフィールドにマップする場合、Fuse Online がフローを実行するときにソースコレクションの最後の要素のみから値を割り当てます。コレクションの他の要素は、そのマッピングステップで無視されます。しかし、後続のマッピングステップはソースコレクションのすべての要素にアクセスできます。

コネクションが JSON または Java ドキュメントに定義されたコレクションを返すと、データマッパーは通常コレクションとしてソースドキュメントを処理できます。

### 4.5.4. 分割ステップの追加

フローの実行中に、コネクションがオブジェクトのコレクションを返すと、Fuse Online はコレクションに後続のステップを1度実行します。コレクションにある各オブジェクトに後続のステップを1度実行する場合は、分割ステップを追加します。たとえば、Google スプレッドシートコネクションは行オブジェクトのコレクションを返します。行ごとに後続のステップを1度実行するには、Google スプレッドシートコネクションの後に分割ステップを追加します。

分割ステップへの入力が常にコレクションであるようにしてください。分割ステップが、コレクションタイプではないソースドキュメントを取得する場合、ステップは空白文字で入力を分割します。たとえば、Fuse Online は Hello world! の入力を Hello と world! という 2 つの要素に分割し、これらの要素をフローの次のステップに渡します。特に XML データはコレクションタイプではありません。

#### 前提条件

- フローを作成または編集することになります。
- フローに必要なコネクションがすべて存在する必要があります。
- フロービジュアライゼーションでは、ソースデータを取得するコネクションはデータが **(Collection)** であると示します。

#### 手順

1. フロービジュアライゼーションの分割ステップを追加する場所で  をクリックします。
2. **Split** をクリックします。このステップに設定は必要ありません。
3. **Next** をクリックします。

#### 関連情報

通常、データマッパーステップを追加する前に、分割ステップと集約ステップを追加します。これは、データがコレクションまたは個々のオブジェクトであるかがマッピングに影響するためです。データマッパーステップを追加して分割ステップを追加する場合、通常はマッピングをやり直す必要があります。同様に、分割または集約ステップを削除する場合もマッピングをやり直す必要があります。

### 4.5.5. 集約ステップの追加

フローに、Fuse Online が個別のオブジェクトからコレクションを作成する、集約ステップを追加します。実行中、Fuse Online は集約ステップの後に各オブジェクトに対して後続のステップを1度実行せずに、コレクションに対して後続のステップを1度実行します。

集約ステップをフローに追加するかどうかを決定する場合は、フローのコネクションを考慮してください。分割ステップの後、Fuse Online は後続の各コネクションに対して、フローのデータの各要素のために 1 度アプリケーションに接続します。コネクションによっては、複数回接続するよりも 1 度接続した方が望ましいことがあります。

**前提条件**

- フローを作成または編集することになります。
- フローに必要なコネクションがすべて存在する必要があります。
- 前の手順でコレクションを個別のオブジェクトに分割している必要があります。

**手順**

1. フロービジュアライゼーションの、集約ステップをフローに追加する場所で  をクリックします。
2. **Aggregate** をクリックします。このステップに設定は必要ありません。
3. **Next** をクリックします。

**関連情報**

通常、データマッパーステップを追加する前に、分割および集約ステップを追加します。これは、データがコレクションまたは個々のオブジェクトであるかがマッピングに影響するためです。データマッパーステップを追加して集約ステップを追加する場合、通常はマッピングをやり直す必要があります。同様に、集約ステップを削除する場合もマッピングをやり直す必要があります。

### 4.5.6. フローでコレクションを処理する例

このシンプルなインテグレーションは、Fuse Online によって提供されるサンプルデータベースからタスクのコレクションを取得します。フローはコレクションを個別のタスクオブジェクトに分割し、これらのオブジェクトをフィルターして実行されたタスクを見つけます。その後、フローは完了したタスクをコレクションで集約し、そのコレクションのフィールドをスプレッドシートのフィールドにマップします。完了したタスクのリストをスプレッドシートに追加して終了します。

以下の手順は、このシンプルなインテグレーションを作成する方法を説明します。

**前提条件**

- Google スプレッドシートコネクションが作成済みである必要があります。
- Google スプレッドシートコネクションがアクセスするアカウントに、データベースレコードを受信するスプレッドシートがある必要があります。

**手順**

1. **Create Integration** をクリックします。
2. 最初のコネクションを追加します。
   a. **Choose a connection** ページで **PostgresDB** をクリックします。
   b. **Choose an action** ページで **Periodic SQL Invocation** を選択します。

c. **SQL Statement** フィールドに **select * from todo** を入力し、**Next** をクリックします。

このコネクションは、タスクオブジェクトのコレクションを返します。

3. 最後のコネクションを追加します。

a. **Choose a connection** ページで、Google スプレッドシートコネクションをクリックします。

b. **Choose an action** ページで **Append values to a sheet** を選択します。

c. **SpreadsheetId** フィールドにスプレッドシートの ID を入力し、タスクの一覧を追加します。

d. **Range** フィールドに **A:B** を値を追加するターゲット列として入力します。最初のコラムである **A** はタスク ID のコラムです。次のコラムである **B** は、タスク名のコラムです。

e. **Major Dimension** と **Value Input Option** のデフォルト値を受け入れ、**Next** をクリックします。

Google スプレッドシートコネクションは、コレクションの各要素をスプレッドシートに追加してフローを終了します。

4. フローに分割ステップを追加します。

a. フロービジュアライゼーションで、プラス記号をクリックします。

b. **Split** をクリックします。

フローが分割ステップを実行した後、結果は個別のタスクオブジェクトのセットになります。Fuse Online は、各タスクオブジェクトに対してフローの後続ステップを 1 度実行します。

5. フィルターステップをフローに追加します。

a. フロービジュアライゼーションにて、分割ステップの後でプラスマークをクリックします。

b. **Basic Filter** をクリックし、以下のようにフィルターを設定します。

i. 最初のフィールドをクリックし、評価するデータが含まれるフィールドの名前である **completed** を選択します。

ii. 2 つ目のフィールドに、**completed** フィールドの値が満たさなければならない条件として **equals** を選択します。

iii. 3 番目のフィールドに、**completed** フィールドになければならない値として **1** を指定します。**1** は、タスクが完了したことを示します。

c. **Next** をクリックします。

実行中、フローは各タスクオブジェクトに対してフィルターステップを 1 度実行します。結果は、個別の完了したタスクオブジェクトのセットになります。

6. 集約ステップをフローに追加します。

a. フロービジュアライゼーションにて、フィルターステップの後でプラス記号をクリックします。

b. **Aggregate** をクリックします。

結果セットには、完了したタスクごとに要素が含まれるコレクションが含まれるようになりました。

7. データマッパーステップをフローに追加します。

a. フロービジュアライゼーションにて、集約ステップの後でプラスマークをクリックします。

b. **Data Mapper** をクリックし、以下のフィールドを SQL 結果ソースのコレクションから Google スプレッドシートのターゲットコレクションにマップします。

- **id** から **A**
- **task** から **B**

c. **Done** をクリックします。

8. **Publish** をクリックします。

**結果**

インテグレーションの実行時に、毎分サンプルデータベースからタスクを取得し、完了したタスクをスプレッドシートの最初のシートに追加します。インテグレーションは、タスク ID を最初の列である **A** にマップし、タスク名を 2 番目の列である **B** にマップします。

## 4.6. コネクション間のステップの追加

必須ではありませんが、必要なコネクションをすべてプライマリーフローに追加してから、フローが実行するプロセスにしたがって、コネクションの間に追加ステップを追加することが推奨されます。フローでは、各ステップは以前のコネクションおよび以前のステップから取得したデータで操作します。結果となるデータは、フローの次のステップで利用できます。

多くの場合、コネクションから受け取ったデータフィールドを、フローの次のコネクションが操作できるデータフィールドにマップする必要があります。すべてのコネクションをフローに追加したら、フロービジュアライゼーションを確認します。入力データで操作する前にデータマッピングを必要とする各コネクションに対し、Fuse Online は  を表示します。このアイコンをクリックして、**Data Type Mismatch: Add a data mapper step before this connection to resolve the difference.** を表示します。

メッセージのリンクをクリックして、データマッパーステップを追加および指定する **Configure Mapper** ページを表示します。しかし、必要な他のステップを追加してから、データマッパーステップを最後に追加することが推奨されます。

## 4.7. 実行フローを決定するためのインテグレーションデータの評価

フローでは、**Conditional Flows** ステップでインテグレーションデータが指定の条件に対して評価されます。指定した条件ごとに、コネクションおよびその他のステップをその条件に関連するフローに追加します。実行中、**Conditional Flows** ステップによって受信データが評価され、実行するフローが決定されます。

詳細は以下のセクションを参照してください。

- 「Conditional Flows ステップの動作」

- 「Conditional Flows ステップの例」
- 「Conditional Flows ステップの一般的な設定手順」
- 「基本の式ビルダーを使用した条件の指定」
- 「上級の式ビルダーを使用した条件の指定」
- 「条件付きフローへのステップの追加」

### 4.7.1. Conditional Flows ステップの動作

統合開発中、**Conditional Flows** (条件付きフロー) ステップをフローに追加して 1 つ以上の条件を追加できます。条件ごとに、ステップをその条件のみに関連する条件付きフローに追加します。インテグレーションの実行中、前のインテグレーションステップによって **Conditional Flows** ステップに渡されるメッセージごとに、**Conditional Flows** ステップは条件を指定するために Fuse Online ページで定義する順序で、メッセージの内容を指定の条件に対して評価します。

**Conditional Flows** ステップでの動作は以下のいずれかになります。

- true に評価される最初の条件では、その条件に関連する条件付きフローがインテグレーションによって実行されます。
- 条件が true に評価されず、デフォルトの条件付きフローがある場合は、インテグレーションはそのフローを実行します。
- 条件が true に評価されず、デフォルトの条件付きフローがない場合、インテグレーションは条件付きフローを実行しません。

条件付きフローの実行後、または true に評価される条件とデフォルトの条件フローがない場合は、インテグレーションによってプライマリーフローの次のステップが実行されます。

### 4.7.2. Conditional Flows ステップの例

インテグレーションが SQL データベースに接続し、各社員の有給休暇 (PTO) に関する情報を取得するとします。返されたデータは以下を示します。

- 特定日までに PTO を消化しないと失効する可能性がある社員。
- 付与分を超える PTO を使用した社員。
- 時間の制限なく PTO を使える残りの社員。

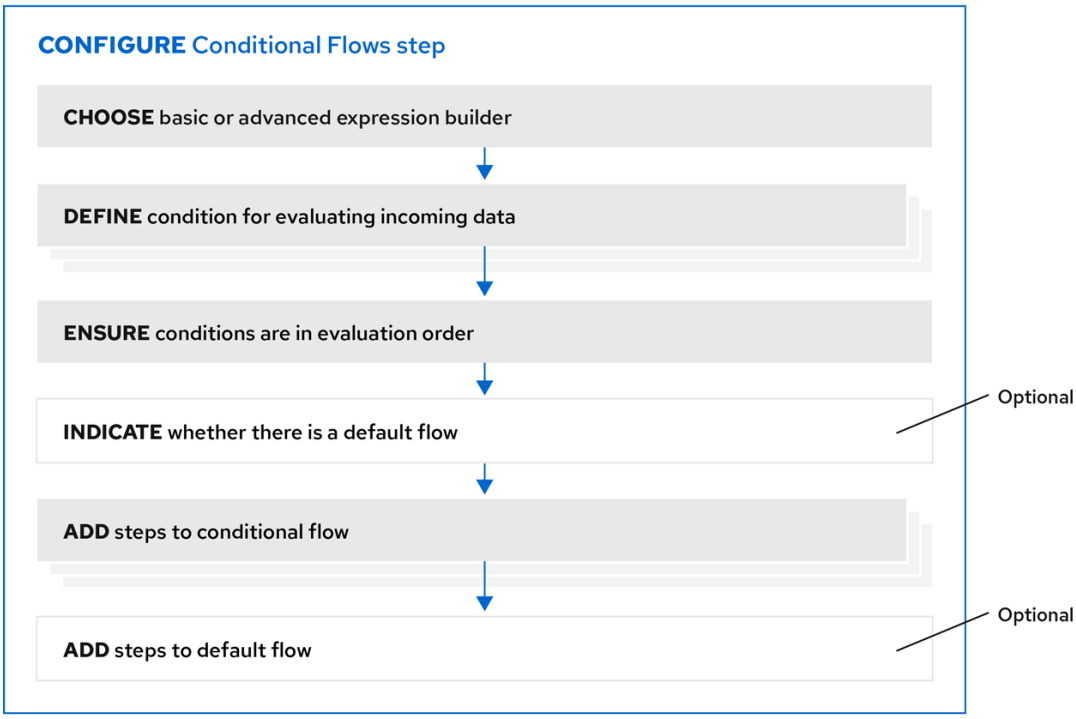
このインテグレーションの例の場合、**Conditional Flows** ステップでは、2 つの条件、各条件の 1 つの実行フロー、およびデフォルトの実行フローを定義できます。

- PTO が特定の数を超えた場合、特定日までに消化しないと失効する可能性がある PTO があることを意味します。この条件が true である場合、インテグレーションは該当する社員にメールを送信するフローを実行します。メールには、消化する必要がある PTO の時間数または日数と、失効日が含まれます。
- PTO が負の値の場合、付与分を超える PTO が使用されたこと意味します。この条件が true である場合、インテグレーションは該当する社員にメールを送信するフローを実行します。メールには、社員が取得した過剰分の PTO が含まれ、PTO の付与が再開される日付が指定されます。

- 2 つの条件が両方とも true でない場合、インテグレーションはデフォルトのフローを実行します。このインテグレーションの例は、PTO が負の値ではなく、指定の数を超えない社員にデフォルトの条件付きフローを実行します。このフローは、社員が使用できる PTO の時間数または日数の明細をメールで該当する社員に送信します。

### 4.7.3. Conditional Flows ステップの一般的な設定手順

Conditional Flows ステップをフローに追加した後に行うステップ設定のワークフローは次のようになります。




**ワークフローの詳細**

- 基本の式ビルダーによって、評価する内容が含まれるプロパティーや、テストする条件および値が要求されます。基本の式ビルダーは、ほとんどの Conditional Flows ステップに適しています。
- 上級の式ビルダーを使用すると、Camel Simple 言語で条件式を指定できます。
- すべての条件で同じ式ビルダーを使用する必要があります。つまり、基本の式ビルダーまたは上級の式ビルダーを使用して Conditional Flows ステップを設定する必要があります。両方を使用することができません。
- 条件付きフローでは、Conditional Flows ステップを追加できません。

### 4.7.4. 基本の式ビルダーを使用した条件の指定

受信データを評価してインテグレーションの実行パスを決定する場合に、フローで Conditional Flows ステップを追加します。次の手順では、基本の式ビルダーを使用して条件を指定する方法を説明します。

**前提条件**

- プライマリーフローを作成または編集することになります。これがシンプルなインテグレーションである場合、最初と最後のコネクションが追加されている必要があります。
- Conditional Flows ステップへの入力は、個別のメッセージである必要があります。インテグレーションビジュアリゼーションで、前のステップの Data Type が (Collection) である場合、前のステップとこの Conditional Flows ステップの間に Split ステップを追加します。
- インテグレーションは、追加する Conditional Flows ステップにメッセージを渡します。このメッセージに含まれるフィールドを熟知している必要があります。

**手順**

1. インテグレーションビジュアライゼーションの、Conditional Flows ステップを追加する場所で  をクリックします。
2. Conditional Flows をクリックします。
3. Basic expression builder エントリーで Select をクリックします。
4. Configure Conditional Flows ページで、以下の条件を 1 つまたは複数定義します。
   a. 最初の When フィールドをクリックします。
   b. プロパティーのリストで、Conditional Flows ステップによって評価される内容が含まれるプロパティーをクリックします。
   c. 次のフィールドで、ステップがデータを評価する条件として Contains を指定するか、別の条件を選択します。このフィールドに選択する条件は、次のフィールドに入力する値に対して true である必要があります。
   d. 3 つ目のフィールドに、条件がテストする値を指定します。
   e. 任意設定:Add another condition をクリックして、別の条件を指定します。
   f. 定義する追加条件ごとに、この手順を繰り返します。
   g. 任意設定:条件の右にある上矢印または下矢印をクリックして、インテグレーションで定義された条件を評価する順番を変更します。
   h. 任意設定:デフォルトの条件付きフローにする場合は Execute default flow をクリックします。
   実行中に Execute default flow を選択した場合、指定した条件に true となるものがなければ、インテグレーションはデフォルトの条件付きフローを実行します。実行中に Execute default flow を選択しなかった場合、指定した条件に true となるものがなければ、この Conditional Flows ステップに続くステップで、インテグレーションの実行が継続されます。
5. Next をクリックします。
6. 任意設定:Fuse Online で出力データタイプの指定を要求された場合は指定します。この Conditional Flows ステップの一部であるすべての条件付きフローは、同じ出力タイプである必要があります。
7. Next をクリックします。
   Fuse Online にフロービジュアライゼーションが表示されます。追加する Conditional Flows ステップの下に、指定した各条件のエントリーがあります。Conditional Flows ステップにデフォ

ルトフローがあることを示した場合は、**Otherwise** デフォルトフローのエントリーも表示されます。

### 次のステップ

条件ごとに、条件に関連するフローにステップを追加します。デフォルトフローがある場合は、そのフローにステップを追加します。

### その他のリソース

- 各条件の中間フィールドに選択できる条件に関する詳細は Camel Simple Language Operator Support を参照してください。**matches** 条件は、Simple 言語の **regex** 演算子に対応することに注意してください。
- 基本の式ビルダーを使用して、必要な条件を定義できない場合は、上級の式ビルダーを使用した条件の指定 を参照してください。

## 4.7.5. 上級の式ビルダーを使用した条件の指定

受信データを評価してインテグレーションの実行パスを決定する場合に、フローで **Conditional Flows** ステップを追加します。次の手順では、上級の式ビルダーを使用して Camel Simple 言語で条件式を指定する方法を説明します。

### 前提条件

- プライマリーフローを作成または編集することになります。これがシンプルなインテグレーションである場合、最初と最後のコネクションが追加されている必要があります。
- **Conditional Flows** ステップへの入力は、個別のメッセージである必要があります。インテグレーションビジュアリゼーションで、前のステップの **Data Type** が **(Collection)** である場合、**分割** ステップを追加します。
- インテグレーションは、追加する **Conditional Flows** ステップにメッセージを渡します。このメッセージに含まれるフィールドを熟知している必要があります。
- Camel Simple Expression 言語を熟知しているか、評価する条件の式がある必要があります。

### 手順

1. インテグレーションビジュアライゼーションの、**Conditional Flows** ステップを追加する場所で  をクリックします。
2. **Conditional Flows** をクリックします。
3. **Advanced expression builder** エントリーで **Select** をクリックします。
4. **Configure Conditional Flows** ページで、以下の条件を 1 つまたは複数定義します。
   a. 最初の **When** フィールドに Camel Simple 言語の条件式を入力します。式の左側は、${...} で囲まれた変数式でなければなりません。
   有効な式の例を次に示します。

```
${header.type} == 'note'
```

```
${body.title} contains 'Important'
```

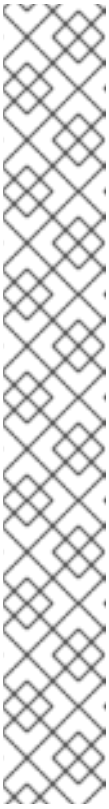
以下は、無効な式の例です。

```
'note' == ${header.type}
```

以下は、メッセージの本文に **160** より大きい **pto** フィールドが含まれている場合に true と評価される式を記述する方法を示す例です。

```
${body.pto} > 160
```

この式が true に評価されると、インテグレーションはこの条件で作成および関連付けする条件付きフローを実行します。



**注記**

**Conditional Flows** ステップが以下のようなフローの 1 つにある場合は、式に追加のプロパティーを指定する必要があります。

- API プロバイダーインテグレーションのオペレーションフロー。
- Webhook コネクションで始まるシンプルなインテグレーション。
- カスタム REST API コネクションで始まるシンプルなインテグレーション。

これらのフローでは、Fuse Online は **body** プロパティー内で実際のメッセージコンテンツをラッピングします。これは、**Conditional Flows** ステップへの入力に **body** プロパティーが含まれ、このプロパティーに実際のメッセージコンテンツが含まれる別の **body** プロパティーが含まれることを意味します。そのため、このようなフローの 1 つにある **Conditional Flows** ステップの式に、**body** のインスタンスを 2 つを指定する必要があります。たとえば、入力メッセージの **pto** フィールドにあるコンテンツを評価するとします。この場合、以下のように式を指定します。

```
${body.body.pto} > 160
```

b. 任意設定:**Add another condition** をクリックし、前のステップを繰り返します。定義する追加条件ごとに、この作業を行います。

c. 任意設定:条件付きフィールドの右にある上矢印または下矢印をクリックして、定義された条件が **Conditional Flows** ステップによって評価される順番を変更します。

d. 任意設定:デフォルトの条件付きフローにする場合は **Execute default flow** をクリックします。
実行中に **Execute default flow** を選択した場合、指定した条件に true となるものがなければ、インテグレーションはデフォルトの条件付きフローを実行します。実行中に **Execute default flow** を選択しなかった場合、指定した条件に true となるものがなければ、この **Conditional Flows** ステップに続くステップで、インテグレーションの実行が継続されます。

5. **Next** をクリックします。

6. 任意設定:Fuse Online で出力データタイプの指定を要求された場合は指定します。この **Conditional Flows** ステップの一部であるすべての条件付きフローは、同じ出力タイプである必要があります。

7. **Next** をクリックします。
   Fuse Online にフロービジュアライゼーションが表示されます。追加する **Conditional Flows** ステップの下に、指定した各条件のエントリーがあります。**Conditional Flows** ステップにデフォルトフローがあることを示した場合は、**Otherwise** デフォルトフローのエントリーも表示されます。

### 次のステップ

条件ごとに、条件に関連するフローにステップを追加します。デフォルトフローがある場合は、そのフローにステップを追加します。

### その他のリソース

Camel Simple Language Operator Support

## 4.7.6. 条件付きフローへのステップの追加

**Conditional Flows** ステップでは、条件を定義した後に、ステップを各条件に関連するフローに追加します。実行中、**Conditional Flows** ステップによって条件が true であると評価されると、その条件に関連するフローが実行されます。

### 前提条件

- この **Conditional Flows** ステップの条件が定義済みである必要があります。
- インテグレーションによってこの **Conditional Flows** ステップに渡されるメッセージに含まれるフィールドを熟知している必要があります。
- 条件付きフローに追加する各コネクションが作成されている必要があります。

### 手順

1. インテグレーションビジュアライゼーションで、フローを追加する条件に対して **Open Flow** をクリックします。
   Fuse Online は、ページ上部付近にその条件を表示します。条件付きフロービジュアライゼーションは、すべての条件付きフローにある **Flow Start** ステップおよび **Flow End** ステップを表示します。

2. フロービジュアライゼーションの、この条件付きフローにステップを追加する箇所で  をクリックします。

3. 追加するステップをクリックします。プライマリーフローに追加できるコネクションまたはステップを追加できます。
   **Flow Start** ステップからの出力は、この **Conditional Flows** ステップの前にあるプライマリーフローステップからの出力と常に同じです。たとえば、フィルターステップまたはデータマッパーステップをこの条件付きフローに追加する場合、利用可能なフィールドはプライマリーフローで利用可能なフィールドと同じになります。

4. 必要に応じて手順を設定します。

5. この条件付きフローに追加するステップごとに、前述の 3 つの手順を繰り返します。

6. ページ上部の **Flow** フィールドで、下矢印をクリックし、**Back to primary flow** をクリックします。これにより、条件付きフローが保存され、プライマリーフローが表示されます。

7. 追加する条件付きフローごとに、この手順を繰り返し行います。

**結果**

プライマリーフローには、**Conditional Flows** ステップで定義した条件ごとに条件付きフローが存在します。**Execute default flow** オプションを選択した場合、プライマリーフローにはデフォルトの条件付きフローも存在します。

実行中、**Conditional Flows** ステップは true となる最初の条件に関連する条件付きフローを実行します。その後、インテグレーションによって **Conditional Flows** ステップに続くステップが実行されます。

true となる条件がない場合、**Conditional Flows** ステップはデフォルトの条件付きフローを実行します。その後、インテグレーションによって **Conditional Flows** ステップに続くステップが実行されます。

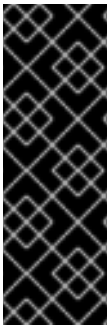
以下の条件を両方満たすとします。

- true となる条件がない
- デフォルトの条件付きフローがない

この場合、インテグレーションは **Conditional Flows** ステップに続くステップを実行します。

## 4.8. データマッパーステップの追加

ほぼすべてのインテグレーションにはデータマッピングが必要になります。データマッパーステップは、前のコネクションおよびその他のステップのデータフィールドからフローの次のコネクションが操作できるデータフィールドにマップします。たとえば、インテグレーションデータに **Name** フィールドが含まれ、フローの次のコネクションに **CustomerName** フィールドがある場合に、ソースの **Name** フィールドをターゲットの **CustomerName** フィールドにマップする必要があります。



**重要**

データマッパーでは、以前のインテグレーションステップで提供できるソースフィールドの最大セットが表示されます。ただし、すべてのコネクションによって、表示された各ソースフィールドにデータが提供されるわけではありません。たとえば、サードパーティーアプリケーションへの変更によって、特定のフィールドにデータが提供されなくなる可能性があります。インテグレーションの作成時に、データマッピングが想定どおりに機能していないことが判明した場合は、マップするソースフィールドに予想されるデータが含まれていることを確認してください。

**前提条件**

フローを作成または編集することになります。

**手順**

1. フロービジュアライゼーションの、データマッパーステップを追加する箇所で  をクリックします。

2. **Data Mapper** をクリックして、データマッパーキャンバスにソースおよびターゲットフィールドを表示します。

### 次のステップ

インテグレーションデータを次のコネクションのフィールドにマッピング を参照してください。

## 4.9. 基本のフィルターステップの追加

ステップのフローを追加して、フローが操作するデータをフィルターできます。フィルターステップでは、Fuse Online はデータを検査し、コンテンツが定義した基準を満たしている場合にのみ継続されます。たとえば、Twitter からデータを取得するフローでは、Red Hat が含まれるツイートのみ操作して、実行の継続を指定できます。

### 前提条件

- フローには必要なコネクションがすべて含まれている必要があります。
- フローを作成または編集することになります。

### 手順

1. フロービジュアライゼーションの、フィルターステップを追加する箇所で  をクリックします。
2. **Basic Filter** をクリックします。
3. **Configure Basic Filter Step** ページの **Continue only if incoming data match**フィールドで以下を行います。
   - 定義されたすべてのルールを満たす必要があるデフォルトを使用します。
   - あるいは、**ANY of the following** を選択して、1 つのルールのみを満たす必要があることを示します。
4. フィルタールールを定義します。
   a. **Property Name** フィールドで、フィルターによって評価される内容が含まれるフィールドの名前を入力または選択します。たとえば、ステップに送信されるデータがご自分の Twitter ハンドルをメンションするツイートで設定されるとします。また、ツイートに特定の内容が含まれる場合のみ実行を継続するとします。ツイートは **text** という名前のフィールドにあるため、**text** を Property Name フィールドの値として入力または選択します。プロパティー名は以下の方法で定義できます。
      - 入力を開始します。フィールドには、ポップアップボックスに補完の候補が表示される自動補完機能があります。ボックスから適切な候補を選択します。
      - フィールドをクリックします。ドロップダウンボックスが表示され、利用可能なプロパティーのリストが表示されます。リストから対象のプロパティーを選択します。
   b. **Operator** フィールドで、ドロップダウンボックスから演算子を選択します。デフォルト設定は **Contains** です。実行を継続するには、このフィールドで選択する条件が、**Keywords** フィールドに入力する値に対して true である必要があります。

c. **Keywords** フィールドに、絞り込む値を入力します。たとえば、デフォルトの **Contains** 演算子を使用し、受信テキストに特定の製品が含まれる場合のみインテグレーションの実行を継続するとします。この場合、このフィールドに製品名を入力します。

5. 必要に応じて、**+ Add another rule** クリックして、別のルールを定義します。
ルールエントリーの右上にあるごみ箱アイコンをクリックすると、ルールを削除できます。

6. フィルターステップが完了したら、**Done** をクリックしてフローに追加します。

**その他のリソース**

- 演算子の詳細や、評価するテキストを指定する例については Camel Simple Language Operator Support を参照してください。基本フィルターステップの **matches** 演算子は、Simple 言語の **regex** 演算子に対応することに注意してください。
- 基本のフィルターステップに必要なフィルターを定義できない場合は、高度なフィルターステップの追加 を参照してください。

## 4.10. 高度なフィルターステップの追加

フィルターステップでは、Fuse Online はデータを検査し、コンテンツが定義した基準を満たしている場合にのみフローの実行を継続します。基本のフィルターステップでは必要なフィルターを定義できない場合は、高度なフィルターステップを追加します。

**前提条件**

- フローには必要なコネクションがすべて含まれている必要があります。
- フローを作成または編集することになります。
- Camel Simple 言語を熟知するか、フィルター言語が提供されている必要があります。

**手順**

1. フロービジュアライゼーションの上級なフィルターステップを追加する場所で  をクリックします。
2. **Advanced Filter** をクリックします。
3. 編集ボックスで Camel Simple 言語 を使用してフィルター式を指定します。たとえば、メッセージヘッダーの **type** フィールドが **widget** に設定されている場合、以下の式は true に評価されます。

```
${in.header.type} == 'widget'
```

以下の例では、メッセージのボディーに **title** フィールドが含まれる場合に式が true に評価されます。

```
${in.body.title}
```

4. **Next** をクリックして、高度なフィルターステップをフローに追加します。

**フローでの追加プロパティーの指定**

式では、高度なフィルターステップが以下のようなフローの場合、追加のプロパティーを指定する必要があります。

- API プロバイダーインテグレーションのオペレーションフロー。
- Webhook コネクションで始まるシンプルなインテグレーション。
- カスタム REST API コネクションで始まるシンプルなインテグレーション。

これらのフローでは、Fuse Online は **body** プロパティー内で実際のメッセージコンテンツをラッピングします。これは、高度なフィルターへの入力に **body** プロパティーが含まれ、このプロパティーには実際のメッセージコンテンツが含まれる別の **body** プロパティーが含まれることを意味します。そのため、このようなフローの 1 つにある高度なフィルター式に、**body** のインスタンスを 2 つを指定する必要があります。たとえば、入力メッセージの **completed** フィールドにあるコンテンツを評価するとします。この場合、以下のように式を指定します。

```
${body.body.completed} = 1
```

## 4.11. テンプレートステップの追加

フローでは、テンプレートステップはソースからデータを取得し、Fuse Online にアップロードするテンプレートで定義された形式に挿入します。テンプレートステップの利点は、指定した一貫性のある形式でデータの出力を提供できることです。

テンプレートでは、プレースホルダーを定義して静的テキストを指定します。フローの作成時に、テンプレートステップを追加してソースフィールドをテンプレートプレースホルダーにマップし、テンプレートコンテンツをフローの次のステップにマップします。Fuse Online がフローを実行するとき、マップされたソースフィールドにある値がテンプレートのインスタンスに挿入されるため、フローの次のステップで利用できるようになります。

フローにテンプレートステップが含まれる場合、そのフローの唯一のテンプレートステップとなる可能性が高くなります。ただし、フローに複数のテンプレートステップを含めることも可能です。

Fuse Online は、Freemarker、Mustache、および Velocity のテンプレートをサポートします。

**前提条件**

- フローを作成または編集する必要があります。
- シンプルなインテグレーションを作成する場合は、すでに最初と最後のコネクションがある必要があります。

**手順**

1. フロービジュアライゼーションのテンプレートステップを追加する場所で  をクリックします。
2. **Template** をクリックします。**Upload Template** ページが開きます。
3. Freemarker、Mustache、または Velocity をテンプレートタイプとして指定します。
4. テンプレートを定義するには、以下のいずれかを行います。

- テンプレートファイルまたはテンプレートを作成するために編集するテキストが含まれるファイルを、テンプレートエディターにドラグアンドドロップします。
- **browse to upload** をクリックしてファイルを選択し、アップロードします。
- テンプレートエディターで、テンプレートを定義します。

5. テンプレートエディターでは、テンプレートが Fuse Online で使用できるようにしてください。有効なテンプレートの例は、この手順の後に記載されています。Fuse Online では、構文エラーが含まれる行の左側に  が表示されます。構文エラーインジケーターにマウスオーバーすると、エラーを解決するためのヒントが表示されます。

6. **Done** をクリックして、フローにテンプレートステップを追加します。
**Done** ボタンが有効でない場合は、修正する必要のある構文エラーが 1 つ以上存在します。

   テンプレートステップへの入力は、JSON オブジェクトの形式である必要があります。そのため、データマッピングステップをテンプレートステップの前に追加する必要があります。

7. テンプレートステップの前にデータマッパーステップを追加するには、以下を行います。

   a. フロービジュアライゼーションで、先ほど追加したテンプレートステップの直前にある  をクリックします。

   b. **Data Mapper** をクリックします。

   c. データマッパーで、ソースフィールドを各テンプレートプレースホルダーフィールドにマップします。
   たとえば、この手順の後に記載されているテンプレート例を使用して、ソースフィールドを以下のテンプレートフィールドにマップします。

      - **time**
      - **name**
      - **text**

   d. 右上の **Done** をクリックし、データマッパーステップをフローに追加します。

   テンプレートステップからの出力は常に JSON オブジェクトになります。そのため、テンプレートステップの後にデータマッパーステップを追加する必要があります。

8. テンプレートステップの後にデータマッパーステップを追加するには、以下を行います。

   a. フロービジュアライゼーションで、先ほど追加したテンプレートステップの直後にある  をクリックします。

   b. **Data Mapper** をクリックします。

   c. データマッパーで、テンプレートの **message** フィールドをターゲットフィールドにマップします。message フィールドには常にソースフィールドをテンプレートに挿入した結果が含まれます。たとえば、フローの次のコネクションが Gmail コネクションで、テンプレートステップの結果を Gmail メッセージの内容として送信するとします。これには、**message** ソースフィールドを **text** ターゲットフィールドにマップします。

d. 右上の **Done** をクリックします。

### テンプレートの例

Mustache テンプレートの例:

```
At {{time}}, {{name}} tweeted:
{{text}}
```

Freemarker および Velocity では、以下のテンプレート例がサポートされます。

```
At ${time}, ${name} tweeted:
${text}
```

Velocity では、以下の例のようにかっこを使用しない構文もサポートされます。

```
At $time, $name tweeted:
$text
```

プレースホルダーに **.** (ピリオド) を使用することはできません。

### その他のリソース

フィールドのマッピングに関する詳細は、インテグレーションデータを次のコネクションのフィールドにマッピング を参照してください。

## 4.12. カスタムステップの追加

Fuse Online がフローで必要なステップを提供しない場合、開発者はエクステンションでカスタムステップを 1 つ以上定義できます。カスタムステップは、フローのコネクション間のデータで操作します。

カスタムステップは、組み込みのステップを追加するのと同じ方法でフローに追加します。シンプルなインテグレーションでは、最初と最後のコネクションを選択し、必要に応じて他のコネクションを追加した後、追加のステップを追加します。API プロバイダーインテグレーションでは、フローがカスタムステップを実行するオペレーションを選択し、必要に応じてコネクションをフローに追加した後、他のステップを追加します。ステップを追加すると、Fuse Online はフローの前のステップから受信するデータで操作します。

### 前提条件

- Fuse Online にカスタムステップのエクステンションがアップロードされている必要があります。カスタム機能の使用 を参照してください。
- フローを作成または編集することになります。
- フローに必要なコネクションがすべて存在する必要があります。

### 手順

1. フロービジュアライゼーションのカスタムステップを追加する場所で  をクリックします。
2. 追加するカスタムステップをクリックします。

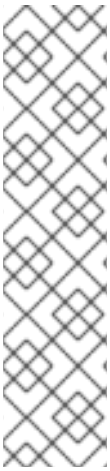
使用できるステップには、Fuse Online 環境にアップロードされたエクステンションで定義されたカスタムステップが含まれます。

3. ステップの実行に必要な情報のプロンプトに応答します。この情報はカスタムステップごとに異なります。

# 第5章 REST API 呼び出しによってトリガーされるインテグレーションの作成

必要時にインテグレーションの実行をトリガーするには、ユーザーが提供する REST API 記述ドキュメントでインテグレーションを開始します。この方法で開始するインテグレーションは、**API プロバイダーインテグレーション** と呼ばれます。API プロバイダーインテグレーションでは、REST API クライアントはインテグレーションの実行をトリガーするコマンドを呼び出しできます。

Fuse Online が API プロバイダーインテグレーションをパブリッシュすると、インテグレーションエンドポイントにネットワークアクセスできるクライアントはすべてインテグレーションの実行をトリガーできます。



**注記**

Fuse Online on OpenShift Container Platform をオンサイトで使用している場合、管理者は Fuse Online サーバーを設定して Red Hat 3scale の API プロバイダーインテグレーション API の検出を有効にすることができます。デフォルトでは、Fuse Online は 3scale と使用するために API プロバイダーインテグレーションの API サービス定義にアノテーションを付けますが、これらの API を 3scale の自動検出に公開しません。3scale の検出が行われないと、アクセス制御が行われません。3scale の検出を使用すると、アクセスポリシーを設定したり、管理を一元化でき、API プロバイダーインテグレーション API の高可用性を提供することもできます。詳細は、Red Hat 3scale のドキュメントページ の API Gateway に関するドキュメントを参照してください。

API の 3scale 検出を有効化する Fuse Online の設定 も参照してください。

API プロバイダーインテグレーションを作成するための情報および手順は以下を参照してください。

- 「API プロバイダーインテグレーションを作成する利点、概要、およびワークフロー」
- 「OpenAPI オペレーションを API プロバイダーインテグレーションフローと関連させる方法」
- 「API プロバイダーインテグレーションの作成」
- 「API プロバイダーインテグレーションのオペレーションフローの定義」
- 「API プロバイダークイックスタートインテグレーションの例のインポートおよびパブリッシュ」
- 「API プロバイダークイックスタートインテグレーションの例のテスト」

API プロバイダーインテグレーションの作成、パブリッシュ、およびテスト方法については、https://youtu.be/sox8SSqJ0zQ の動画を参照してください。

## 5.1. API プロバイダーインテグレーションを作成する利点、概要、およびワークフロー

API プロバイダーインテグレーションは、REST API サービスから開始します。この REST API サービスは、API プロバイダーインテグレーションの作成時に提供する OpenAPI 3 (または 2) ドキュメントによって定義されます。API プロバイダーインテグレーションをパブリッシュした後、Fuse Online は REST API サービスを OpenShift にデプロイします。API プロバイダーインテグレーションの利点は、REST API クライアントがインテグレーションの実行をトリガーする呼び出しを実行できることです。

**複数の実行フロー**

API プロバイダーインテグレーションには、フローと呼ばれる複数の実行パスがあります。OpenAPI ドキュメントが定義する各オペレーションには独自のフローがあります。Fuse Online では、OpenAPI ドキュメントが定義する各オペレーションに対して、コネクションおよびその他のステップをそのオペレーションの実行フローに追加します。これらのステップは、特定のオペレーションに必要なデータを処理します。

### 実行フローの例

たとえば、Fuse Online によって利用可能になった REST API サービスを呼び出す人事アプリケーションがあるとします。新しい従業員を追加する操作が呼び出されたとします。この呼び出しを処理する操作のフローは、以下のとおりです。

- 新入社員のハードウェアに関する経費報告書を作成するアプリケーションに接続します。
- 新しいハードウェアを設定するための社内チケットを追加する SQL データベースに接続します。
- 新入社員にオリエンテーションの情報を提供するメッセージを送信する Google メールに接続します。

### 実行をトリガーする方法

インテグレーションの実行をトリガーする REST API を呼び出す方法は複数あります。これには以下が含まれます。

- データ入力を取得し、呼び出しを生成する Web ブラウザーページ。
- **curl** ユーティリティーなどの REST API を明示的に呼び出すアプリケーション。
- REST API を呼び出す他の API (Webhook など)。

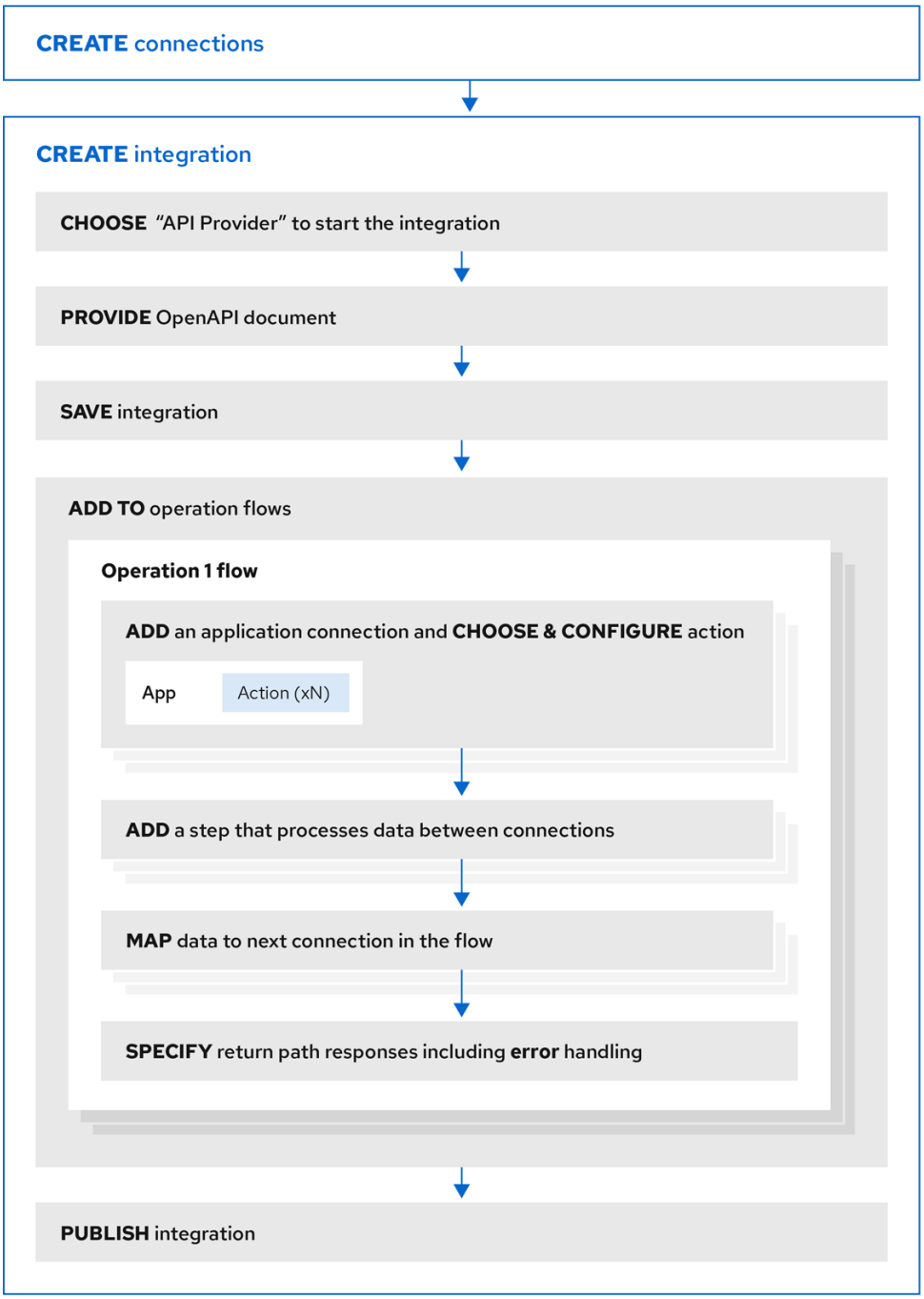
### フローを編集する方法

以下を行って、各オペレーションのフローを編集できます。

- データを処理する必要があるアプリケーションにコネクションを追加します。
- 分割、集計、およびデータマッピングステップを含む、コネクション間のステップを追加します。
- コネクションエラーメッセージを、フローを終了する HTTP 応答の戻りコードにマッピングします。この応答は、インテグレーションの実行をトリガーした呼び出しを実行したアプリケーションに送信されます。

### API プロバイダーインテグレーションを作成するためのワークフロー

API プロバイダーインテグレーションを作成するための **一般的な** ワークフローを以下の図に示します。




### API プロバイダーインテグレーションのパブリッシュ

API プロバイダーインテグレーションをパブリッシュした後、Fuse Online はインテグレーションの summary ページに REST API サービスの外部 URL を表示します。この外部 URL は、クライアントが REST API サービスを呼び出すために使用するベース URL です。

OCP 上の Fuse Online 環境では、Red Hat 3scale の API プロバイダーインテグレーションの検出が有効になっている可能性があります。この場合、3scale ではサービスを呼び出すための URL が公開されます。

### API プロバイダーインテグレーションのテスト

API プロバイダーインテグレーションのフローをテストするには、**curl** ユーティリティーを使用できます。たとえば、以下の **curl** コマンドは、REST API サービス URL https://i-task-api-proj319352.6a63.fuse-ignite.openshiftapps.com/api/ の **Get Task by ID** オペレーションに対し、フローの実行をトリガーします。

HTTP **GET** コマンドはデフォルトのリクエストであるため、**GET** を指定する必要はありません。URL の最後の部分は、取得するタスクの ID を指定します。

```
curl -k https://i-task-api-proj319352.6a63.fuse-ignite.openshiftapps.com/api/todo/1
```

## 5.2. OPENAPI オペレーションを API プロバイダーインテグレーションフローと関連させる方法

API プロバイダーインテグレーションの OpenAPI ドキュメントは、REST API クライアントが呼び出しできるオペレーションを定義します。各 OpenAPI オペレーションには、独自の API プロバイダーインテグレーションフローがあります。そのため、各オペレーションは独自の REST API サービス URL を持つこともできます。各 URL は API サービスのベース URL で定義され、任意でサブパスによって定義されます。REST API 呼び出しは、オペレーションの URL を指定し、そのオペレーションのフローの実行をトリガーします。

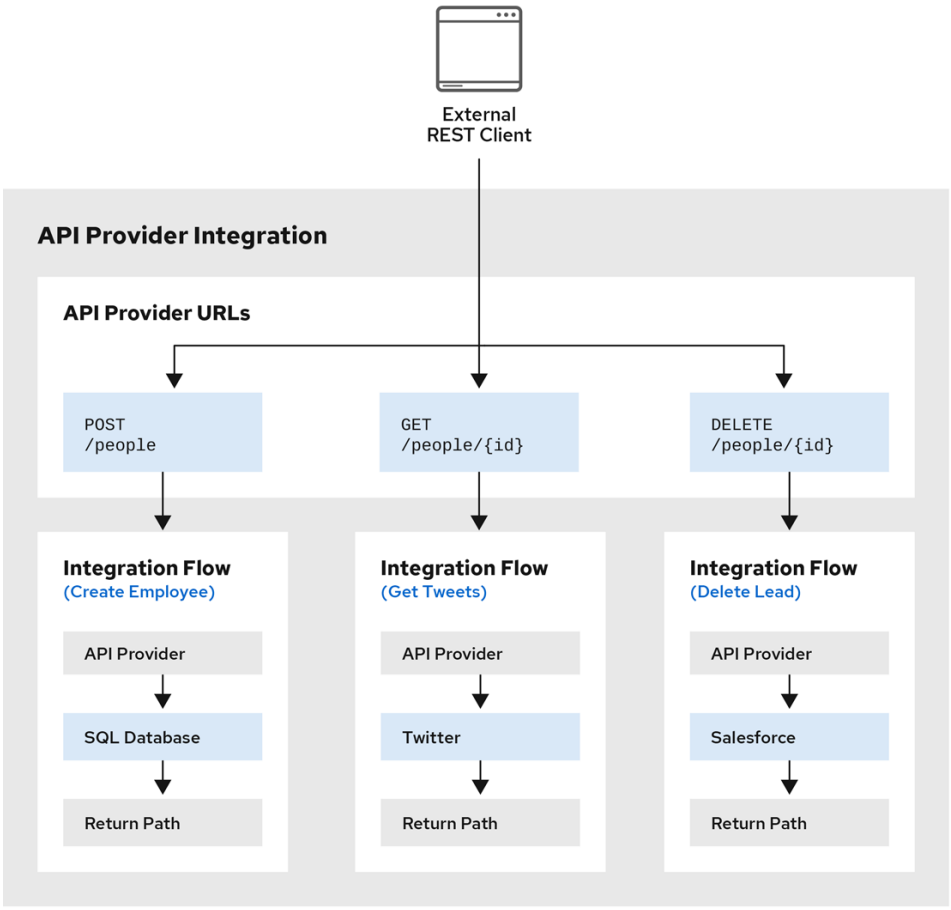
OpenAPI ドキュメントは、REST API サービス URL への呼び出しに指定できる HTTP 動詞 (**GET**、**POST**、**DELETE** など) を決定します。API プロバイダー URL への呼び出しの例は、 API プロバイダークイックスタートの例を試すための手順 を参照してください。

OpenAPI ドキュメントで、オペレーションによって返信可能な HTTP ステータスコードも判断されます。オペレーションのリターンパスでは、OpenAPI ドキュメントで定義される応答のみが処理されます。たとえば、ID を基にしてオブジェクトが削除されるオペレーションでは、以下のような応答を定義できます。

```
"responses": {
  "204": {
          "description": "Task deleted"
        },
  "404": {
          "description": "No Record found with this ID"
        },
  "500": {
           "description": "Server Error"
       }
}
```

### API プロバイダーインテグレーションの例

以下の図は、人に関するデータを処理する API プロバイダーインテグレーションを示しています。外部の REST API クライアントは、API プロバイダーインテグレーションによってデプロイされた REST API URL を呼び出します。URL が呼び出されると、1 つの REST オペレーションに対するフローの実行がトリガーされます。この API プロバイダーインテグレーションには 3 つのフローがあります。各フローは、Fuse Online で利用可能なすべてのコネクションまたはステップを使用できます。REST API とそのフローは、1 つの OpenShift Pod にデプロイされる Fuse Online API プロバイダーインテグレーションの 1 つです。




### API プロバイダーインテグレーションの作成時における OpenAPI ドキュメントの編集

API プロバイダーインテグレーションの OpenAPI ドキュメントを指定した後、API オペレーションの実行フローを定義するときに必要に応じてドキュメントを更新できます。これには、API プロバイダーインテグレーションを編集するページの右上にある **View/Edit API Definition** をクリックします。これにより、API Designer エディターに OpenAPI ドキュメントが表示されます。ドキュメントを編集および保存し、Fuse Online で変更が反映されるようにします。

OpenAPI ドキュメントの編集時に以下を考慮します。

- **同期化の operationId プロパティー。**
  API Designer エディターの OpenAPI ドキュメントのバージョンと、Fuse Online インテグレーションエディターの OpenAPI ドキュメントのバージョンの同期は、ドキュメントに定義される各オペレーションに割り当てられた一意な **operationId** プロパティーに応じて行われます。特定の **operationId** プロパティー値を各オペレーションに割り当てるか、Fuse Online が自動生成する値を使用します。

- **リクエストと応答の定義**
  オペレーションのリクエストおよび応答を定義する JSON スキーマを各オペレーションの定義に提供できます。Fuse Online は以下のように JSON スキーマを使用します。

  - オペレーションの入力および出力データシェイプのベースとして使用します。
  - データマッパーでオペレーションフィールドを表示するために使用します。

- **循環スキーマ参照がない**
  API プロバイダーインテグレーションオペレーションの JSON スキーマにはスキーマの循環参照を含めることはできません。たとえば、リクエストまたは応答ボディーを指定する JSON スキーマは、そのスキーマ自体を全体的に参照することはできず、中間スキーマを介してそのス

キーマ自体を部分的に参照することもできません。

## 5.3. API プロバイダーインテグレーションの作成

API プロバイダーインテグレーションを作成するには、インテグレーションが実行できるオペレーションを定義する OpenAPI ドキュメント (**.json**、**9.yaml**、または **.yml** ファイル) を提供します。Fuse Online は各オペレーションの実行フローを作成します。各オペレーションのフローを編集し、そのオペレーションの要件に応じてインテグレーションデータを処理するコネクションおよびステップを追加します。

**前提条件**

- インテグレーションが実行する REST API オペレーションの OpenAPI ドキュメントを提供または定義できる必要があります。
  検証するには、API プロバイダークイックスタートの OpenAPI ドキュメントである raw バージョンの **task-api.json** ファイルをダウンロード します。Fuse Online が OpenAPI ドキュメントの提供を要求したときに、このファイルをアップロードできます。この代わりに、raw の **task-api.json** ファイルである https://raw.githubusercontent.com/syndesisio/syndesis-quickstarts/1.15/api-provider/task-api.json を指定できます。
- 各 OpenAPI オペレーションのフローが計画されている必要があります。
- オペレーションのフローを追加する各アプリケーションまたはサービスのコネクションが作成済みである必要があります。

**手順**

1. Fuse Online の左ナビゲーションパネルで **Integrations** をクリックします。
2. **Create Integration** をクリックします。
3. **Choose a connection** ページで **API Provider** をクリックします。
4. **Start integration with an API call**ページで以下を行います。
   - REST API オペレーションを定義する OpenAPI ドキュメントがある場合は、OpenAPI ドキュメントをアップロードします。
   - OpenAPI ドキュメントを定義する必要がある場合は、**Create a new OpenAPI 3.x document** または **Create a new OpenAPI 2.x document**を選択します。
5. **Next** をクリックします。
   - ドキュメントをアップロードした場合は、これを確認または編集します。
     a. **Review/Edit** をクリックして API Designer エディターを開きます。
     b. 必要に応じて確認や編集を行います。
     任意の手順: ドキュメントによって OpenAPI 2 仕様が使用される場合に、API Designer でドキュメントを変換して OpenAPI 3 仕様に準拠するようにするには、**Convert to OpenAPI 3** をクリックします。
     c. 右上の **Save** または **Cancel** をクリックし、エディターを閉じます。
     d. **Next** をクリックします。

- ドキュメントを作成する場合は、Fuse Online で起動される API Designer エディターで以下を行います。

  a. API Designer を使用した API 定義の設計および開発 の説明どおりに OpenAPI ドキュメントを定義します。

  b. 右上の **Save** をクリックし、エディターを閉じます。

  c. **Next** をクリックします。

**結果**

Fuse Online は OpenAPI ドキュメントが定義するオペレーションの一覧を表示します。

**次のステップ**

それぞれのオペレーションでは、そのオペレーションを実行するフローを定義します。

## 5.4. API プロバイダーインテグレーションのオペレーションフローの定義

REST API サービスを定義する OpenAPI ドキュメントは、サービスが実行できるオペレーションを定義します。API プロバイダーインテグレーションの作成後、各オペレーションのフローを編集できます。

各オペレーションには必ず 1 つのフローがあります。オペレーションフローでは、コネクションを他のアプリケーションやサービスに追加でき、コネクション間のデータで操作するステップも追加できます。

オペレーションフローへ追加すると、API プロバイダーインテグレーションがベースとする OpenAPI ドキュメントの更新が必要であることがあります。これには、API プロバイダーインテグレーションを編集するページの右上にある **View/Edit API Definition**をクリックします。これにより、API Designer エディターにドキュメントが表示されます。OpenAPI 定義では、各オペレーションに固有の **operationId** プロパティーがある限り、API Designer に更新を保存することができ、Fuse Online は API プロバイダーインテグレーションのフロー定義を同期して更新が反映されるようにすることができます。

**前提条件**

- API プロバイダーインテグレーションを作成し、名前を付け、保存している必要があります。
- オペレーションフローが接続する各アプリケーションまたはサービスへのコネクションが作成済みである必要があります。詳細は、コネクションの作成に関する情報 を参照してください。
- Fuse Online は API が定義するオペレーションのリストを表示します。

**手順**

1. **Operations** リストページで、定義するフローのオペレーションに対して **Create flow** をクリックします。
2. このフローに追加する各コネクションに対して以下を行います。

   a. フロービジュアライゼーションで、プラス記号をクリックしてその場所にコネクションを追加します。

   b. 使用するコネクションをクリックします。

   c. コネクションが実行するアクションを選択します。

d. ラベルが付いたフィールドにデータを入力して、アクションを設定します。

e. **Next** をクリックします。

フローに必要なコネクションをすべて追加してから、続行します。

3. このオペレーションフローでコネクション間のデータを処理するには、以下を行います。

a. フロービジュアライゼーションで、ステップを追加する場所にあるプラス記号をクリックします。

b. 追加するステップをクリックします。

c. ラベル付が付いたフィールドにデータを入力して、ステップを設定します。

d. **Next** をクリックします。
ヘルプが必要な場合は コネクション間のステップの追加 を参照してください。

コネクションの間のデータを処理する別のステップを追加する場合は、この手順のサブセットを繰り返します。

4. データを次のコネクションのフィールドにマップします。

a. フロービジュアライゼーションで、コネクションが受信データを処理できないことを示す、データタイプ不一致の  アイコンを確認します。ここでは、データマッパーステップを追加する必要があります。

b. フロービジュアライゼーションの各データ不一致アイコンに対して以下を行います。

i. そのステップの直前にあるプラスマークをクリックします。

ii. **Data Mapper** をクリックします。

iii. 必要なマッピングを定義します。ヘルプが必要な場合は インテグレーションデータを次のコネクションのフィールドにマッピング を参照してください。

iv. **Done** をクリックして、データマッパーステップをフローに追加します。

5. フロービジュアライゼーションの、**Provided API Return Path**ステップで **Configure** をクリックします。
すべての API プロバイダーインテグレーションは、オペレーションフローの実行をトリガーした REST API の呼び出し元に応答を送信することで、各オペレーションフローを終了します。応答には、オペレーションのフローを終了する **Provided API Return Path**ステップのに設定した戻りコードの 1 つが含まれます。以下のように、Return Path ステップを設定します。

a. **Return Code** フィールドの **Default Response** で、Fuse Online に表示されるデフォルトの応答を指定するか、下向きのキャレットをクリックしてスクロールし、希望のデフォルト応答を選択します。オペレーションフローを実行しても設定されたエラー応答がどれも返されなかった場合、フローはこの応答を送信します。通常、デフォルト応答の戻りコードはオペレーションに成功したことを意味します。

b. 返されたメッセージのボディー部にエラーメッセージが含まれるようにするかどうかを **Error Handling** に指定します。
通常、開発中はエラーメッセージを返すようにします。しかし、実稼働では機密情報が含まれる場合にエラーメッセージを非表示にする場合があります。エラーメッセージは、**responseCode**、**category**、**message**、および **error** 要素が含まれる JSON 形式の文字列です。例を以下に示します。

```
{
  responseCode: 404,
  category:  "ENTITY_NOT_FOUND_ERROR",
  message: "SQL SELECT did not SELECT any records"
  error: SYNDESIS_CONNECTION_ERROR
 }
```

開発中、最も確実にエラーの発生を確認する方法は、呼び出し元への応答の **HTTP_RESPONSE STATUS** ヘッダーをチェックすることです。インテグレーション Pod のログで **INFO** メッセージを確認することもできます。インテグレーションの **Activity** ログには成功した交換が記録され、エラーは **Activity** ログに常に記録されるとは限りません。

c. **Error Response Codes** には、フローのコネクションが返す可能性がある各エラーのエントリーが表示されます。エラーごとに、デフォルトの戻りコードである **200 All is good** を指定するか、クリックして別の HTTP ステータスの戻りコードを選択します。
選択可能な戻りコードは、このフローで実行されるオペレーションのために OpenAPI ドキュメントで定義される戻りコードです。必要な戻りコードが Fuse Online に表示されない場合、OpenAPI ドキュメントを編集して追加できます。

これを行うには、右上の **View/Edit API Definition** をクリックします。必要に応じて OpenAPI ドキュメントを編集します。編集し終わったら、OpenAPI ドキュメントを保存します。Fuse Online が **Provided API Return Path** の編集に戻り、保存した変更が反映されます。

d. Next をクリックし、リターンパスの設定を完了します。

6. このフローに、必要なコネクションとステップがすべてあり、データの不一致アイコンがない場合や、現時点でフローを編集しない場合は、以下の 1 つを行います。

- **Publish**: インテグレーションの実行を開始するには、右上の **Publish** をクリックします。これにより、インテグレーションがビルドされ、REST API サービスが OpenShift にデプロイされます。さらにインテグレーションが実行できるようになります。オペレーションフローの作成を完了するときやオペレーションフローを編集するときに、インテグレーションをパブリッシュできます。

- **Save**: オペレーションのリストを表示するには、右上の **Save** をクリックします。

この手順を繰り返して、別のオペレーションフローを編集します。

### API プロバイダーインテグレーションのテスト

- 以下のプラットフォームの 1 つで稼働している API プロバイダーインテグレーションのテスト
  - OpenShift Online
  - OpenShift Dedicated
  - **API 検出が無効になっている** 場合の OpenShift Container Platform

  **curl** ユーティリティーを使用すると、インテグレーションが想定どおりに動作していることを確認できます。**curl** コマンドに、API プロバイダーインテグレーションのパブリッシュ後に Fuse Online に表示される外部 URL を指定します。この例については API プロバイダークイックスタートインテグレーションの例のテスト を参照してください。

- **API 検出が有効になっている** 場合の OpenShift Container Platform で稼働している API プロバイダーインテグレーションのテスト

Red Hat 3scale は API プロバイダーインテグレーションをパブリッシュします。インテグレーションをテストするには、3scale ダッシュボードを開き、インテグレーションの URL を取得します。

たとえば、Red Hat 3scale でインテグレーションの API へのアクセスを制御したくない場合や、Fuse Online で API プロバイダーをテストしたい場合は、API プロバイダーインテグレーションの検出を無効にできます。検出を無効にすると、インテグレーションが Fuse Online によって再パブリッシュされ、インテグレーションの実行を呼び出しおよびテストする外部 URL が提供されます。これには、Fuse Online でインテグレーションの概要ページに移動します。このページの **Disable discovery** をクリックします。Fuse Online によってインテグレーションが再パブリッシュされ、インテグレーションの URL が提供されます。インテグレーションのテスト方法の例は、API プロバイダークイックスタートインテグレーションの例のテスト を参照してください。テスト後に、API プロバイダーインテグレーションの検出を再度有効にすると、3scale でパブリッシュできるようになります。

各 API プロバイダーインテグレーションの検出を有効または無効にすることができます。

## 5.5. API プロバイダークイックスタートインテグレーションの例のインポートおよびパブリッシュ

Fuse Online は、Fuse Online 環境にインポートできる API プロバイダークイックスタートインテグレーションを提供します。このクイックスタートには、タスク管理 API の OpenAPI ドキュメントが含まれています。クイックスタートインテグレーションをインポートした後、フローを確認し、インテグレーションをパブリッシュします。以下の手順の完了後、**TaskAPI** インテグレーションは稼働状態になり、実行される準備が整います。

API プロバイダークイックスタートは、API プロバイダーインテグレーションを設定、パブリッシュ、およびテストする方法を短時間で理解するのに役立ちます。しかし、API プロバイダーインテグレーションが便利であることを実証する実際の例ではありません。実例として、Fuse Online をすでに使用して、複数のシンプルなインテグレーションをパブリッシュしている場合に、このようなインテグレーションの実行をトリガーするために、OpenAPI ドキュメントを定義できます。これには、パブリッシュ済みのシンプルなインテグレーションとほぼ同じになるように、各 OpenAPI オペレーションのフローを編集します。

**前提条件**

- Fuse Online がブラウザーで開かれている必要があります。
- OCP で実行している Fuse Online 環境へのサンプルデータの追加 で説明されているように、Fuse Online 環境には **Todo** サンプルアプリケーションとサンプル PostgreSQL データベースが含まれている必要があります。

**手順**

1. **TaskAPI** クイックスタートインテグレーションをインポートします。
   a. https://github.com/syndesisio/syndesis-quickstarts/api-provider にアクセスし、**TaskAPI-export.zip** をダウンロードします。
   b. Fuse Online の左側のナビゲーションパネルで **Integrations** をクリックします。
   c. 右上の **Import** をクリックします。
   d. ダウンロードした **TaskAPI-export.zip** ファイルを **Import** ページにドラッグします。Fuse Online は、ファイルが正常にインポートされたことを示します。

e. 左側のナビゲーションパネルで **Integrations** をクリックし、先ほどインポートした **TaskAPI** インテグレーションのエントリーを表示します。設定が必要であるとエントリーに表示されますが、このインテグレーションをパブリッシュする準備は整っています。

2. **TaskAPI** エントリーで  をクリックした後、**Edit** をクリックしてこの API によって提供されるオペレーションのリストを表示します。

3. 各オペレーションのフローを確認するには、以下を行います。

a. **Edit flow** ボタンをクリックし、そのフローのビジュアライゼーションを表示します。各フローにはすでに 1 つのデータベースコネクション、1 つ以上のデータマッパーステップ、およびフローを終了する 1 つの **Provided API Return Path** ステップが存在します。

b. **Invoke SQL** ステップでは **Configure** をクリックし、コネクションが実行する SQL ステートメントを表示します。その後、**Cancel** をクリックし、オペレーションのビジュアライゼーションフローに戻ります。

c. データマッパーステップでは、**Configure** をクリックし、マッピングを表示します。その後、**Cancel** をクリックしてビジュアライゼーションに戻ります。

d. 各オペレーションのフローの最終ステップである **Provided API Return Path** ステップでは、**Configure** をクリックし、オペレーションによって呼び出し元に送信された可能性がある HTTP リターンコードを表示します。**Cancel** をクリックしてビジュアライゼーションに戻ります。

e. オペレーションのフローの確認後、ドロップダウンメニューで **Integrations**> **TaskAPI**> **Operation** の順にクリックし、別のオペレーションを選択します。

f. この手順を繰り返し、各フローを確認します。

4. フローの確認後、**Publish** をクリックします。必要な場合はインテグレーション名を編集し、**Save and publish** をクリックします。
Fuse Online は、このインテグレーションの概要ページを表示し、アセンブル、ビルド、デプロイ、およびインテグレーションの実行中にパブリッシュの進捗を表示します。

5. **TaskAPI** インテグレーション概要ページに **Running** が表示されると、Fuse Online は Task API サービスの外部 URL を表示します。以下のような URL が表示されます。
**https://i-task-api-proj319352.6a63.fuse-ignite.openshiftapps.com/api/**

これは、Task API サービスを利用できる場所を示しています。REST API 呼び出しは、このベース URL で始まる URL を指定します。

Fuse Online を OpenShift Container Platform で使用し、外部 URL がインテグレーションの概要ページにない場合、Red Hat 3scale の検出は有効になっています。これは、Red Hat 3scale によってインテグレーションの API へのアクセスが制御され、API プロバイダーインテグレーションもパブリッシュされることを意味します。インテグレーションをテストするには、3scale ダッシュボードを開き、インテグレーションの URL を取得します。

インテグレーションの API へのアクセスを 3scale が制御しないようにするには、検出を無効にします。これは、Fuse Online でインテグレーションの概要ページを表示して行います。このページの **Disable discovery** をクリックします。インテグレーションが Fuse Online によって再パブリッシュされ、インテグレーションの実行を呼び出すための外部 URL が提供されます。

各 API プロバイダーインテグレーションの検出を有効または無効にすることができます。

## 5.6. API プロバイダークイックスタートインテグレーションの例のテスト

Fuse Online の **TaskAPI** クイックスタートインテグレーションの稼働時に、HTTP リクエストを Task API サービスに送信する **curl** ユーティリティーコマンドを呼び出すことができます。HTTP リクエストを指定する方法によって呼び出しがトリガーするフローが判断されます。

**前提条件**

- Fuse Online によって **TaskAPI** インテグレーションが **Running** 状態であることが示される必要があります。
- Fuse Online 環境が OCP で稼働している場合、Fuse Online は API を 3scale に公開するよう設定されないか、**TaskAPI** インレグレーションの検出が無効になっています。

**手順**

1. Fuse Online の左側のナビゲーションパネルで **Integrations** をクリックします。
2. **TaskAPI** インテグレーションエントリーで **View** をクリックし、インテグレーションの概要を表示します。
3. インテグレーションの外部 URL をコピーします。
4. ターミナルで以下のようなコマンドを実行し、インテグレーションの外部 URL を **externalURL** 環境変数に割り当てます。必ず、このサンプルコマンドの URL を、コピーした URL に置き換えてください。

```
export externalURL="https://i-task-api-proj319352.6a63.fuse-ignite.openshiftapps.com/api"
```

5. **Create new task** オペレーションに対してフローの実行をトリガーする **curl** コマンドを実行します。

```
curl -k --header "Content-Type: application/json" --request POST --data '{ "task":"my new task!"}' $externalURL/todo
```

   - **-k** を指定すると、サーバーコネクションがセキュアでなくても **curl** は続行および動作します。
   - **--header** は、コマンドが JSON 形式のデータを送信することを示します。
   - **--request** は、データを格納する HTTP **POST** コマンドを指定します。
   - **--data** は、保存する JSON 形式のコンテンツを指定します。この例では、コンテンツは **{ "task":"my new task!"}** になります。
   - **$externalURL/todo** は呼び出す URL です。
     このコマンドは、HTTP **POST** リクエストを、**Create new Task** オペレーションのフローの実行をトリガーする Task API サービスに送信します。フロー実行により、新しいタスクがサンプルデータベースに追加され、以下のようなメッセージを返して実行された内容を示します。

```
{"completed":false,"id":1,"task":"my new task!"}
```

6. ID オペレーションによる **Fetch task** のフローの実行をトリガーする **curl** コマンドを実行します。

```
curl -k $externalURL/todo/1
```

タスクを取得するには、**curl** コマンドに URL のみを指定する必要があります。HTTP **GET** コマンドはデフォルトのリクエストです。URL の最後の部分は、取得するタスクの ID を指定します。

7. ID オペレーションに対する **Delete Task** のフローの実行をトリガーする **curl** コマンドを実行します。

```
curl -k -X DELETE $externalURL/todo/1
```

このコマンドは、ID でタスクを取得したコマンドと同じ URL で HTTP **DELETE** コマンドを実行します。

# 第6章 HTTP リクエスト (WEBHOOK) によってトリガーされるインテグレーションの作成

HTTP **GET** または **POST** リクエストを Fuse Online が公開する HTTP エンドポイントに送信して、シンプルなインテグレーションの実行をトリガーできます。詳細は以下のセクションを参照してください。

- 「Fuse Online Webhook を使用するための一般的な手順」
- 「HTTP リクエストがトリガー可能なインテグレーションの作成」
- 「Fuse Online による HTTP リクエストの処理方法」
- 「Fuse Online Webhook を呼び出す HTTP クライアントのガイドライン」
- 「リクエストパラメーターを指定するための JSON スキーマ」
- 「HTTP リクエストの指定方法」

## 6.1. FUSE ONLINE WEBHOOK を使用するための一般的な手順

HTTP **GET** または **POST** リクエストでインテグレーションの実行をトリガーするには、以下を行う必要があります。

1. **GET** または **POST** リクエストを Fuse Online に送信するかどうかを決定します。
2. このリクエストを処理するようインテグレーションを計画します。
3. インテグレーションを終了するコネクションを作成します。
   Fuse Online は、最初のコネクションとして使用する Webhook コネクションを提供します。
4. インテグレーションに追加する他のコネクションを作成します。
5. インテグレーションを作成します。
   a. Webhook コネクションを最初のコネクションとして追加します。
   b. 最後のコネクションを追加した後、インテグレーションで必要な他のコネクションを追加します。最後のコネクションと途中のコネクションは、インテグレーションの実行をトリガーする HTTP リクエストを処理します。目的を達成するために最も適切な HTTP リクエストを選択および指定するのはユーザー自身です。これには以下を考慮してください。
      - 取得または更新するデータが含まれるアプリケーションへのコネクションを追加します。
      - **GET** リクエストは、キー/値パラメーターの指定に限定されます。
      - **POST** リクエストは、XML や JSON インスタンスなどの任意のボディーを提供します。
      - Fuse Online は HTTP ステータスヘッダーのみを返し、データは返しません。そのため、**GET** リクエストによってトリガーされるインテグレーションや、データを取得せずにデータを更新するインテグレーションを定義できます。同様に、**POST** リクエストによってトリガーされるインテグレーションや、データを更新せずにデータを取得するインテグレーションを定義することもできます。

c. Webhook コネクションの後にデータマッパーステップを追加します。
**GET** リクエストでは、HTTP リクエストのパラメーターフィールドを次のコネクションのデータフィールドにマップします。

**POST** リクエストの場合、JSON インスタンス、JSON スキーマ、XML インスタンス、XML スキーマ、または CSV スキーマを渡して、リクエストに出力データシェイプを指定した可能性があります。指定しなかった場合は、Webhook コネクションをインテグレーションの最初のコネクションとして追加します。指定しないと、Webhook コネクションの出力データタイプのデフォルトは JSON 形式になります。

d. インテグレーションに必要な他のステップを追加します。

6. インテグレーションをパブリッシュし、**Running** 状態になるまで待ちます。

7. インテグレーション概要ページに移動し、Fuse Online が提供する外部 URL をコピーします。

8. 外部 URL を編集して、**GET** または **POST** リクエストを作成します。

9. HTTP **GET** または **POST** リクエストを Fuse Online に送信するアプリケーションを実装します。

## 6.2. HTTP リクエストがトリガー可能なインテグレーションの作成

HTTP **GET** または **POST** リクエストでインテグレーションの実行をトリガーするには、Webhook コネクションをインテグレーションの最初のコネクションとして追加します。

**手順**

1. Fuse Online パネルの左側にある **Integrations** をクリックします。

2. **Create Integration** をクリックします。

3. **Choose a connection** ページで Webhook コネクションをクリックします。

4. **Choose an action** ページで **Incoming Webhook** アクションを選択します。
**Webhook Configuration** ページで、Fuse Online はこのインテグレーションのために生成した Webhook トークンを表示します。

HTTP リクエストを作成するとき、このトークンは URL の最後の部分になります。このインテグレーションをパブリッシュし、稼働状態になった後、Fuse Online はこのトークンが末尾にある Fuse Online 外部 URL を表示します。

**Webhook Configuration** ページには、**Default Response** および **Error Handling** セクションも含まれます。Webhook ステップは、これを呼び出す HTTP クライアントに応答を送信します。応答には戻りコードの 1 つが含まれ、デフォルトでは、返されるメッセージのボディー部にあるエラーメッセージが含まれます。

5. **Return Code** フィールドの **Default Response** で、Fuse Online に表示されるデフォルトの応答を指定するか、ドロップダウンリストを使用して希望のデフォルト応答を選択します。オペレーションフローを実行しても設定されたエラー応答がどれも返されなかった場合、フローはこの応答を送信します。通常、デフォルト応答の戻りコードはオペレーションに成功したことを意味します。

6. 返されたメッセージのボディー部にエラーメッセージが含まれるようにするかどうかを **Error Handling** に指定します。
通常、開発中はエラーメッセージを返すようにします。しかし、実稼働では機密情報が含まれ

る場合にエラーメッセージを非表示にする場合があります。エラーメッセージは、**responseCode**、**category**、**message**、および **error** 要素が含まれる JSON 形式の文字列です。例を以下に示します。

```
{
  responseCode: 404,
  category:  "ENTITY_NOT_FOUND_ERROR",
  message: "SQL SELECT did not SELECT any records"
  error: SYNDESIS_CONNECTION_ERROR
 }
```

開発中、最も確実にエラーの発生を確認する方法は、呼び出し元への応答の **HTTP_RESPONSE STATUS** ヘッダーをチェックすることです。インテグレーション Pod のログで **INFO** メッセージを確認することもできます。インテグレーションの **Activity** ログには成功した交換が記録され、エラーは **Activity** ログに常に記録されるとは限りません。

7. Webhook ステップが返す可能性のあるエラーごとに、デフォルトの戻りコードを指定するか、ドロップダウンリストを使用して別の HTTP ステータスの戻りコードを選択します。
8. **Next** をクリックします。
9. **Specify Output Data Type** ページで以下を行います。
    a. **Select Type** フィールドをクリックし、**JSON schema** を選択します。
    b. **Definition** フィールドに、HTTP リクエストでパラメーターのデータタイプを定義する JSON スキーマを貼り付けます。リクエストパラメーターを指定するための JSON スキーマ を参照してください。
    c. **Data Type Name** フィールドに、このデータタイプの名前を指定します。これは任意のフィールドですが、名前を指定すると、データマッパーの **Sources** リストに表示されるため、フィールドを正しくマップしやすくなります。
    d. 必要に応じて、このデータタイプを区別するための情報を **Data Type Description** フィールドに入力します。
    e. **Next** をクリックします。
10. 最後のコネクションをインテグレーションに追加します。
11. 必要な他のコネクションを追加します。
12. 必要な他のステップを追加します。
13. 最初のコネクションの直後に、データマッパーステップを追加します。
14. **Publish** をクリックし、インテグレーションに名前を付け、必要に応じて説明、カスタム環境変数、および 1 つ以上のラベルを付けます。
15. **Save and publish** をクリックします。

## 6.3. FUSE ONLINE による HTTP リクエストの処理方法

HTTP **GET** または **POST** リクエストを指定して、シンプルなインテグレーションの実行をトリガーできます。通常 **GET** リクエストはデータを取得し、**POST** リクエストはデータを更新しますが、いずれかのリクエストを使用して、いずれかのオペレーションを行うインテグレーションをトリガーできま

す。リクエストのパラメーターはすべてインテグレーションの次のコネクションにあるデータフィールドへのマッピングに利用できます。詳細は リクエストパラメーターを指定するための JSON スキーマ を参照してください。

Webhook コネクションは受信するデータのみをインテグレーションの次のコネクションに渡します。Fuse Online が HTTP リクエストを受信すると、以下を行います。

- HTTP ステータスヘッダーを要求元に返します。リクエストがインテグレーションの実行を正常にトリガーした場合、Fuse Online の戻りコードは **201** になります。リクエストがインテグレーション実行のトリガーに失敗した場合、戻りコードは **5xx** になります。
- 他のデータを要求元に返しません。ステータスヘッダーが含まれる 応答 の HTTP ボディーにはデータがありません。
- リクエストのデータをインテグレーションの次のコネクションに渡します。

そのため、**GET** リクエストによってトリガーされるシンプルなインテグレーションや、データを取得せずにデータを更新するインテグレーションを定義できます。同様に、**POST** リクエストによってトリガーされるインテグレーションや、データを更新せずにデータを取得するインテグレーションを定義することもできます。

**注記**

インテグレーションの **Activity** タブでは、Webhook ステップのステータスは毎回 **Success** になります。この **Success** ステータスは、Fuse Online} Webhook と、それを呼び出す HTTP クライアントとの間の通信の状態を示します。この **Success** ステータスは、インテグレーションが正常に渡されたことや、ステップにエラーがないことを示すわけではありません。HTTP リクエストによって生成されたエラーは、インテグレーションの **Activity** ログには記録されません。

Webhook を設定する場合、**Include error message in the return body**オプションはデフォルトでチェックされます。このオプションにチェックマークを入れると、HTTP リクエストによって生成されたエラーが Webhook 応答に含まれているかどうかを確認するため、エラーを生成して応答ヘッダーをチェックするテストリクエストを送信します。インテグレーション Pod のログで **INFO** メッセージを確認することもできます。以下のコマンドを使用してインテグレーションの Pod ログを表示します。**example-integration-pod** は Pod の名前になります。

```
oc logs -f pod/example-integration-pod
```

## 6.4. FUSE ONLINE WEBHOOK を呼び出す HTTP クライアントのガイドライン

HTTP リクエストを Fuse Online に送信するクライアントを実装する場合、実装は以下を行う必要があります。

- **GET** または **POST** リクエストを作成する URL を作成するため、Fuse Online が提供する外部 URL に追加します。
- URL リクエストに、**io:syndesis:webhook** JSON スキーマに準拠するデータタイプを持つ、HTTP ヘッダーとクエリーパラメーターの値を指定します。リクエストパラメーターを指定するための JSON スキーマ を参照してください。ヘッダーとクエリーパラメーターがこのデータタイプの指定に準拠する場合、パラメーターフィールドをインテグレーションの次のコネクションが処理できるフィールドにマップできます。

- リクエストに成功した場合、返された成功コード **201** を処理します。
- リクエストに失敗した場合、HTTP の **5xx** エラーコードを処理します。
- Fuse Online からの他の応答を想定しません。そのため、リクエストを送信しても戻りコード以外の戻りデータは直接要求元のクライアントに返されません。

## 6.5. リクエストパラメーターを指定するための JSON スキーマ

インテグレーションでは、通常 HTTP リクエストのヘッダーおよびクエリーパラメーターをインテグレーションの次のコネクションで処理できるデータフィールドにマップします。これを可能にするには、Webhook コネクションをインテグレーションに追加するときに、以下の構造を持つ JSON スキーマの出力データタイプを指定します。

```
{
  "$schema": "http://json-schema.org/schema#",
  "id": "io:syndesis:webhook",
  "type": "object",
  "properties": {
    "parameters": {
      "type": "object",
      "properties": { 1
      }
    },
    "body": {
      "type": "object",
      "properties": { 2
      }
    }
  }
}
```

必要なデータ構造を追加するには、HTTP リクエストの JSON インスタンスで以下を行います。

1. **parameters** オブジェクト下の **properties** セクションにクエリーパラメーターを指定します。
2. **body** オブジェクト下の **properties** セクションに HTTP ボディースキーマを指定します。

HTTP クライアントが送信するデータはすべてインテグレーションで利用できますが、Webhook コネクションのデータシェイプがこの JSON スキーマに準拠する場合はクエリーパラメーターとボディーのコンテンツをマッピングに使用できます。

例については HTTP リクエストの指定方法 を参照してください。

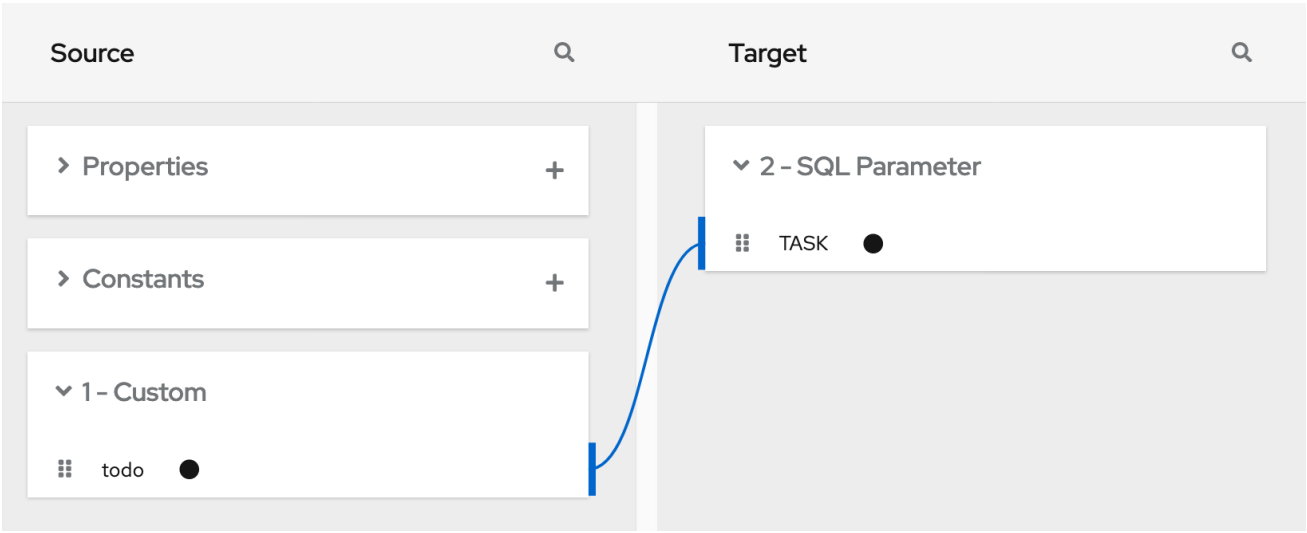
## 6.6. HTTP リクエストの指定方法

以下の例は、Fuse Online Webhook に HTTP リクエストを指定する方法を示しています。

### Webhook における HTTP ボディーのみの POST リクエストの例

Webhook コネクションで開始し、Fuse Online が提供するデータベースの **Todo** テーブルの行を作成するインテグレーションについて考えます。

このインテグレーションの作成中、Webhook の最初のコネクションを追加するときに **{"todo":"text"}** がコンテンツにある JSON インスタンスで出力データタイプを指定します。

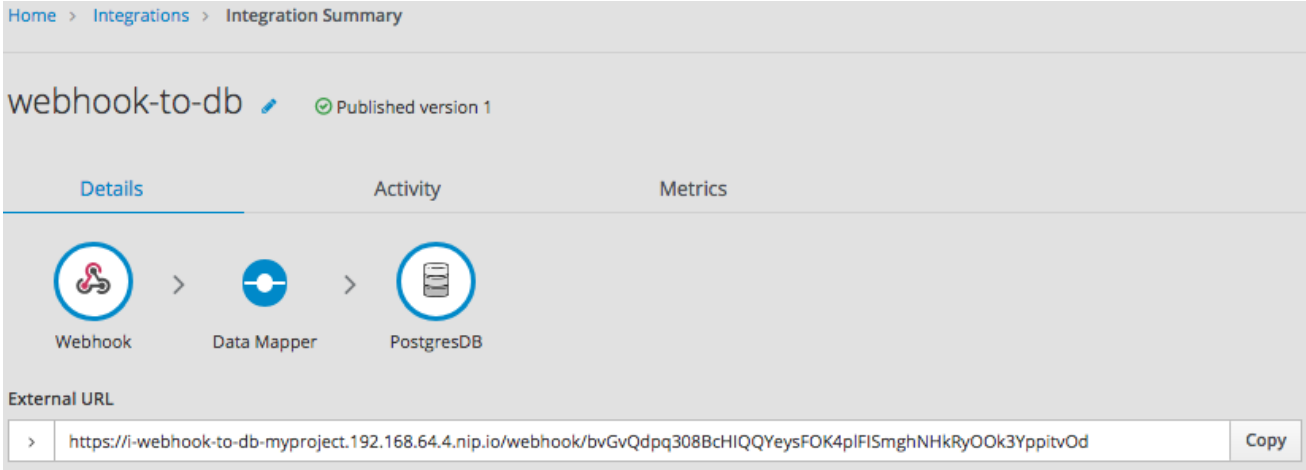



**PostgresDB** コネクションを最後のコネクションとして追加するとき、**Invoke SQL** アクションを選択し、この SQL ステートメントを指定します。

**INSERT INTO TODO (TASK) VALUES (:#TASK)**

データベースコネクションを追加した後、マッピングステップを追加します。


インテグレーションを保存し、パブリッシュします。実行中に Fuse Online が提供する外部 URL をコピーできます。




外部 URL の一部を理解するため、以下の URL の例を見てください。

**https://i-webhook-to-db-myproject.192.168.64.4.nip.io/webhook/bvGvQdpq308BcHIQQYeysFOK4plFISmghNHkRyOOk3YppitvOd**

| 値 | 説明 |
|---|---|
| **i-** | Fuse Online は常にこの値を URL の最初に挿入します。 |
| **webhook-to-db** | インテグレーションの名前。 |
| **myproject** | インテグレーションを実行している Pod が含まれる OpenShift namespace。 |
| **192.168.64.4.nip.io** | OpenShift 用に設定された DNS ドメイン。これは、Webhook を提供している Fuse Online 環境を示しています。 |
| **webhook** | 各 Webhook コネクション URL に表示されます。 |

| 値 | 説明 |
| --- | --- |
| **bvGvQdpq308BcHlQQYeysFOK4plFlSmghNHkRyOOk3YppitvOd** | Webhook コネクションをインテグレーションに追加するときに Fuse Online が提供する Webhook コネクショントークン。トークンは、URL を識別しにくくすることでセキュリティーを提供する無作為の文字列です。これにより、該当の送信者以外がリクエストを送信できないようにします。<br><br>リクエストでは、Fuse Online が提供するトークンを指定するか、独自のトークンを定義します。独自に定義する場合は、必ず簡単に推測できないものにしてください。 |

外部 URL が確認できたら、Fuse Online はインテグレーションの名前、OpenShift namespace の名前、および OpenShift DNS ドメインからホスト名を作成します。Fuse Online は使用できない文字を削除し、空白文字をハイフンに変換します。上記の外部 URL の例では、ホスト名は次のようになります。

**https://i-webhook-to-db-myproject.192.168.64.4.nip.io**

**curl** を使用して Webhook を呼び出すには、コマンドを以下のように指定します。

**curl -H 'Content-Type: application/json' -d '{"todo":"from webhook"}' https://i-webhook-to-db-myproject.192.168.64.4.nip.io/webhook/bvGvQdpq308BcHlQQYeysFOK4plFlSmghNHkRyOOk3YppitvOd**

- **-H** オプションは HTTP **Content-Type** ヘッダーを指定します。
- **-d** オプションは、デフォルトで HTTP メソッドを **POST** に設定します。

このコマンドの実行により、インテグレーションがトリガーされます。データベースの最後のコネクションは新しいタスクをタスクテーブルに挿入します。これを確認するには、たとえば **https://todo-myproject.192.168.64.4.nip.io** で Todo アプリケーションを表示し、**Update** をクリックします。**from webhook** が新しいタスクとして表示されるはずです。

### Webhook におけるクエリーパラメーターでの POST リクエストの例

この例では、前述の例と同じインテグレーションを使用します。

しかし、この例では以下のコンテンツを持つ JSON スキーマを指定して Webhook コネクションの出力データタイプを定義します。

```
{
  "type": "object",
  "definitions": {},
  "$schema": "http://json-schema.org/draft-07/schema#",
  "id": "io:syndesis:webhook",
  "properties": {
    "parameters": {
      "type": "object",
      "properties": {
        "source": {
          "type": "string"
        },
        "status": {
          "type": "string"
        }
      }
    },
    "body": {
      "type": "object",
      "properties": {
        "company": {
          "type": "string"
```

```
        },
        "email": {
          "type": "string"
        },
        "phone": {
          "type": "string"
        }
      }
    }
  }
}
```

この JSON スキーマでは以下を行います。

- **id** を **io.syndesis.webhook** に設定する必要があります。
- **parameters** セクションは HTTP クエリーパラメーターを指定する必要があります。
- **body** セクションはボディーのコンテンツを指定し、必要に応じて複雑に指定することができます。たとえば、入れ子のプロパティーやアレイを定義できます。

これは、Webhook コネクターがインテグレーションの次のステップのコンテンツを準備するために必要な情報を提供します。

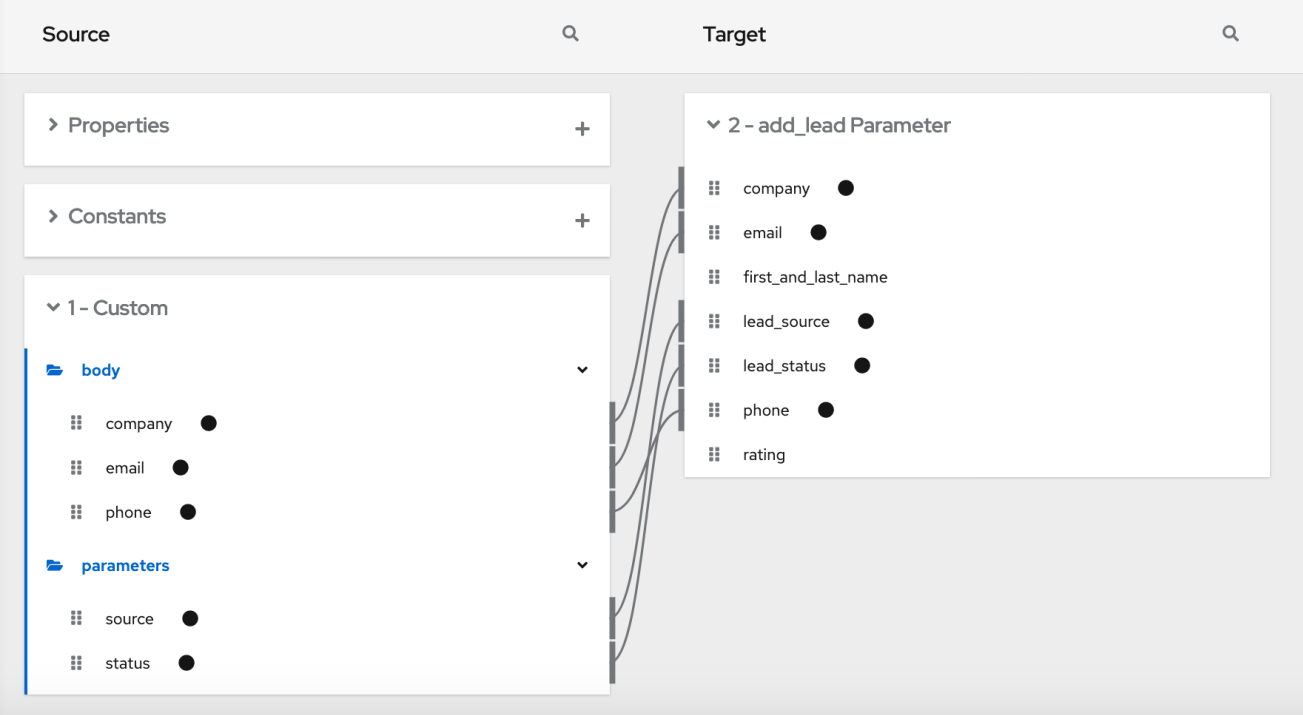
**curl** を使用して HTTP リクエストを送信するには、以下のようなコマンドを呼び出します。

**curl -H 'Content-Type: application/json' -d '{"company":"Gadgets","email":"sales@gadgets.com","phone":"+1-202-555-0152"}'https://i-webhook-params-to-db-myproject.192.168.42.235.nip.io/webhook/ZYWrhaW7dVk097vNsLX3YJ1GyxUFMFRteLpw0z4O69MW7d2Kjg?source=web&status=new**

Webhook コネクションがこのリクエストを受信すると、以下のような JSON インスタンスを作成します。

```
{
  "parameters": {
    "source": "web",
    "status": "new"
  },
  "body": {
    "company": "Gadgets",
    "email": "sales@gadgets.com",
    "phone": "+1-202-555-0152"
  }
}
```

この JSON インスタンスが以下のマッピングを有効にします。




### Webhook での GET の例

入力データを提供しない **GET** リクエストでインテグレーションをトリガーするには、Webhook コネクションの出力データシェイプを、定義 **'{}'** を持つ JSON インスタンスとして指定します。その後、クエリーパラメーターを指定しない以下の **curl** コマンドを呼び出すことができます。

**curl 'https://i-webhook-params-to-db-myproject.192.168.42.235.nip.io/webhook/ZYWrhaW7dVk097vNsLX3YJ1GyxUFMFRteLpw0z4O69MW7d2Kjg'**

前述の **POST** の例を変更して、クエリーパラメーターがありボディーはない **GET** リクエストを送信します。Webhook コネクションの出力データシェイプを、以下の定義を持つ JSON スキーマとして指定します。

```
{
  "type": "object",
  "definitions": {},
  "$schema": "http://json-schema.org/draft-07/schema#",
  "id": "io:syndesis:webhook",
  "properties": {
    "parameters": {
      "type": "object",
      "properties": {
        "source": {
          "type": "string"
        },
        "status": {
          "type": "string"
        }
      }
    }
  }
}
```

以下の **curl** コマンドは **GET** リクエストを送信します。

```
curl 'https://i-webhook-params-to-db-
myproject.192.168.42.235.nip.io/webhook/ZYWrhaW7dVk097vNsLX3YJ1GyxUFMFRteLpw0z4O69
MW7d2Kjg?source=web&status=new'`
```

# 第7章 インテグレーションデータを次のコネクションのフィールドにマッピング

ほとんどのフローでは、すでに取得または処理された不データフィールドを、フローの次のコネクションで処理できるデータフィールドにマップする必要があります。Fuse Online は、これを行うためにデータマッパーを提供します。フローで、データフィールドをマップする必要がある各場所に、データマッパーステップを追加します。

データマッパーステップを追加すると、Syndesis はデータマッパーのキャンバスに以下を表示します。* 以前の接続で提供されるソースデータフィールド * 以降の接続で必要とされるターゲットデータフィールド

**重要**

データマッパーでは、以前のインテグレーションステップで提供できるソースフィールドの最大セットが表示されます。ただし、すべてのコネクションによって、表示された各ソースフィールドにデータが提供されるわけではありません。たとえば、サードパーティーアプリケーションへの変更によって、特定のフィールドにデータが提供されなくなる可能性があります。インテグレーションの作成時に、データマッピングが想定どおりに機能していないことが判明した場合は、マップするソースフィールドに予想されるデータが含まれていることを確認してください。

データフィールドをマッピングする詳細については、以下を参照してください。

- 「ステップでのマッピングの表示」
- 「データマッピングが必要な場所を特定」
- 「マップするデータフィールドの検索」
- 「データ型とコレクションについて」
- 「マッピングの種類について」
- 「1 つのソースフィールドを 1 つのターゲットフィールドへマッピング」
- 「欠落しているソースまたはターゲットの値を指定する」
- 「フィールドを組み合わせまたは分割する場合の不足または不必要なデータの例」
- 「複数のソースフィールドを 1 つのターゲットフィールドに組み合わせる」
- 「1 つのソースフィールドを複数のターゲットフィールドに分割」
- 「データマッパーを使用したコレクションの処理」
- 「コレクションと非コレクション間のマッピング」
- 「ソースまたはターゲットデータの変換」
- 「1 つのターゲットフィールドにマップする前の複数のソース値の変換」
    - 「利用可能な変換の説明」
    - 「条件のマッピングへの適用」

- 「データマッピングのトラブルシューティング」

## 7.1. ステップでのマッピングの表示

データマッパーステップを追加または編集する間に、このステップにすでに定義されたマッピングを表示できます。このオプションにより、正しいマッピングがあるかどうかを確認できます。

**前提条件**

- インテグレーションを作成または更新することになります。
- データマッパーステップを追加することになります。よって、データマッパーは表示されます。

**手順**

1. マッピングのテーブルビューを表示するには、 をクリックします。
2. マッピングのテーブルビューを解除し、ソースおよびターゲットフィールドを再表示するには、 をクリックします。

## 7.2. データマッピングが必要な場所を特定

Fuse Online は、フローにデータマッピングが必要な場所を示す警告アイコンを表示します。

**前提条件**

- フローを作成または編集することになります。
- フローには必要なコネクションがすべて含まれている必要があります。

**手順**

1. フロービジュアライゼーションで、 アイコンを見つけます。
2. アイコンをクリックして **Data Type Mismatch** 通知を確認します。
3. メッセージで **Add a data mapping step**をクリックし、データマッパーを表示します。
4. 必要なデータマッパーステップを追加すると、次の理由により、 アイコンが引き続き表示されることがあります (編集プロセス中にいつでも表示される可能性があります)。
   - ソースステップの 1 つが出力を変更しました
   - ターゲットステップの入力がマッパーの出力と互換性がありません
   - ソースステップの 1 つがありません
   - 対象のステップがありません
     このシナリオでは、この警告は、上で追加したデータマッパーステップを編集する必要があることを示しています。

## 7.3. マップするデータフィールドの検索

比較的ステップの数が少ないフローでは、データフィールドのマッピングは簡単で感覚的に行うことができます。より複雑なフローや、大量のデータフィールドを処理するフローでは、データマッパーの処理方法に関する背景が分かればソースからターゲットへのマッピングがより簡単になります。

データマッパーはデータフィールドの列を 2 つ表示します。

- **Source** は、フローのこれまでのステップすべてで取得または処理されたデータフィールドの一覧です。
- **Target** は、フローの次のコネクションが予期して処理できるデータフィールドの一覧です。

マップするデータフィールドを素早く検索するには、以下のいずれかを行います。

- 検索します。
  **Sources** パネルと **Target** パネルには、それぞれに上部に検索フィールドがあります。検索フィールドが表示されていない場合は、**Sources** または **Target** パネルの右上にある  をクリックします。

  マップするフィールドの名前を入力します。**Sources** 検索フィールドに、ソースフィールドの名前を入力します。**Target** 検索フィールドに、マップ先のフィールドの名前を入力します。
-  および  オプションを使用して、表示されるフィールドを絞り込みます。
- フォルダーを展開したり折りたたんだりして、表示するフィールドを制限します。
  特定のステップで利用可能なデータフィールドを表示するには、そのステップのフォルダーを展開します。

  手順をフローに追加すると、ステップに番号が割り当てられ、Fuse Online がステップを処理する順番を示します。データマッパーステップを追加する場合、ステップ番号は **Sources** パネルと **Target** パネルのフォルダーラベルに表示されます。
- 各フィールドのデータタイプを表示するには、  をクリックし、各フィールドのラベルのデータタイプを表示 (または非表示) します。フォルダーラベルには、ステップによって出力されるデータタイプの名前も表示されます。フォルダーラベルには、ステップによって出力されるデータタイプの名前も表示されます。Twitter、Salesforce、SQL などのアプリケーションへのコネクションでは、独自のデータタイプを定義します。Amazon S3、AMQ、AMQP、Dropbox、FTP/SFTP などのアプリケーションに接続するには、コネクションをフローに追加するときにコネクションの入力および出力タイプを定義し、コネクションが実行するアクションを選択します。データタイプを指定するときに、タイプに名前を付けます。指定したタイプ名は、データマッパーのフォルダー名として表示されます。データタイプの宣言時に説明を指定した場合は、マッパーの step フォルダーにマウスオーバーするとタイプの説明が表示されます。

## 7.4. データ型とコレクションについて

データマッパーはソースフィールドとターゲットフィールドを表示し、必要なフィールド間のマッピングを定義します。

データマッパーでは、フィールドは以下のいずれかになります。

- 単一の値を格納する **プリミティブ** タイプ。プリミティブタイプの例には、**boolean**、**char**、**byte**、**short**、**int**、**long**、**float**、および **double** があります。プリミティブタイプは単一のフィールドであるため拡張できません。

- 異なるタイプの複数のフィールドで設定される **コンプレックス** タイプ。コンプレックスタイプの子フィールドを設計時に定義します。データマッパーではコンプレックスタイプを拡張して子フィールドを表示できます。

各タイプのフィールド (プリミティブおよびコンプレックス) をコレクションとすることもできます。コレクションは、複数の値を持つことができる単一のフィールドです。コレクションのアイテム数はランタイム時に決定されます。設計時、データマッパーではコレクションは  で示されます。コレクションがデータマッパーインターフェイスで拡張可能かどうかは、タイプにより決定されます。コレクションがプリミティブタイプの場合、拡張できません。コレクションがコンプレックスタイプの場合、データマッパーを展開し、コレクションの子フィールドを表示できます。各フィールドをマップ元またはマップ先とすることができます。

以下に例を示します。

- **id** はプリミティブタイプのフィールド (**int**) です。実行時に、社員は **ID** を 1 つだけ持てます。たとえば、**ID=823** のみを持てます。そのため、**ID** はコレクションではないプリミティブタイプです。データマッパーでは、**ID** は拡張できません。

- **email** はプリミティブタイプのフィールド (文字列) です。実行時に、社員は複数の **email** の値を持つことができます。たとえば、**email<0>=aslan@home.com** および **email<1>=aslan@business.com** を持つことができます。そのため、**email** はコレクションでもあるプリミティブタイプです。データマッパーは、  を使用して、**email** フィールドがコレクションであることを示しますが、**email** はプリミティブタイプであるため (子フィールドがない)、拡張できません。

- **employee** は、**ID** や **email** を含む複数の子フィールドを持つ複雑なオブジェクトフィールドです。会社には多くの従業員がいるため、起動時に **employee** はコレクションでもあります。設計時に、データマッパーは  を使用して **employee** がコレクションであることを示します。**employee** フィールドは、子フィールドが含まれるコンプレックスタイプであるため、拡張可能です。

## 7.5. マッピングの種類について

データマッパーは、次の一般的なタイプのマッピングをサポートしています。

- **1対1** – 1 つのソースフィールドを 1 つのターゲットフィールドにマップします。

- **多対1** – 複数のソースフィールドを 1 つのターゲットフィールドにマップします。データマッパーがマップされたソースフィールド間のターゲットフィールドに挿入する区切り文字を指定します。デフォルトの区切り文字は空白文字です。

- **1対多** – 1 つのソースフィールドを複数のターゲットフィールドにマップします。ソースフィールドにある区切り文字を指定します。AtlasMap は、区切られた各値を、選択したターゲットフィールドにマップします。

- **For each** – 1 つのソースコレクションフィールドを 1 つのターゲットコレクションフィールドに繰り返しマップします。

## 7.6. 1 つのソースフィールドを 1 つのターゲットフィールドへマッピング

1つのソースフィールドを1つのターゲットフィールドにマップするのがデフォルトのマッピング動作です。たとえば、**Name** フィールドを **CustomerName** フィールドにマップします。

**手順**

1. **Sources** パネルで以下を行います。
   a. 必要な場合はステップを展開し、提供されるデータフィールドを表示します。
      ソースフィールドが多数ある場合は、 をクリックして対象のフィールドを検索し、検索フィールドにデータフィールドの名前を入力します。
   b. マップ元となるデータフィールドをクリックし、 をクリックします。**Mapping Details** パネルが表示されます。
2. **Target** パネルで、マップ先となるデータフィールドを見つけ、 をクリックします。
   データマッパーは、先ほど選択した2つのフィールドを接続する行を表示します。
3. 必要に応じて、データマッピングの結果をプレビューします。このオプションは、マッピングに変換を追加する場合や、マッピングにタイプ変換が必要な場合に便利です。
   a. データマッパーの右上で をクリックし、ソースフィールドにテキスト入力フィールドを表示し、ターゲットフィールドに読み取り専用結果フィールドを表示します。
   b. ソースフィールドのデータ入力フィールドに、サンプル入力値を入力します。マッピングの結果は、ターゲットフィールドの読み取り専用フィールドに表示されます。
   c. 任意で、変換の結果を確認するには、**Mapping Details** パネルに変換を追加します。
   d. プレビューフィールドを非表示にするには、 を再度クリックします。
4. 任意で、マッピングが定義されていることを確認するには、 をクリックし、定義されたマッピングを表示します。
   このビューでデータマッピングの結果をプレビューすることもできます。プレビューフィールドが表示されない場合は、 をクリックします。前の手順で説明したデータを入力します。
   定義されたマッピングの表では、プレビューフィールドは選択したマッピングでのみ表示されます。
   a. 別のマッピングのプレビューフィールドを表示するには、そのマッピングを表示します。
   b. を再度クリックし、データフィールドパネルを表示します。
5. 右上の **Done** をクリックし、データマッパーステップをインテグレーションに追加します。

### トラブルシューティングのヒント

データマッパーでは、以前のインテグレーションステップで提供できるソースフィールドの最大セットが表示されます。ただし、すべてのコネクションによって、表示された各ソースフィールドにデータが提供されるわけではありません。たとえば、サードパーティーアプリケーションへの変更によって、特定のフィールドにデータが提供されなくなる可能性があります。インテグレーションの作成時に、データマッピングが想定どおりに機能していないことが判明した場合は、マップするソースフィールドに予想されるデータが含まれていることを確認してください。

## 7.7. 欠落しているソースまたはターゲットの値を指定する

フィールドをマップすると、ソースデータシェイプがターゲットデータシェイプに必要な値を提供しない場合や、その逆の場合があります。プロパティーまたは定数を定義することによって、欠落している値を指定することができます。

たとえば、ターゲットデータシェイプが、値が HORIZONTAL または VERTICAL でなければならない **Layout** フィールドを定義するとします。ソースデータシェイプはこのフィールドを提供しません。定数を作成して、**Layout** ターゲットフィールドにマップできます。

### 前提条件

- データマッパーで、**Mapping Details** パネルが開いています。

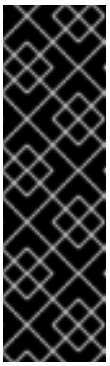
### 手順

定数を定義するには:

1. **Source** または **Target** パネルの上部で、**Constants** の右側にある **Add** (+) をクリックします。**Create Constant** ダイアログが開きます。
2. 定数の値を入力します。
3. データ型を選択します。
4. **Save** をクリックして新しいフィールドを作成します。

プロパティーを定義するには:

1. **Source** または **Target** パネルの上部で、**Properties** の右側にある **Add** (+) をクリックします。**Create Property** ダイアログが開きます。
2. プロパティー名を入力します。
3. データ型を選択します。
4. **Scope** プルダウンメニューから、オプションの 1 つを選択して、プロパティーのスコープを定義します。
   - **Current message header** - 前のステップから Data Mapper ステップに渡されたメッセージヘッダー。
   - **Camel Exchange Property** - Camel 固有のプロパティー用。
   - **Result** - 前のステップのメッセージヘッダー。



**重要**

スコープオプションはテクノロジープレビュー機能のみです。テクノロジープレビュー機能は、Red Hat 製品のサービスレベルアグリーメント (SLA) の対象外であり、機能的に完全ではないことがあります。Red Hat は、実稼働環境での使用を推奨しません。テクノロジープレビューの機能は、最新の製品機能をいち早く提供して、開発段階で機能をテストし、フィードバックを提出できるようにすることを目的としています。Red Hat のテクノロジープレビュー機能のサポート範囲に関する詳細は、テクノロジープレビュー機能のサポート範囲 を参照してください。

5. **Save** をクリックして新しいフィールドを作成します。

## 7.8. フィールドを組み合わせまたは分割する場合の不足または不必要なデータの例

データマッピングでは、ソースまたはターゲットフィールドに複合データが含まれている場合に、不足しているデータや不必要なデータを特定する必要がある可能性があります。たとえば、以下のような形式の **long_address** フィールドがあるとします。

number street apartment city state zip zip+4 country

**number**、**street**、**city**、**state**、および **zip** のソースフィールドを **long_address** ターゲットフィールドに組み合わせることにします。これを行うには、**long_address** をソースフィールドとして選択し、ターゲットフィールドを選択します。次に、ソースフィールドの不要なパーツがある場所に、パディングフィールドを追加します。この例では、不要なパーツは **apartment**、**zip+4**、および **country** になります。

不要なパーツを特定するには、これらのパーツの順番を知っておく必要があります。順番は、複合フィールドにある内容の各パーツのインデックスを示します。たとえば、**long_address** フィールドには順番が付けられたパーツが 8 つあります。インデックスの各パーツの順番は次のようになり、1 で始まります。

| | |
|---|---|
| 1 | **number** |
| 2 | **street** |
| 3 | **apartment** |
| 4 | **city** |
| 5 | **state** |
| 6 | **zip** |
| 7 | **zip+4** |
| 8 | **country** |

データマッパーで **apartment**、**zip+4**、および **country** が不足していると識別するには、インデックス 3、7、および 8 にパディングフィールドを追加します。複数のソースフィールドを 1 つのターゲットフィールドに組み合わせる を参照してください。

**number**、**street**、**city**、**state**、および **zip** のソースフィールドを **long_address** ターゲットフィールドに組み合わせることにします。さらに、**apartment**、**zip+4**、および **country** の内容を提供するソースフィールドがないとします。データマッパーで、これらのフィールドが不足していると特定する必要があります。ここでも、インデックス 3、7、および 8 にパディングフィールドを追加します。1 つのソースフィールドを複数のターゲットフィールドに分割 を参照してください。

## 7.9. 複数のソースフィールドを 1 つのターゲットフィールドに組み合わせる

データマッパーステップでは、複数のソースフィールドを 1 つの複合ターゲットフィールドに組み合わせることができます。たとえば、**FirstName** および **LastName** フィールドを **CustomerName** にマップできます。

### 前提条件

ターゲットフィールドでは、この複合フィールドの各パーツのコンテンツタイプ、コンテンツの各パーツの順番およびインデックス、および空白やコンマなどのパーツの区切り文字を知っている必要があります。フィールドを組み合わせまたは分割する場合の不足または不必要なデータの例 を参照してください。

### 手順

1. **Target** パネルで、複数のソースフィールドをマップするフィールドをクリックし、  をクリックします。**Mapping Details** パネルが表示されます。
2. **Mapping Details** パネルの **Source** ドロップダウンリストから、マップ先となる 1 つ以上のデータフィールドを選択します。
   作業が完了したら、各ソースフィールドからターゲットフィールドへの線が表示されるはずです。

   **Mapping Details** パネルの **Sources** の上に、デフォルトの多重変換である **Concatenate** が表示されます。これは、マッピングの実行により、**Concatenate** 変換が指定のソースフィールドの値に適用され、連結された値を指定のターゲットフィールドにマップすることを意味します。
3. **Mapping Details** パネルで、以下のようにマッピングを設定します。
   a. **Sources** の **Delimiter** フィールドで、異なるソースフィールドからのコンテンツの間に挿入される文字を指定または選択します。デフォルトは空白文字です。
   b. 必要であれば、ソースフィールドエントリーごとに  をクリックすると、ソースフィールドがターゲットフィールドにマップされる前に変換をソースフィールド値に適用することができます。
   c. **Sources** で、選択したソースフィールドのエントリーの順番を確認します。エントリーの順番は、複合ターゲットフィールドの対応するコンテンツと同じである必要があります。エントリーの順番が正しくない場合は、フィールドエントリーのインデックス番号を変更し、同じ順番になるようにします。

      ソースフィールドを複合ターゲットフィールドの各パーツにマップした場合は、次のステップを省略します。
   d. ソースフィールドエントリーのインデックスが、ターゲットフィールドの対応データと同じでない場合、インデックスが同じになるように編集します。各ソースフィールドエントリーのインデックスは、ターゲットフィールドの対応データと同じである必要があります。データマッパーは、不足しているデータを示すために、必要に応じてパディングフィールドを自動的に追加します。
      誤って余分なパディングフィールドを作成した場合は、削除する余分なパディングフィールドごとに、  をクリックします。
   e. 任意で、**Targets** 下の  をクリックして内容をターゲットフィールドにマップし、ソースまたはターゲットデータの変換 の説明どおりに変換を適用します。

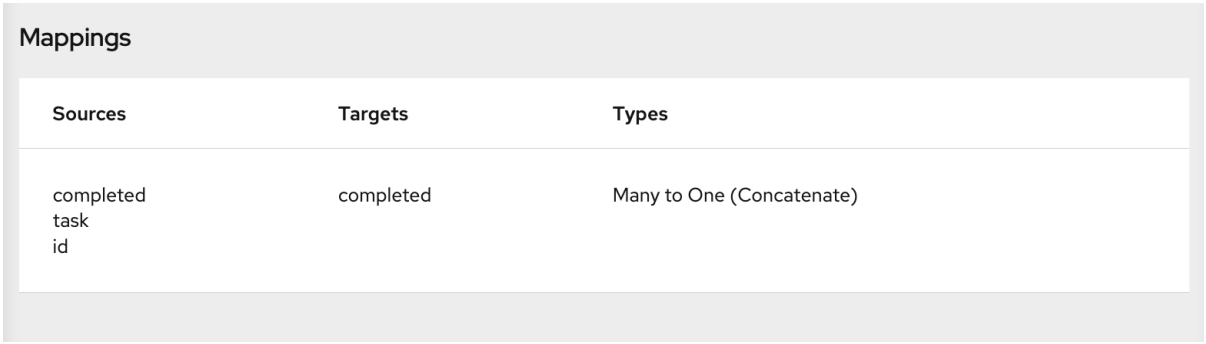
4. 必要に応じて、データマッピングの結果をプレビューします。

   a. をクリックし、現在選択されているマッピングの各ソースフィールドにテキスト入力フィールドを表示し、現在選択されているマッピングのターゲットフィールドに読み取り専用結果フィールドを表示します。

   b. ソースデータ入力フィールドに、サンプル値を入力します。
   ソースフィールドの順序を変更したり、マッピングに変換を追加すると、ターゲットフィールドの結果フィールドにこれが反映されます。データマッパーがエラーを検出すると、**Mapping Details** パネルの上部に情報メッセージが表示されます。

   c. を再度クリックして、プレビューフィールドを非表示にします。
   プレビューフィールドを再表示すると、入力したデータはデータマッパーが終了するまでそのまま存在します。

5. マッピングが適切に定義されていることを確認するには、 をクリックし、このステップで定義されたマッピングを表示 (表形式) します。複数のソースフィールドの値を 1 つのターゲットフィールドに組み合わせるマッピングは次のようになります。




   このビューでは、マッピングの結果をプレビューすることもできます。 をクリックし、前のステップの説明どおりにテキストを入力します。プレビューフィールドは選択したマッピングのみに表示されます。表の別のマッピングをクリックして、そのプレビューフィールドを表示します。

### その他のリソース

パディングフィールドの追加例:1 つのソースフィールドを複数のターゲットフィールドに分割

この例は 1 対多のマッピング例ではありませんが、原理は同じです。

## 7.10. 1 つのソースフィールドを複数のターゲットフィールドに分割

データマッパーステップでは、複合ソースフィールドを複数のターゲットフィールドに分割できます。たとえば、**Name** フィールドを **FirstName** および **LastName** フィールドにマップします。

### 前提条件

ソースフィールドでは、この複合フィールドの各パーツのコンテンツタイプ、コンテンツの各パーツの順番およびインデックス、および空白やコンマなどのパーツの区切り文字を知っている必要があります。フィールドを組み合わせまたは分割する場合の不足または不必要なデータの例 を参照してください。

### 手順

1. **Sources** パネルで、分割するコンテンツのあるフィールドをクリックし、  をクリックします。

2. **Mapping Details** パネルで、**Target** ドロップダウンリストから、マップ先のデータフィールドを選択します。
ターゲットフィールドの選択を終了すると、ソースフィールドから選択した各ターゲットフィールドへの線が表示されるはずです。

   **Mapping Details** の上部に **Split** が表示されます。これは、マッピングの実行によりソースフィールドの値が分割され、複数のターゲットフィールドにマップされることを意味します。

   **Targets** に選択した各ターゲットフィールドのエントリーがあります。

3. **Mapping Details** パネルで、以下のようにマッピングを設定します。

   a. **Sources** の **Delimiter** フィールドで、ソースフィールドの値を区切る場所を示すソースフィールドの文字を指定または選択します。デフォルトは空白文字です。

   b. 必要に応じて、  をクリックし、ソースフィールドがターゲットフィールドにマップされる前に変換をソースフィールド値に適用します。

   c. **Targets** 下で、選択したターゲットフィールドのエントリーの順番を確認します。エントリーの順番は、複合ソースフィールドの対応するコンテンツと同じである必要があります。ソースフィールドのコンテンツのパーツ 1 つ以上に対してターゲットフィールドを指定しなくても、問題ありません。
   エントリーの順番が正しくない場合は、フィールドエントリーのインデックス番号を変更し、同じ順番になるようにします。

      複合ソースフィールドの各パーツをターゲットフィールドにマップした場合は、次のステップを省略します。

   d. ソースフィールドに、不必要なデータが含まれる場合は、**Mapping Details** パネルで、ソースフィールドの対応するデータと同じインデックスを持たない各ターゲットフィールドのインデックスを編集します。各ターゲットフィールドエントリーのインデックスは、ソースフィールドの対応するデータのインデックスと同じである必要があります。データマッパーは、不要なデータを示すため、必要に応じてパディングフィールドを自動的に追加します。
   この手順の最後にある例を参照してください。

   e. 任意で、  をクリックして内容をターゲットフィールドにマップし、ソースまたはターゲットデータの変換 の説明どおりに変換を適用します。

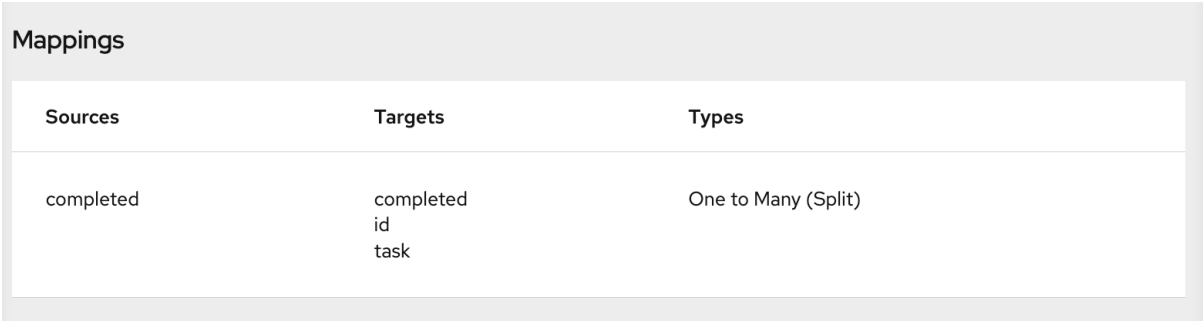
4. 必要に応じて、データマッピングの結果をプレビューします。

   a.  をクリックし、ソースフィールドにテキスト入力フィールドを表示し、各ターゲットフィールドに読み取り専用結果フィールドを表示します。

   b. ソースフィールドのデータ入力フィールドに、サンプル値を入力します。フィールドのパーツの間には必ず区切り文字を入力してください。マッピングの結果は、ターゲットフィールドの読み取り専用フィールドに表示されます。
   ターゲットフィールドの順序を変更したり、ターゲットフィールドに変換を追加すると、ターゲットフィールドの結果フィールドにこれが反映されます。データマッパーがエラーを検出すると、**Mapping Details** パネルの上部に情報メッセージが表示されます。

c. を再度クリックして、プレビューフィールドを非表示にします。
プレビューフィールドを再表示すると、入力したデータはデータマッパーが終了するまでそのまま存在します。

5. マッピングが適切に定義されていることを確認するには、 をクリックし、このステップで定義されたマッピングを表示します。ソースフィールドの値を複数のターゲットフィールドに分割するマッピングは次のようになります。




このビューでは、マッピングの結果をプレビューすることもできます。 をクリックし、前のステップの説明どおりにテキストを入力します。プレビューフィールドは選択したマッピングのみに表示されます。表の別のマッピングをクリックして、そのプレビューフィールドを表示します。

### 1つのフィールドを複数のフィールドに分割する例

ソースデータに1つのアドレスフィールドが含まれ、そのフィールドでは以下の例のようにコンマを使用してコンテンツのパーツを区切るとします。

> 77 Hill Street, Brooklyn, New York, United States, 12345, 6789

住所フィールドでは、コンテンツのパーツにこれらのインデックスがあります。

| コンテンツ | インデックス |
|---|---|
| 番地およびストリート名 | 1 |
| 市 | 2 |
| 州 | 3 |
| 国名 | 4 |
| 郵便番号 | 5 |
| zip+4 コード | 6 |

ターゲットデータに住所のフィールドが 4 つあるとします。

> number-and-street
> city
> state
> zip

マッピングを定義するには以下を行います。

- ソースフィールドを選択し、 をクリックします。
- Mapping Details パネルの Sources セクションで、区切り文字を選択します。この例ではコンマが使用されます。
- ４つのターゲットフィールドを選択します。

この作業を行った後、Targets の Mapping Details パネルに、選択した各ターゲットフィールドのエントリーが表示されます。例を以下に示します。


をクリックします。

データマッパーで表示される順にターゲットエントリーが表示されます。これはアルファベット順になります。この順番がソースフィールドの順番を反映するように変更する必要があります。この例では、ソースフィールドの **city** コンテンツの前に **number-and-street** コンテンツが含まれます。ターゲットエントリーの順序を修正するには、**city** インデックスフィールドを **2** に変更します。結果は以下のようになります。


をクリックします。

ターゲットフィールドエントリーでは、インデックス番号はこのターゲットフィールドにマップされるソースフィールドのパーツを示します。インデックス値の１つを変更してターゲットフィールドの値を修正する必要があります。以下のターゲットフィールドを見てみましょう。

- **number-and-street**: ソースフィールドの番地およびストリート名コンテンツのインデックス番号は１です。インデックスが１のソースを **number-and-street** ターゲットフィールドにマップするのは適切です。このターゲットエントリーに変更を加える必要はありません。
- **city**: のソースフィールドの市コンテンツのインデックス番号は２です。このターゲットエントリーも適切です。
- **state**: ソースフィールドの州コンテンツのインデックス番号は３です。このターゲットエントリーも適切です。

- **zip**: ソースフィールドの郵便番号コンテンツのインデックス番号は 5 です。ターゲットフィールドエントリーのインデックス番号は 4 で、正しくありません。これを変更しないと、実行中にソースフィールドの国パーツが **zip** ターゲットフィールドにマップされます。インデックス番号を 5 に変更する必要があります。インデックス 5 のソースコンテンツを **zip** ターゲットフィールドにマップするよう、データマッパーに指示します。インデックスの変更後、インデックス番号が 4 のパディングフィールドがデータマッパーによって追加されます。結果は以下のようになります。

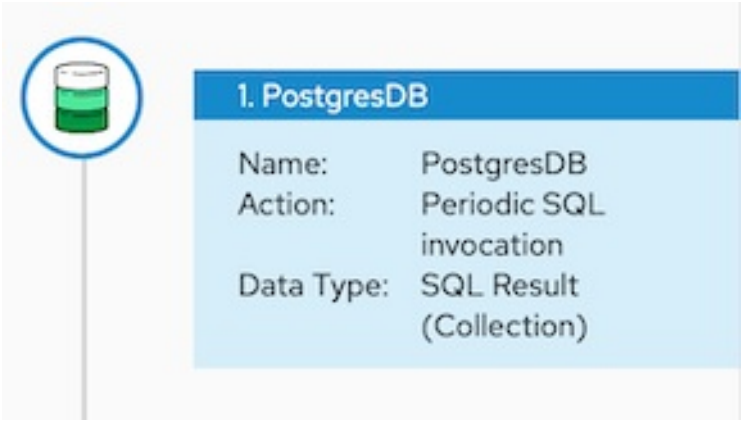

をクリックします。

これでこのマッピングが完了します。ソースフィールドにはインデックス番号が 6 のコンテンツ (zip+4) がありますが、このデータはターゲットに必要なく、何もする必要はありません。

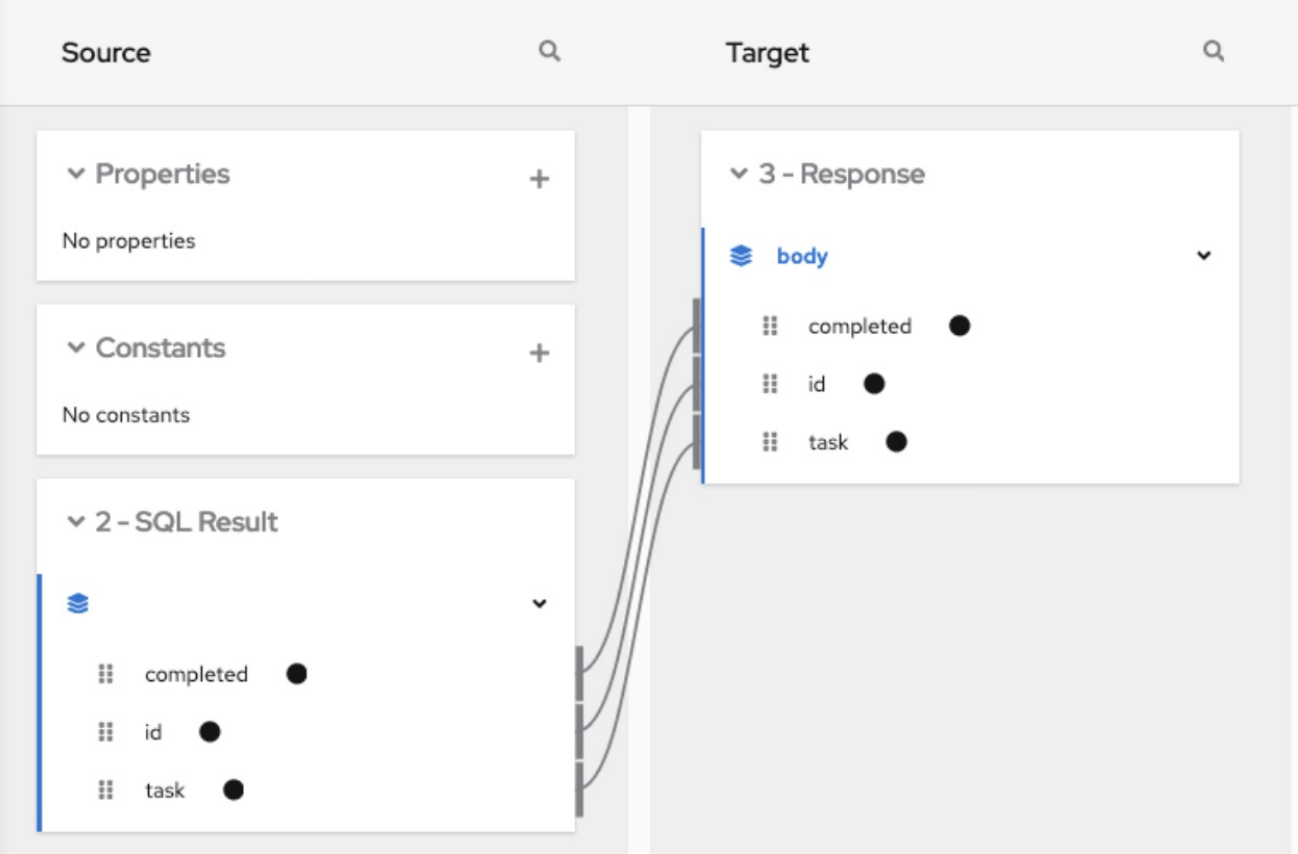
## 7.11. データマッパーを使用したコレクションの処理

フローでは、ステップがコレクションを出力し、フローの後続のコネクションはコレクションを入力として想定する場合、データマッパーを使用してフローがどのようにコレクションを処理するかを指定できます。

ステップがコレクションを出力すると、フロービジュアライゼーションはステップの詳細で **Collection** を表示します。以下に例を示します。




データマッパーステップを、コレクションを提供するステップの後およびマッピングを必要とするステップの前に追加します。フローでこのデータマッパーステップが必要な場所は、フローの他のステップによって異なります。以下のイメージは、ソースコレクションフィールドからターゲットコレクションフィールドへのマッピングを示しています。




ソースおよびターゲットパネルで、データマッパーは を表示し、コレクションを示します。

コレクションがコンプレックスタイプの場合、データマッパにコレクションの子フィールドが表示されます。各フィールドをマップ元またはマップ先とすることができます。

ソースフィールドが複数のコレクションで入れ子になっている場合、以下の条件の 1 つを満たすターゲットフィールドにマップできます。

- ターゲットフィールドは、ソースフィールドと同じ数のコレクションで入れ子になっています。たとえば、以下のマッピングが許可されます。
  - /A<>/B<>/C → /D<>/E<>/F
  - /A<>/B<>/C → /G<>/H/I<>/J
- ターゲットフィールドは 1 つのコレクションでのみ入れ子になっています。たとえば、以下のマッピングが許可されます。
  /A<>/B<>/C → /K<>/L

  この場合、データマッパーで深さ優先アルゴリズムが使用され、ソースのすべての値が反復処理されます。データマッパーによって、ソース値は発生順に単一のターゲットコレクションに配置されます。

以下のマッピングは許可されません。

/A<>/B<>/C cannot-map-to /M<>/N/O<>/P<>/Q

Fuse Online でフローが実行されると、ソースコレクション要素が繰り返し処理され、ターゲットコレクション要素が入力されます。1 つ以上のソースコレクションフィールドをターゲットコレクションまたはターゲットコレクションフィールドにマップする場合、ターゲットコレクション要素にはマップされたフィールドのみの値が含まれます。

ソースコレクションまたはソースコレクションのフィールドをコレクションではないターゲットフィールドにマップする場合、Fuse Online がフローを実行するときにソースコレクションの最後の要素のみから値を割り当てます。コレクションの他の要素は、そのマッピングステップで無視されます。しかし、後続のマッピングステップはソースコレクションのすべての要素にアクセスできます。

コネクションが JSON または Java ドキュメントに定義されたコレクションを返すと、データマッパーは通常コレクションとしてソースドキュメントを処理できます。

## 7.12. コレクションと非コレクション間のマッピング

データマッパーの **Source** および **Target** パネルには以下があります。

-  はコレクションを示します。コレクションにプリミティブタイプが含まれる場合は、コレクションをマップ元またはマップ先として直接マッピングできます。コレクションに 2 つ以上の異なるタイプが含まれる場合、データマッパーはコレクションの子フィールドを表示し、コレクションのフィールドをマップ元またはマップ先としてマッピングできます。

-  は、コンプレックスタイプの拡張可能なコンテナーを示します。コンプレックスタイプには、異なるタイプの複数のフィールドが含まれます。コンプレックスタイプのフィールドは、アレイなどのコレクションのタイプにすることができます。コンプレックスタイプのコンテナー自体をマップすることはできません。コンプレックスタイプにあるフィールドのみをマップできます。

**(COMPLEX)**、**STRING**、**INTEGER** などのデータタイプの表示を切り替えるには、  をクリックします。

### コレクションから非コレクション (多対 1) へのマッピング

コレクションフィールドから非コレクションフィールドにマップする場合、多対 1 のマッピングがデータマッパーによって認識されます。デフォルトの動作では、データマッパーによって **Concatenate** 変換がソースコレクションまたはソースコレクションフィールドに適用されます。デフォルトの区切り文字は空白文字です。たとえば、以下のソースコレクションについて考えてみましょう。

- 最初の要素では、**city** フィールドの値は **Boston** です。
- 2 つ目の要素では、**city** フィールドの値は **Paris** です。
- 3 つ目の要素では、**city** フィールドの値は **Tokyo** です。

実行中、ターゲットフィールドに以下が入力されます。

**Boston Paris Tokyo**

別の変換を適用すると、このデフォルトの動作を変更できます。たとえば、選択する要素からのみマップするには、**Item At** 変換をソースに適用し、インデックスを指定します。ソースコレクションの最初の要素にある値をマップするには、インデックスに **0** を指定します。

マップしないフィールドがソースコレクションに含まれている場合、これらのフィールドはフローにある後続のステップでも使用できます。

### 非コレクションからコレクション (1 対多) へのマッピング

非コレクションソースフィールドからターゲットコレクションやコレクション要素のターゲットフィールドにマップする場合、1 対多のマッピングがデータマッパーによって認識されます。デフォルトの動作では、空白を区切り文字として使用し、ソース値を複数の値に分割して **Split** 変換がデータマッパー

によって適用されます。実行中、それぞれの分割値がターゲットコレクションの独自の要素に挿入されます。たとえば、ソースフィールドが 4 つの値に分割される場合、ターゲットコレクションには 4 つの要素が存在します。

たとえば、以下が含まれるコレクションでない **cities** ソースフィールドについて考えてみましょう。

**Boston Paris Tokyo**

このソースフィールドをターゲットコレクションまたはコレクションのターゲットフィールドにマップできます。実行中、**cities** フィールドの値は区切り文字 (空白文字) で分割されます。結果として、3 つの要素が含まれる 1 つのコレクションが作成されます。最初の要素では、**city** フィールドの値は **Boston** です。2 つ目の要素では、**city** フィールドの値は **Paris** です。3 つ目の要素では、**city** フィールドの値は **Tokyo** です。

**その他のリソース**

- 1 つのターゲットフィールドにマップする前の複数のソース値の変換
- 複数のソースフィールドを 1 つのターゲットフィールドに組み合わせる
- 1 つのソースフィールドを複数のターゲットフィールドに分割

## 7.13. ソースまたはターゲットデータの変換

マッピングの定義後、データマッパーでマッピングのフィールドを変換できます。データフィールドの変換は、データの格納方法を定義します。たとえば、Capitalize 変換を指定して、データ値の最初の文字が大文字になるようにすることができます。

同じマッピング内の異なるフィールドに異なる変換を適用できます。1 つのソースフィールドを 1 つのターゲットフィールドにマップする 1 対 1 のマッピングでは、変換をソースフィールドまたはターゲットフィールドのどちらに適用するかは問題ではありません。

1 対多または多対 1 のマッピングでは、変換を指定するときに必要なターゲットフィールドの値を考慮してください。たとえば、番号、番地、都市、州のソースフィールドを 1 つのターゲット住所フィールドに結合する多対 1 マッピングを考えてみます。ターゲットアドレスフィールドの文字列をすべて大文字にする場合は、ターゲットアドレスフィールドを選択し、大文字変換を適用します。州のみを大文字にする必要がある場合は、ソース州フィールドを選択し、大文字変換を適用します。

ソースフィールド変換は前処理を実行し、ターゲットフィールド変換は後処理を実行すると考えることができます。

**注記:** マッピングに条件を追加する場合は、条件のマッピングへの適用 の説明どおりに、変換を条件式内に追加する必要があります。

**手順**

1. フィールドをマップします。これは、1 対 1 のマッピング、組み合わせのマッピング、または分割のマッピングになります。
2. Mapping Details パネルの Sources または Targets 下にある変換するフィールドのボックスで、 をクリックします。このオプションは、利用可能な変換のドロップダウンリストを表示します。
3. データマッパーが実行する変換を選択します。

4. 変換に入力パラメーターが必要な場合は、該当する入力フィールドに指定します。

5. 別の変換を追加するには、 を再度クリックします。

**その他のリソース**

- 利用可能な変換の説明。
- 1つのターゲットフィールドにマップする前の複数のソース値の変換

### 7.13.1. 1つのターゲットフィールドにマップする前の複数のソース値の変換

複数のソースフィールドや、コレクションなどの複数の値が含まれるソースフィールドの値に適用できる変換があります。変換の結果はデータマッパーによってターゲットフィールドに挿入されます。以下の表は、これらの多重変換について説明しています。

| 多重変換 | 説明 |
|---|---|
| Add | ソースの数値を追加し、その合計をターゲットフィールドに挿入します。選択したソースフィールドまたは選択したコレクションの値は数値である必要があります。 |
| Average | ソースの数値の平均を算出し、その算出値をターゲットフィールドに挿入します。選択したソースフィールドまたは選択したコレクションの値は数値である必要があります。 |
| Concatenate | ソース値を結合し、その結合値をターゲットフィールドに挿入します。空白文字または他の文字を区切り文字として指定できます。区切り文字は、ターゲットフィールドのソース値の間に挿入されます。FirstName、MiddleName、および LastName などの複数のソースフィールド値を、CustomerName などの1つのターゲットフィールドに組み合わせるのが、この変換の一般的な使用方法です。 |
| Contains | ソース値を評価し、指定したパラメーター値が含まれる値があるかどうかを判断します。指定のパラメーター値が含まれるソース値がある場合、true がターゲットフィールドに挿入されます。指定のパラメーター値が含まれるソース値がない場合は、false がターゲットフィールドに挿入されます。<br><br>たとえば、特定の顧客に関連するアクティビティーを追跡するとします。この場合、各コレクションメンバーに顧客情報が含まれるソースコレクションフィールドを選択します。Value パラメーターに特定のメールアドレスを指定します。コレクションで指定のメールアドレスが見つかった場合、true がターゲットフィールドに挿入されます |

| 多重変換 | 説明 |
| --- | --- |
| Count | ソース値の数をターゲットフィールドに挿入します。これは、ソースフィールドがコレクションである場合に便利です。データマッパーによって、コレクションのサイズがターゲットフィールドに挿入されます。<br><br>たとえば、アイテムオブジェクトのコレクションである **Order** ソースフィールドを選択するとします。**Count** 変換を適用すると、その Order のアイテム数がターゲットフィールドに挿入されます。<br><br>また、4 つの個別のソースフィールドを選択した場合は **4** がターゲットフィールドに挿入されます。 |
| Divide | 最初のソース値を 2 つ目のソース値で割り、その値をターゲットフィールドに挿入します。3 つ以上のソース値がある場合は、次の値で割って除算を継続します。例として、**{1000, 100, 10}** が含まれる **numbers[]** コレクションについて考えてみましょう。**1000** を **100** で割った値は **10** になります。その **10** を **10** で割ると、値は **1** になります。**1** がターゲットフィールドに挿入されます。 |
| Format | 指定したテンプレートのプレースホルダーを選択したソースフィールドの値に置き換えます。置き換え後の文字列がターゲットフィールドに挿入されます。たとえば、以下の 3 つのソースフィールドを選択したとします。<br><br>time<br>name<br>text<br><br>**Format** 変換を選択し、**Template** パラメーターに以下を指定します。<br><br>**At $time, $name tweeted: $text**<br><br>その結果、ターゲットフィールドは次のようになります。<br><br>At 8:00 AM, Aslan tweeted: ROAR!<br><br>これは、Java や C などのプログラミング言語で使用できるメカニズムに似ています。 |
| Item At | 選択したソースフィールドに対して、指定したインデックスの値を見つけ、その値をターゲットフィールドに挿入します。ソースフィールドは、区切り文字で区切られた複数の値が含まれるコレクションまたはフィールドである必要があります。<br><br>たとえば、選択したソースフィールドが顧客メールアドレスのコレクションであるとします。**Item At** 変換の選択後、**0** を **Index** パラメーターフィールドに指定します。データマッパーによって、インデックス 0 の最初のメールアドレスがターゲットフィールドに挿入されます。 |
| Maximum | ソース値を評価し、最大値をターゲットフィールドに挿入します。ソース値は数字である必要があります。 |

| 多重変換 | 説明 |
| --- | --- |
| Minimum | ソース値を評価し、最小値をターゲットフィールドに挿入します。ソース値は数字である必要があります。 |
| Multiply | 最初のソース値と 2 つ目のソース値を掛けて、その値をターゲットフィールドに挿入します。3 つ以上のソース値がある場合は、次の値を掛けて乗算を継続します。例として、**{10, 100, 1000}** が含まれる **numbers[]** コレクションについて考えてみましょう。**10** と **100** を掛けた値は **1000** になります。その **1000** を **1000** で掛けた値は **1000000** になります。**1000000** がターゲットフィールドに挿入されます。 |
| Subtract | 最初のソース値から 2 つ目のソース値を引き、その値をターゲットフィールドに挿入します。3 つ以上のソース値がある場合は、次の値を引いて減算を継続します。例として、**{100, 90, 9}** が含まれる **numbers[]** コレクションについて考えてみましょう。**100** から **90** を引いた値は **10** となります。さらに、その **10** から **9** を引いた値は **1** になります。**1** がターゲットフィールドに挿入されます。 |

**その他のリソース**

- 複数のソースフィールドを 1 つのターゲットフィールドに組み合わせる
- 1 つのソースフィールドを複数のターゲットフィールドに分割

### 7.13.2. 利用可能な変換の説明

以下の表では利用可能な変換について説明します。日付と時刻のタイプは、これらの概念のいずれの形式にも汎用的に参照されます。よって、数字には **integer**、**long**、**double** などが含まれます。日付には **date**、**Time**、**ZonedDateTime** などが含まれます。

| 変換 | 入力タイプ | 出力タイプ | パラメーター (* = 必須) | 説明 |
| --- | --- | --- | --- | --- |
| **AbsoluteValue** | number | number | なし | 数字の絶対値を返します。 |
| **AddDays** | date | date | **days** | 日付に日を追加します。デフォルトは 0 日です。 |
| **AddSeconds** | date | date | **seconds** | 日付に秒数を追加します。デフォルトは 0 秒です。 |
| **Append** | string | string | string | 文字列の最後に文字列を追加します。デフォルトでは、何も追加しません。 |

| 変換 | 入力タイプ | 出力タイプ | パラメーター (* = 必須) | 説明 |
|---|---|---|---|---|
| **Camelize** | string | string | なし | 空白を削除し、最初の単語を小文字にし、その後の各単語の最初の文字を大文字にして、キャメルケースを用いた文字列に変換します。 |
| **Capitalize** | string | string | なし | 文字列の最初の文字を大文字にします。 |
| **Ceiling** | number | number | なし | 数字の整数上限を返します。 |
| **Contains** | any | Boolean | **value** | フィールドに指定された値が含まれる場合、true を返します。 |
| **CopyTo** | 任意の単純な型 | 配列内の任意の単純型 | index* | コレクションの場合は、文字列フィールドの値をターゲットコレクションの指定されたフィールドに分割せずにコピーします。index パラメーターは、ターゲットコレクション内のフィールドのインデックスを定義します。デフォルトは **0** (ターゲットコレクションの最初のフィールド) です。 |

| 変換 | 入力タイプ | 出力タイプ | パラメーター (* = 必須) | 説明 |
|---|---|---|---|---|
| **ConvertAreaUnit** | number | number | **fromUnit*** <br> **toUnit** * | 範囲を表す数値を別の単位に変換します。From Unit および To Unit メニューから **fromUnit** および **toUnit** パラメーターに適した単位を選択します。**Square Foot**、**Square Meter**、または **Square Mile** を選択できます。 |
| **ConvertDistanceUnit** | number | number | **fromUnit** * <br> **toUnit** * | 距離を表す数値を別の単位に変換します。From Unit および To Unit メニューから **fromUnit** および **toUnit** パラメーターに適した単位を選択します。**Foot**、**Inch**、**Meter**、**Mile**、または **Yard** を選択できます。 |
| **ConvertMassUnit** | number | number | **fromUnit** * <br> **toUnit** * | 質量を表す数値を別の単位に変換します。From Unit および To Unit メニューから **fromUnit** および **toUnit** パラメーターに適した単位を選択します。**Kilogram** または **Pound** を選択できます。 |

| 変換 | 入力タイプ | 出力タイプ | パラメーター (* = 必須) | 説明 |
|---|---|---|---|---|
| **ConvertVolumeUnit** | number | number | **fromUnit** *<br>**toUnit** * | 体積を表す数値を別の単位に変換します。**From Unit** および **To Unit** メニューから **fromUnit** および **toUnit** パラメーターに適した単位を選択します。**Cubic Foot**、**Cubic Meter**、**Gallon US Fluid**、または **Liter** を選択できます。 |
| **DayOfWeek** | date | number | なし | 日付に対応する週 (1 から 7) を返します。 |
| **DayOfYear** | date | number | なし | 日付に対応する年間通算日 (1 から 366) を返します。 |
| **EndsWith** | string | Boolean | **string** | 文字列が指定の **string** で終わり、両方の文字列で大文字と小文字が同じである場合に true を返します。 |
| **Equals** | any | Boolean | **value** | 入力フィールドが指定の **value** と同等で、大文字と小文字がフィールドと値で同じ場合に true を返します。 |
| **FileExtension** | string | string | なし | ファイル名を表す文字列から、ドットなしでファイル拡張子を返します。 |
| **floor** | number | number | なし | 数字の整数下限を返します。 |

| 変換 | 入力タイプ | 出力タイプ | パラメーター (* = 必須) | 説明 |
|---|---|---|---|---|
| **Format** | any | string | **template** * | **template** で各プレースホルダー (**%s** など) を入力フィールドの値に置き換え、結果が含まれる文字列を返します。これは、Java や C などのプログラミング言語で使用できるメカニズムに似ています。 |
| **IndexOf** | string<br>最初の文字はインデックス 0。 | number | **string**<br>この文字列の入力文字列を検索します。 | パラメーター文字列の最初の文字である、入力文字列の文字のインデックスを返します。パラメーター文字列が見つからないと **-1** が返されます。 |
| **IsNull** | any | Boolean | なし | フィールドが null の場合、true を返します。 |
| **LastIndexOf** | string<br>最初の文字はインデックス 0。 | number | **string**<br>この文字列の入力文字列を検索します。 | パラメーター文字列の最後の文字である、入力文字列の文字のインデックスを返します。パラメーター文字列が見つからないと **-1** が返されます。 |
| **Length** | any | number | なし | フィールドの長さを返し、フィールドが null の場合は **-1** を返します。 |
| **Lowercase** | string | string | なし | 文字列を小文字に変換します。 |

| 変換 | 入力タイプ | 出力タイプ | パラメーター (* = 必須) | 説明 |
|---|---|---|---|---|
| **Normalize** | string | string | なし | 連続した空白文字を単一のスペースに置き換え、文字列の前後にある空白文字を切り取ります。 |
| **PadStringLeft** | string | string | **padCharacter** *<br>**padCount** * | **padCharacter** に指定された文字を文字列の最初に挿入します。これを **padCount** に指定された回数行います。 |
| **PadStringRight** | string | string | **padCharacter** *<br>**padCount** * | **padCharacter** に指定された文字を文字列の最後に挿入します。これを **padCount** に指定された回数行います。 |
| **Prepend** | string | string | **string** | **string** を文字列の最初に追加します。デフォルトでは何も追加しません。 |
| **ReplaceAll** | string | string | **match** *<br>**newString** | 文字列で、提供された一致する文字列をすべて提供された **newString** に置き換えます。デフォルトの **newString** は空の文字列です。 |
| **ReplaceFirst** | string | string | **match** *<br>**newString** * | 文字列で、最初に見つかった指定の **match** 文字列を指定の **newString** に置き換えます。デフォルトの **newString** は空の文字列です。 |

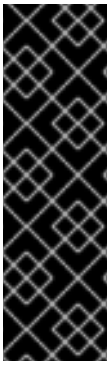
| 変換 | 入力タイプ | 出力タイプ | パラメーター (* = 必須) | 説明 |
|---|---|---|---|---|
| **Repeat** | 任意の単純な型 | 配列内の任意の単純型 | **count** * (ソースフィールドが配列に **ない** 場合は必須) | 単純型を配列型にマップする場合は、複数の配列要素に単純型から取得した値を入力します。 |
| **Round** | number | number | なし | 数字の四捨五入した整数を返します。 |
| **SeparateByDash** | string | string | なし | 空白文字、コロン (:)、アンダーライン (_)、プラス記号 (+)、および等号 (=) をハイフン (-) に置き換えます。 |
| **SeparateByUnderscore** | string | string | なし | 空白文字、コロン (:)、ハイフン (-)、プラス記号 (+)、および等号 (=) をアンダーライン (_) に置き換えます。 |
| **Split** | 複合タイプ | any | なし | 複合ソースフィールドの区切り文字に基づいて、複合ソースフィールドを複数のターゲットフィールドに分割します。デフォルトの区切り文字は空白文字です。 |
| **StartsWith** | string | Boolean | **string** | 文字列が指定された文字列 (大文字と小文字を区別) で始まる場合は true を返します。 |

| 変換 | 入力タイプ | 出力タイプ | パラメーター (* = 必須) | 説明 |
|---|---|---|---|---|
| **Substring** | string | string | **startIndex** *<br>**endIndex** | 指定された **startIndex** (含まれる) から指定された **endIndex** (含まれない) までの文字列のセグメントを取得します。両方のインデックスはゼロで始まります。**startIndex** は包括的です。**endIndex** は排他的です。**endIndex** のデフォルト値は文字列の長さになります。 |
| **SubstringAfter** | string | string | **startIndex** *<br>**endIndex**<br>**match** * | 指定された **startIndex** (範囲に含まれる) から指定された **endIndex** (範囲に含まれない) までの **match** 文字列の後にある文字列のセグメントを取得します。両方のインデックスはゼロで始まります。**endIndex** のデフォルト値は、指定の **match** 文字列の後にある文字列の長さになります。 |

| 変換 | 入力タイプ | 出力タイプ | パラメーター (* = 必須) | 説明 |
|---|---|---|---|---|
| **SubstringBefore** | string | string | **startIndex** *<br>**endIndex**<br>**match** * | 指定された **startIndex** (範囲に含まれる) から指定された **endIndex** (範囲に含まれない) までの **match** 文字列の前にある文字列のセグメントを取得します。両方のインデックスはゼロで始まります。**endIndex** のデフォルト値は、指定の **match** 文字列の前にある文字列の長さになります。 |
| **Trim** | string | string | なし | 文字列の前後にある空白文字を削除します。 |
| **TrimLeft** | string | string | なし | 文字列の前にある空白文字を削除します。 |
| **TrimRight** | string | string | なし | 文字列の後にある空白文字を削除します。 |
| **Uppercase** | string | string | なし | 文字列を大文字に変換します。 |

## 7.14. 条件のマッピングへの適用

インテグレーションによっては、条件処理をマッピングに追加すると便利なものがあります。たとえば、ソースの zip コードフィールドに、ターゲットの zip コードフィールドをマッピングすると仮定し、ソースの zip コードフィールドが空白の場合には、ターゲットのフィールドに **99999** と入力することにします。これを実行するには、ソースの zip コードフィールドをテストして、空かどうかを判断するための式を指定し、空の場合は **99999** をターゲットの zip コードフィールドにマップします。



**重要**

条件のマッピングへの適用はテクノロジープレビュー機能です。テクノロジープレビュー機能は、Red Hat の本番環境のサービスレベルアグリーメント (SLA) ではサポートされず、機能的に完全ではないことがあるため、Red Hat は本番環境での使用は推奨しません。Red Hat は実稼働環境でこれらを使用することを推奨していません。これらの機能により、近日発表予定の製品機能をリリースに先駆けてご提供でき、お客様は開発プロセス時に機能をテストして、フィードバックをお寄せいただくことができます。Red Hat のテクノロジープレビュー機能のサポート範囲に関する詳細は、https://access.redhat.com/ja/support/offerings/techpreview を参照してください。

データマッパーがサポートする式は Microsoft Excel の式と似ていますが、Microsoft Excel のすべての式構文をサポートしていません。条件式は、個別のフィールド、またはコレクションにあるフィールドを参照できます。

各マッピングには 1 つの条件を定義でき、条件を定義しないことも可能です。

以下の手順は、条件をマッピングに適用できるようにします。

**注記:** 条件をマッピングに追加した後、Source および Target の変換オプションが無効になります。変換を条件式内に追加する必要があります。

**前提条件**

- **データマッパー** ステップでフィールドをマッピングします。
- Microsoft Excel 式に詳しいか、マッピングに適用する条件付きの式がある必要があります。

**手順**

1. データタイプが表示されない場合は、 をクリックして表示します。
これは、条件を指定するための要件ではありませんが、データタイプを表示すると便利です。

2. 条件を適用するマッピングを作成するか、現在選択しているマッピングが条件を適用するマッピングであることを確認します。たとえば、以下のマッピングについて考えてみましょう。

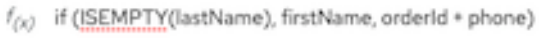

3. 左上の をクリックして条件式入力フィールドを表示します。
データマッパーは、式フィールドに現在のマッピングにあるソースフィールドの名前を自動的に表示します。以下に例を示します。


式入力フィールドでは、ソースフィールドの順序は、マッピングの作成時に選択した順序になります。これは、データマッパーがこの順序でフィールド値を連結し、結果をターゲットフィールドに挿入するのがデフォルトのマッピング動作であるため、重要になります。この例では、このマッピングの作成では、最初に **lastName** が選択されてから **firstName** が選択されます。

4. 式入力フィールドを編集して、データマッパーがマッピングに適用する条件式を指定します。サポートされる条件式の詳細は、次の手順に従います。
条件マッピングに変換を含める場合は、条件式に変換を追加する必要があります。

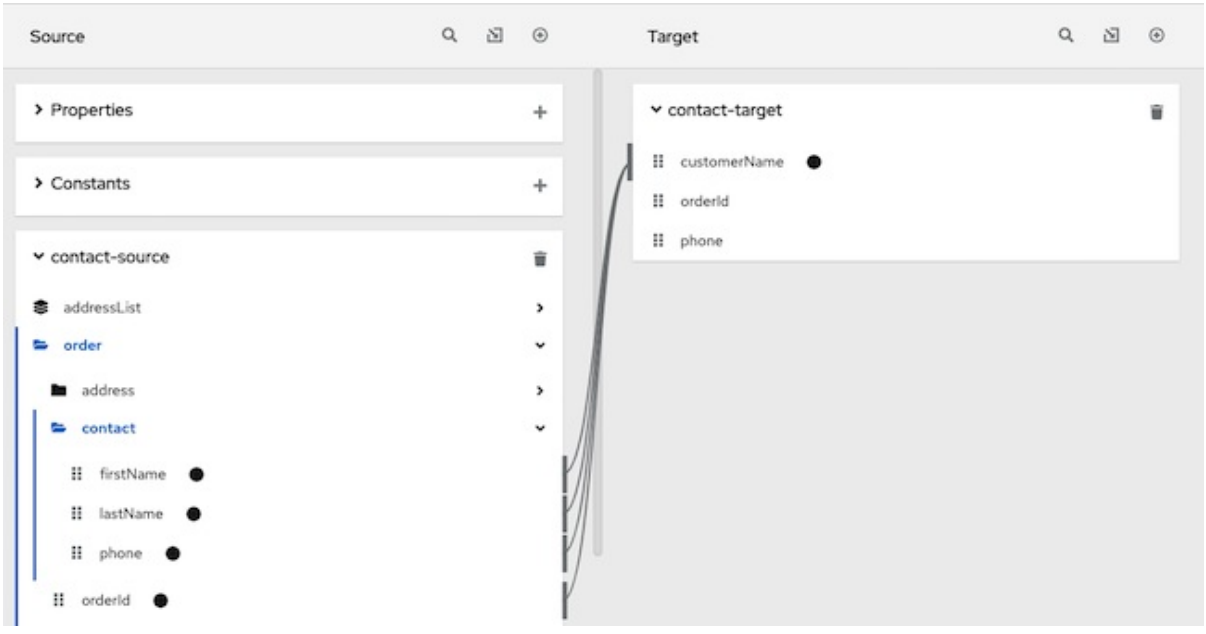
   式を指定するときに、@ と入力してからフィールドの名前を入力します。データマッパーは入力内容に一致するフィールドのリストを表示します。式に指定するフィールドを選択します。

   フィールド名を式に追加するとき、データマッパーはそのフィールドをマッピングに追加します。たとえば、以下の条件式について考えてみましょう。




   実行中、データマッパーによって **lastName** が空であると判断されると、firstname フィールドのみを **firstName** フィールドのみをターゲット **customerName** フィールドにマップします。**lastName** フィールドに値が含まれ、空でない場合、データマッパーはソースの **orderId** および **phone** フィールドの値を連結し、結果を **customerName** フィールドに挿入します。(以下の例は論理の仕組みを示していますが、**lastName** フィールドに値があるときは、データマッパーと使って単にマッピングを実行し、他の値をターゲットにマップしない可能性が高いため、実践的な例とは言えません。)

   この例では、式の入力後にデータマッピングは次のようになります。




   条件式で式が適用されるマッピングにあるフィールド名を削除すると、データマッパーはそのフィールドをマッピングから削除します。つまり、マッピングのすべてのフィールド名は条件式にある必要があります。

5. マッピングプレビューフィールドが表示されていない場合は、 をクリックして表示します。

6. ソースのプレビュー入力フィールドにサンプルデータを入力して、ターゲットフィールドが正しい値であることを確認します。

7. オプションで、変換を適用する各フィールドの横にある をクリックして、選択したマッピングの1つ以上のソースフィールドまたはターゲットフィールドに変換を適用し、プルダウンメニューから目的の変換を選択します。
たとえば、この手順と同じマッピングで Mapping Details パネルにて **Uppercase** 変換を **firstName** フィールドに適用できます。これをテストするには、**firstName** フィールドのプレビュー入力フィールドにデータを入力します。

8. 必要に応じて条件式を編集して、目的の結果を取得します。

**条件式でサポートされる関数**

- **SELECT(FILTER(source-collection-name, source-field-name1 != 'v1' ), source-field-name2)**
複数のフィールドを持つコレクションの場合、データマッパーは、(同じコレクション内の) 別のソースフィールドの値に基づいて、1つのソースフィールドの値をフィルタリングします。たとえば、コレクション (**person**) に **name** および **gender** を含む複数のフィールドが含まれている場合は、次の条件式を使用して、**gender** フィールドの値で **name** をフィルター処理し、男性の名前のみがターゲットフィールドにマップされるようにすることができます。

  **SELECT(FILTER(person, gender = 'male' ), name)`**

- **ISEMPTY(source-field-name1 [+ source-field-name2])**
**ISEMPTY()** 関数の結果はブール値です。条件を適用するマッピングのソースフィールドの名前である引数を最低でも1つ指定します。指定したソースフィールドが空の場合、**ISEMPTY()** 関数は true を返します。

  必要に応じて、次の例のように + (連結) 演算子を追加のフィールドと追加します。

  **ISEMPTY(lastName + firstName)**

  この式は、ソースフィールドの **lastName** と **firstName** が両方空の場合に true に評価されます。

  多くの場合、 **ISEMPTY()** 関数は **IF()** 関数の最初の引数になります。

- **IF(boolean-expression, then, else)**
**boolean-expression** が true に評価されると、データマッパーは **then** を返します。**boolean-expression** が false に評価されると、データマッパーは **else** を返します。3 つの引数がすべて必要です。最後の引数は null にすることができます。つまり、**boolean-expression** が false に評価されると何もマッピングされません。

  たとえば、ターゲット **customerName** フィールドの **lastName** および **firstName** を組み合わせるマッピングについて考えてみましょう。この条件式を指定できます。

  **IF (ISEMPTY(lastName), firstName, lastName + ',' + firstName )**

  実行中、データマッパーは **lastName** フィールドを評価します。

  - **lastName** フィールドが空で、 **ISEMPTY(lastName)** が true を返す場合、データマッパーは **firstName** の値のみをターゲット **customerName** フィールドに挿入します。

- **lastName** フィールドに値が含まれ、**ISEMPTY(lastName)** が false を返す場合、データマッパーは **firstName** の値とコンマの後に **lastName** の値をターゲットの **customerName** フィールドにマップします。
  この式の 3 つ目の引数が null であった場合の動作について考えてみましょう。

  **IF (ISEMPTY(lastName), firstName, null )**

  実行中、データマッパーは **lastName** フィールドを評価します。

- 前の例のように、**lastName** フィールドが空で、**ISEMPTY(lastName)** が true を返す場合、でデータマッパーは **firstName** の値のみをターゲットの **customerName** フィールドに挿入します。

- しかし、3 つ目の引数が null であり、**lastName** フィールドに値が含まれ、**ISEMPTY(lastName)** が false を返す場合は、データマッパーはターゲットの **customerName** フィールドに何もマップしません。

- **LT(x,y)** または **<(x,y)**
  データマッパーは **x** および **y** を評価し、小さい方の値を返します。**x** と **y** はどちらも数字である必要があります。

- **TOLOWER(string)**
  データマッパーは指定の文字列を小文字に変換し、返します。

**表7.1 条件式でサポートされる演算子**

| 演算子 | 説明 |
|---|---|
| **+** | 数値の値または連結文字列値を加算します。 |
| **-** | 1 つの数値を別の数値から減算します。 |
| * | 数値を乗算します。 |
| \ | 数値を除算します。 |
| **&&**<br>And | 左と右のオペランドが両方 true の場合に true を返します。各オペランドは、ブール値を返す必要があります。 |
| \|\|<br>Or | 左のオペランド、右のオペランド、または両方のオペランドが true の場合に true を返します。各オペランドは、ブール値を返す必要があります。 |
| **!** | Not (否定) |
| **>**<br>Greater than | 左の数値オペランドが右の数値のオペランドよりも大きい場合に true を返します。 |
| <<br>Less than | 左の数値オペランドが右の数値のオペランドよりも小さい場合に true を返します。 |

| | |
|---|---|
| **==**<br>Equal | 左の数値オペランドと右の数値のオペランドが同じ場合に true を返します。 |

## 7.15. データマッピングのトラブルシューティング

データマッパーでは、以前のインテグレーションステップで提供できるソースフィールドの最大セットが表示されます。ただし、すべてのコネクションによって、表示された各ソースフィールドにデータが提供されるわけではありません。たとえば、サードパーティーアプリケーションへの変更によって、特定のフィールドにデータが提供されなくなる可能性があります。インテグレーションの作成時に、データマッピングが想定どおりに機能していないことが判明した場合は、マップするソースフィールドに予想されるデータが含まれていることを確認してください。

すでにマップされたフィールドに影響するデータシェイプの変更により、データマッパーがドキュメントをロードできない場合があります。このような場合、影響を受けるフィールドをマップするデータマッパーステップを編集しようとすると、データマッパーはソースおよびターゲットパネルを表示できません。その代わりに、ドキュメントのロードまたは検索が不可能であることを示すエラーが表示されます。エラーメッセージは、以下のメッセージの 1 つのようになります。

- **Data Mapper UI Initialization Error: Could not load document '-La_rwMD_ggphAW6nE9o': undefined undefined**
- **Could not find document for mapped field 'last_name' at URI atlas:json:-LaX4LMC1CfVJYp3JXM6**

このデータマッパーステップを削除し、更新されたフィールドをマップする新しいデータマッパーステップに置き換える必要があります。

マップされたフィールドのデータシェイプを変更するには、必ずマッピングをやり直す必要がありますが、常にデータマッパーステップを削除する必要はありません。たとえば、XML インスタンスが入力データシェイプを指定し、ユーザーが要素の名前を変更すると、データマッパーは古いフィールド名をマップ元またはマップ先とするマッピングを削除します。更新された名前でフィールドをマップ先またはマップ元としてマッピングする必要があります。

以下の方法で、マップされたフィールドのデータシェイプを変更できます。

- API プロバイダーインテグレーションでフローの編集中に、オペレーションを定義する OpenAPI ドキュメントを編集します。
  オペレーションの応答のデータシェイプを変更すると、データマッパーがドキュメントをロードできなくなります。
- フローでは、以下に記載されているコネクションの 1 つの入力データタイプや出力データタイプを編集します。
  - Amazon S3
  - AMQ
  - AMQP
  - Dropbox
  - FTP/SFTP
  - HTTP/HTTPS

- Kafka
- IRC
- MQTT

# 第8章 インテグレーションの管理

一般的なセットアップには、Fuse Online 開発環境、Fuse Online テスト環境、および Fuse Online デプロイメント環境が含まれます。そのため、Fuse Online は Fuse Online 環境からインテグレーションをエクスポートし、そのインテグレーションを別の Fuse Online 環境にインポートする機能を提供します。インテグレーションの管理に関する情報と手順は、特に記述がない限り、各種の Fuse Online 環境で同じになります。

インテグレーションの管理に役立つ情報は、以下を参照してください。

- 「インテグレーションのライフサイクルの処理」
- 「インテグレーションを使用可能および使用不可能にする」
- 「インテグレーションの実行に関するログ情報」
- 「インテグレーションの監視」
- 「インテグレーションのテスト」
- 「インテグレーション実行のトラブルシューティングに関するヒント」
- 「インテグレーションの更新」
- 「インテグレーションのメモリーおよび CPU 設定属性の調整」
- 「インテグレーションの削除」
- 「インテグレーションの別の環境へのコピー」

## 8.1. インテグレーションのライフサイクルの処理

インテグレーションを作成およびパブリッシュした後に、インテグレーションの動作を更新する必要がある場合があります。パブリッシュされたインテグレーションのドラフトを編集し、実行中のバージョンを更新されたバージョンに置き換えることができます。そのため、Fuse Online はインテグレーションごとに、複数のバージョンと各バージョンの状態を維持します。インテグレーションのバージョンや状態を理解することは、インテグレーションの管理に役立ちます。

### インテグレーションのバージョンの説明

各 Fuse Online 環境では、インテグレーションごとに複数のバージョンを保持できます。複数のインテグレーションバージョンをサポートすることには、以下のような利点があります。

- 適切に動作しないバージョンをパブリッシュした場合、インテグレーションの適切なバージョンに戻して実行することができます。これには、適切に動作しないバージョンを停止して、適切に動作するバージョンを起動します。
- 要件またはツールの変更に応じて、インテグレーションを段階的に更新できます。新しいインテグレーションを作成する必要はありません。

Fuse Online は、インテグレーションの新しいバージョンが実行されるたびに、新しいバージョン番号を割り当てます。たとえば、Twitter to Salesforce Sample Integration をパブリッシュをすると仮定します。実行後に、インテグレーションを更新して、別のアカウントを使用して Twitter に接続します。その後、更新されたインテグレーションを公開します。Fuse Online は実行中のバージョンのインテグレーションを停止し、インクリメントされたバージョン番号でインテグレーションの更新されたバージョンをパブリッシュします。

実行されていた最初のインテグレーションはバージョン 1 になります。現在稼働している更新されたインテグレーションはバージョン 2 になります。バージョン 2 を編集し (たとえば、別のアカウントを使用して Salesforce に接続するためなど)、そのバージョンをパブリッシュする場合、インテグレーションのバージョン 3 になります。

インテグレーションのドラフトバージョンは必ず 1 つになります。Fuse Online には、インテグレーションのドラフトバージョンの定義がありますが、このバージョンのインテグレーションを実行することはありません。インテグレーションのドラフトバージョンには数字がありません。インテグレーションの編集時に、インテグレーションのドラフトバージョンを更新します。

Fuse Online では、インテグレーションの概要ページでインテグレーションのバージョンリストを確認することができます。このページを表示するには、左側のナビゲーションパネルで **Integrations** をクリックします。対象のインテグレーションのエントリーで、**View** をクリックします。

### インテグレーション状態の説明

Fuse Online では、インテグレーションのバージョンリストで、各エントリーがそのバージョンの状態を示します。それは以下の 1 つになります。

| 状態 | 説明 |
|---|---|
| **Running** | **Running** バージョンが実行されています。これはサービスです。インテグレーションの 1 つのバージョンのみを稼働することができます。つまり、1 度に 1 つのバージョンのみが **Running** 状態になることができます。 |
| **Stopped** | **Stopped** バージョンは実行していません。インテグレーションのドラフトバージョンは **Stopped** 状態になります。ある時点で実行されていたインテグレーションが停止されると **Stopped** 状態になります。<br><br>このインテグレーションで **Running** 状態のバージョンがない場合、停止されたバージョンを起動できます。 |
| **Pending** | **Pending** のバージョンは移行中です。Fuse Online はこのバージョンのインテグレーションの開始処理中または停止処理中のいずれかになりますが、インテグレーションは実行または停止されていません。 |
| **Error** | **Error** 状態にあるインテグレーションバージョンは、起動中または実行中に OpenShift エラーが発生しました。そのエラーによって起動または実行が停止されました。これが発生した場合、適切に実行される以前のインテグレーションバージョンを開始してください。または、テクニカルサポートにお問い合わせください。これには、Fuse Online ページの右上にある  アイコンを選択し、**Support** を選択します。 |

## 8.2. インテグレーションを使用可能および使用不可能にする

インテグレーションの作成後、これをドラフトとして保存したり、パブリッシュして実行を開始することができます。インテグレーションをパブリッシュする場合、Fuse Online は必要なリソースをアセンブルし、インテグレーションランタイムをビルドします。さらに、インテグレーションを実行する OpenShift Pod をデプロイし、インテグレーションの実行を開始します。

いつでも、ボタンをクリックしてインテグレーションの実行を停止することができます。インテグレーションを再度開始した場合、Fuse Online には必要なものがすでに揃っているため、パブリッシュして初めて実行したときよりも開始に時間がかかりません。

最初にインテグレーションのバージョンを開始するプロセスは、インテグレーションのパブリッシュと呼ばれます。詳細は以下のセクションを参照してください。

- 「インテグレーションのパブリッシュ」
- 「インテグレーションの停止」
- 「インテグレーションの開始」
- 「インテグレーションの旧バージョンの再起動」

### 8.2.1. インテグレーションのパブリッシュ

初めてインテグレーションのバージョンを実行するには、パブリッシュします。インテグレーションをパブリッシュするとインテグレーションランタイムがビルドおよびデプロイされます。インテグレーションは実行を開始します。インテグレーションをパブリッシュした後、停止および再起動することができます。インテグレーションの 1 つのバージョンを 1 度に実行することができます。

#### パブリッシュの代替

最初にインテグレーションを実行するには、以下の 1 つを実行してパブリッシュします。

- インテグレーションの作成または編集手順の最後で、右上の **Publish** をクリックします。
- インテグレーションのドラフトバージョンをパブリッシュします。
  1. Fuse Online パネルの左側にある **Integrations** をクリックします。
  2. インテグレーションのリストで、ドラフトエントリーの右側の  をクリックし、**Publish** を選択します。

#### パブリッシュの進捗

Fuse Online は、複数のステージがあるパブリッシュプロセスの進捗を表示します。

1. **Assembling** は、インテグレーションのビルドに必要な Pod リソースを作成します。
2. **Building** は、インテグレーションをデプロイするための準備をします。
3. **Deploying** は、インテグレーションを実行する Pod のデプロイメントを待ちます。
4. **Starting** は、Pod がインテグレーションの実行を開始するまで待機します。
5. **Deployed** はインテグレーションが実行されていることを示します。

起動中に **View Logs** をクリックすると、起動時の情報を提供する OpenShift ログを表示できます。

#### パブリッシュ後のインテグレーション状態

インテグレーションのパブリッシュが完了すると、**Running** 状態がインテグレーション名の横に表示されます。Pod がインテグレーションを実行している状態になります。

### 8.2.2. インテグレーションの停止

インテグレーションごとに、稼働中のバージョンが必ず 1 つ存在します。実行中のバージョンが **Running** 状態になります。いつでも、インテグレーションの実行を停止できます。

#### 前提条件

停止するインテグレーションは **Running** 状態になります。

#### 手順

1. Fuse Online パネルの左側にある **Integrations** をクリックします。
2. インテグレーションのリストで、実行を停止するインテグレーションのエントリーを特定します。このエントリーは、このインテグレーションが **Running** 状態であることを示しています。
3. このインテグレーションのエントリーの右端にある  をクリックし、**Stop** を選択します。

#### 結果

Fuse Online がインテグレーションの実行を停止します。インテグレーションリストのインテグレーションのエントリーで、**Stopping** が表示された後に **Stopped** が表示されます。

### 8.2.3. インテグレーションの開始

インテグレーションを初めて開始するプロセスは、インテグレーションの実行前に Fuse Online がインテグレーションランタイムを構築する必要があるため、インテグレーションをパブリッシュすると呼ばれます。いつでも、インテグレーションの実行を停止して、再度開始できます。

#### 前提条件

起動するインテグレーションが **Stopped** 状態である必要があります。

#### 手順

1. 左のナビゲーションパネルで **Integrations** をクリックします。
2. インテグレーションのリストで、開始するインテグレーションのエントリーの右にある  をクリックします。
3. **Start** を選択します。

#### 結果

Fuse Online は、そのインテグレーションバージョンの状態として **Starting** を表示し、インテグレーションが再度稼働したときに **Running** を表示します。

### 8.2.4. インテグレーションの旧バージョンの再起動

思うように動作しないインテグレーションをパブリッシュする可能性があります。このような場合、適切に動作しないバージョンを停止し、以前パブリッシュした適切に実行されるバージョンに置き換えます。

**前提条件**

- 稼働しているバージョンのインテグレーションを停止する必要があります。
- 別のバージョンのインテグレーションがあり、それを実行する必要があります。

**手順**

1. 左側のパネルで **Integrations** をクリックし、この環境のインテグレーションの一覧を表示します。
2. 旧バージョンをパブリッシュするインテグレーションのエントリーをクリックします。Fuse Online はインテグレーションのバージョンリストを表示します。
3. 実行しているバージョンのエントリーで、右端にある  をクリックし、**Stop** を選択します。
4. **OK** をクリックして、このバージョンの実行を停止することを確認します。
5. ページの上部付近にあるインテグレーション名のの右側に **Stopped** が表示されるまで待ちます。
6. そのまま旧バージョンをパブリッシュする場合は、次の手順を飛ばします。また、旧バージョンをパブリッシュする前に、更新することもできます。
   a. 更新するインテグレーションバージョンのエントリーで、右端の  をクリックし、**Replace Draft** を選択します。
   b. 必要に応じてインテグレーションを更新します。
   c. 更新が完了したら、右上の **Publish**をクリックし、再度 **Publish** をクリックして確認します。これは、次の 2 つのステップの代わりになります。
7. そのまま旧バージョンをパブリッシュする場合は、再度実行を開始するインテグレーションバージョンのエントリーで、右端の  をクリックし、**Start** を選択します。
8. **Start** をクリックして、このバージョンのインテグレーションを開始することを確認します。

**結果**

Fuse Online はインテグレーションを開始します。これには数分かかります。インテグレーションの稼働時、**Running version n** がインテグレーション名の右側に表示されます。

## 8.3. インテグレーションの実行に関するログ情報

Fuse Online は、インテグレーションの実行ごとに、以下のアクティビティー情報をフローの各ステップに提供します。

- ステップが実行された日付と時刻
- ステップの実行にかかった時間
- 実行が成功したかどうか
- 実行に成功しなかった場合のエラーメッセージ

Fuse Online でこの情報を表示するには、インテグレーションの概要を表示して **Activity** タブをクリックします。

インテグレーション実行に関する詳細を取得するには、インテグレーションフローにログステップを追加して、インテグレーションが処理するメッセージに関する情報をログに記録します。ログステップは、インテグレーションが受信する各メッセージに対して、以下を 1 つまたは複数提供できます。

- メッセージのヘッダーを含む、メッセージに関するメタデータを提供するメッセージのコンテキスト。
- メッセージの内容を提供するメッセージのボディー。
- 明示的に指定するテキストまたは Apache Camel Simple language 式の評価で指定するテキスト。

**注記**

これまでのリリースで使用できたログコネクションは、インテグレーションに追加されなくなりました。ログコネクションの代わりにログステップを追加します。

**前提条件**

- フローを作成または編集することになります。Fuse Online でインテグレーションを追加するよう要求されます。または、Fuse Online で最後のコネクションを選択するよう要求されます。

**手順**

1. フロービジュアライゼーションにて、ログステップを追加する場所でプラス記号をクリックします。Fuse Online が最後のコネクションを選択するよう要求する場合は、このステップをスキップします。
2. **Log** をクリックします。
3. **Configure Log Step** ページで、ログするコンテンツを選択します。**Custom Text** を選択した場合、テキスト入力フィールドに以下のいずれかを入力します。
   - ログするテキスト
   - Camel Simple 言語式

   式を入力すると、Fuse Online は式を解決し、生成されたテキストをログに記録します。
4. ログステップ設定が完了したら、**Next** をクリックしてフローにステップを追加します。

**次のステップ**

フローの完了後にインテグレーションをパブリッシュすると、新しいログステップからの出力が表示されます。

**その他のリソース**

ログステップのあるフローの実行後、ログステップからの出力はインテグレーションの **Activity** タブに表示されます。インテグレーションアクティビティー情報の表示 を参照してください。

## 8.4. インテグレーションの監視

Fuse Online は、インテグレーションの実行を監視するさまざまな方法を提供します。参照:

- 「インテグレーション履歴の表示」
- 「インテグレーションのアクティビティーに関する情報の表示」
- 「特定インテグレーションのメトリクスの表示」
- 「Fuse Online 環境のメトリクスの表示」

### 8.4.1. インテグレーション履歴の表示

Fuse Online はインテグレーションの各バージョンを維持します。各インテグレーションのバージョンのリストを常に確認できます。

**手順**

1. 左側のパネルで **Integrations** をクリックし、環境のインテグレーションの一覧を表示します。
2. バージョンを表示するインテグレーションのエントリーの右側にある **View** をクリックします。

**結果**

表示されるページの **History** セクションに、インテグレーションのバージョンがリストされます。 アイコンは、正常に稼働している最新バージョンである現在のバージョンを特定します。各バージョンの最後に起動した日付も確認できます。

特定のバージョンを編集、開始、または停止するには、バージョンのエントリーの右にある をクリックします。実行するオペレーションを選択します。

### 8.4.2. インテグレーションのアクティビティーに関する情報の表示

Fuse Online はインテグレーションの実行ごとにアクティビティーの情報を提供します。この情報はインテグレーションのログの一部です。フローの各ステップに対して、Fuse Online は以下を提供します。

- ステップが実行された日付と時刻
- ステップの実行にかかった時間
- 実行が成功したかどうか
- 実行に成功しなかった場合のエラーメッセージ

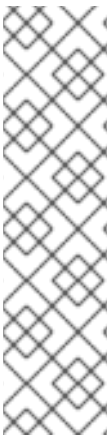
この情報はいつでも確認することができます。

**前提条件**

- アクティビティー情報を表示する現在稼働中のインテグレーションまたは過去に稼働していたインテグレーションが存在する必要があります。
- このインテグレーションは、1 回以上実行されている必要があります。

**手順**

1. 左側のパネルで **Integrations** をクリックします。
2. アクティビティー情報を表示するインテグレーションのエントリーの右側にある **View** をクリックします。
3. インテグレーションの概要ページで **Activity** をクリックします。
4. 必要に応じて、日付フィルターやキーワードフィルターを入力して、表示する実行を制限します。
5. アクティビティー情報を表示するインテグレーションの実行をクリックします。



**注記**

API Provider インテグレーションまたは Webhook ステップでは、インテグレーションの **Activity** タブにある情報は、Fuse Online インテグレーションと、呼び出すクライアントとの間の通信を反映します。HTTP または REST リクエストによって生成されたエラーは、インテグレーションの **Activity** ログには記録されません。HTTP または REST リクエストによって生成されたエラーを表示またはテストする必要がある場合、API Provider または Webhook ステップの設定時に **Include error message in the return body**オプションを確認します (デフォルトでは確認されます)。次に、呼び出し元への応答で HTTP または REST ヘッダーをチェックして、エラーメッセージが応答に含まれているかどうかを検証できます。インテグレーション Pod のログファイルで **INFO** メッセージを確認することもできます。

**その他のリソース**

- ログステップをインテグレーションに追加すると、いずれかのステップの間で追加情報を取得できます。ログステップは、受信する各メッセージに関する情報を提供し、指定するカスタムテキストを提供できます。ログステップを追加する場合、アクティビティー情報を表示するインテグレーションの実行を拡張すると、インテグレーションのステップの 1 つとして表示されます。他のステップの Fuse Online 情報を表示するように、ログステップの Fuse Online 情報を表示します。インテグレーションの実行に関するログ情報 を参照してください。

### 8.4.3. 特定インテグレーションのメトリクスの表示

Fuse Online は、各インテグレーションに以下のメトリクスを提供します。

- **Total Errors** は、過去 30 日以内にこのインテグレーションのすべての実行で発生したランタイムエラーの数を示します。
- **Last Processed** は、このインテグレーションが最後にメッセージを処理した日時を表示します。メッセージが正常に処理されたり、エラーが含まれたりする可能性があります。
- メッセージの**Total Messages**は、過去 30 日以内にこのインテグレーションの実行がすべて処理したメッセージの数になります。これには、メッセージの失敗が含まれます。
- **Uptime** は、このインテグレーションがいつ実行を開始したかと、エラーなで実行を続けている期間を示します。

**前提条件**

メトリクスを表示するインテグレーションが稼働している必要があります。

**手順**

1. 左側のパネルで **Integrations** をクリックします。
2. メトリクスを表示するインテグレーションのエントリーの右側にある **View** をクリックします。
3. インテグレーションの概要ページで **Metrics** をクリックします。

### 8.4.4. Fuse Online 環境のメトリクスの表示

Fuse Online 環境のメトリクスが Fuse Online のホームページに表示されます。

**手順**

左側のパネルで **Home** をクリックします。

**結果**

Fuse Online は以下のメトリクスを 5 秒ごとに更新します。

- この環境に定義されているインテグレーション数、稼働中のインテグレーションの数、停止されたインテグレーションの数、および保留中のインテグレーションの数。Fuse Online は、保留中のインテグレーションを停止または開始します。赤いバツ印は、実行を停止したエラーが発生した稼働状態だったインテグレーションを示しています。
- この環境で定義されたコネクションの数。
- 過去 30 日以内にこの環境でインテグレーションによって処理されたメッセージの合計数。これには、現在は稼働していない可能性のあるインテグレーションや、この環境から削除された可能性のあるインテグレーションによって処理されたメッセージも含まれます。
- アップタイムは、稼働中のインテグレーションが 1 つ以上存在した期間を示します。アップタイム開始時の日付および時間も表示されます。

## 8.5. インテグレーションのテスト

作成したインテグレーションが Fuse Online の開発環境で正しく稼働した後、そのインテグレーションを別の Fuse Online 環境で実行し、テストする場合があります。

**前提条件**

- Fuse Online 開発環境と Fuse Online テスト環境が必要です。
- Fuse Online の開発環境で、正しく稼働しているインテグレーションが存在する必要があります。

**手順**

1. インテグレーションを別の環境にコピーする 方法を確認します。
2. 開発環境からインテグレーションをエクスポートします。インテグレーションのエクスポート を参照してください。
3. インテグレーションをテスト環境にインポートします。インテグレーションのインポート を参照してください。

## 8.6. インテグレーション実行のトラブルシューティングに関するヒント

インテグレーションの動作が停止したら、アクティビティーおよび履歴の詳細を確認してください。インテグレーションアクティビティー情報の表示 および インテグレーション履歴の表示 を参照してください。

### インテグレーション用のメモリーが十分にありません。

Fuse Online Operator のログで、インテグレーションのステータスが **not ready** である場合や、インテグレーションの Pod イベントがメモリー不足を報告する場合は、インテグレーションのメモリーおよび CPU 設定属性の調整 の説明どおりに、インテグレーションのデプロイメント設定の編集が必要な場合があります。

### OAuth を使用するアプリケーションへのコネクション

OAuth を使用するアプリケーションへのコネクションでは、アプリケーションのアクセストークンが期限切れであることを示すエラーメッセージが表示されることがあります。場合によっては、**403 - Access denied** というメッセージが表示されることがあります。メッセージの情報は、インテグレーションが接続しているアプリケーションによって異なります。この場合、アプリケーションへ再接続し、インテグレーションを再パブリッシュします。

1. 左側のパネルで **Integrations** をクリックします。
2. インテグレーションのリストで、実行が停止したインテグレーションのエントリーの右側にある **View** をクリックします。
3. インテグレーションの概要ページのフロービジュアライゼーションで、再接続するアプリケーションのアイコンをクリックします。
   これが API プロバイダーインテグレーションの場合、インテグレーションの概要ページでそのフローアイコンをクリックし、オペレーションのリストを表示します。次に、パスに障害のあるコネクションが含まれるオペレーションをクリックし、オペレーションのパスビジュアライゼーションで障害のあるコネクションをクリックします。
4. コネクションの詳細ページで、**Reconnect** をクリックします。
5. アプリケーションの OAuth ワークフローのプロンプトに応答します。
   Fuse Online は、そのアプリケーションへのアクセスが承認されたことを示すメッセージを表示します。アプリケーションによっては、これに数秒かかることがありますが、より長期間かかることもあります。
6. アプリケーションを再接続した後、インテグレーションを起動します。

再接続が適切に行われない場合は、以下を試行します。

1. Fuse Online をアプリケーションのクライアントとして再登録します。承認取得のための一般的な手順 を参照してください。
2. 新しいコネクションを作成します。
3. 古いコネクションを使用していた各インテグレーションを編集します。
   a. 古いコネクションを削除します。
   b. 新しいコネクションに置き換えます。
4. 更新された各インテグレーションをパブリッシュします。

## 8.7. インテグレーションの更新

インテグレーションの作成後、ステップを追加、編集、または削除するためにインテグレーションを更新する必要があります。

### 前提条件

Fuse Online 環境では、更新するインテグレーションのバージョンがある必要があります。

### 手順

1. Fuse Online パネルの左側にある **Integrations** をクリックします。
2. インテグレーションのリストで、更新するインテグレーションの **View** をクリックします。
3. インテグレーションの概要ページで、右上隅の **Edit Integration** をクリックします。
   これがシンプルなインテグレーションである場合、Fuse Online はインテグレーションのフローを表示します。これが API プロバイダーインテグレーションである場合、Fuse Online はオペレーションリストを表示します。特定のオペレーションのフローを更新するには、オペレーションの右にある **Edit flow** をクリックして、そのフローを表示します。
4. 必要に応じてフローを更新します。
   - ステップを追加するには、フロービジュアライゼーションで、ステップを追加する場所のプラス記号をクリックします。その後、追加するステップを表すカードをクリックします。
   - ステップを削除するには、フロービジュアライゼーションで削除するステップの  をクリックします。
   - ステップの設定を変更する場合は、フロービジュアライゼーションで更新するステップの **Configure** をクリックします。設定ページで、必要に応じてパラメーター設定を更新します。

## 8.8. インテグレーションのメモリーおよび CPU 設定属性の調整

インテグレーションのデプロイメント設定オブジェクトを編集することで、特定のインテグレーションの CPU およびメモリーにカスタム値を指定できます。インテグレーションにデフォルトの割り当てよりも多くのメモリーが必要な場合など、インテグレーションのメモリーおよび CPU 設定属性の調整が必要な場合があります。

### 前提条件

- Red Hat OpenShift **oc** クライアントツールがインストール済みであり、Fuse Online がインストールされている OCP クラスターに接続されている必要があります。
- クラスター管理者権限を持つユーザーから、設定するインテグレーションが含まれるプロジェクトの **admin** 権限が付与されている必要があります。

### 手順

1. Fuse Online インテグレーションが含まれる OpenShift プロジェクトの **admin** 権限を持つアカウントで OpenShift にログインします。以下に例を示します。
   **oc login -u admin -p admin**

2. Fuse Online インテグレーションが含まれるプロジェクトに切り替えます。以下に例を示します。
   **oc project my-fuse-online-project**

3. インテグレーションのデプロイメント設定オブジェクトを編集します。

   a. 以下のコマンドを実行すると、通常はエディターでリソースが開かれます。
   **oc edit deploymentconfig <i-integration-name>**

   たとえば、統合の名前が **my-integration** の場合は、次のコマンドを入力します。

   **oc edit deploymentconfig i-my-integration**

   b. 次の例に示すように、**spec.containers.resources** を設定して CPU とメモリーの値を指定することにより、設定を編集します。

```
spec:
  containers:
    resources:
      limits:
        cpu: 350m
    requests:
        memory: 350Mi
```

   c. 設定を保存します。

**結果**

変更を保存した後、インテグレーションの Pod が再起動し、新しい Pod が新しい値で実行されます。たとえば、**oc describe <intergration-pod-name>** コマンド (<intergration-pod-name> を **i-my-integration** などのインテグレーション Pod の名前に置き換えます) を実行すると、コマンドは新しい次のような値を返します。

```
resources:
  limits:
    cpu: 350m
  requests:
    cpu: 350m
    memory: 350Mi
```

これらの値は、新しいバージョンのインテグレーションをパブリッシュした後でも維持されます。

**その他のリソース**

OpenShift クラスター管理者は、**OpenShift Container Platform での Fuse Online のインストールと操作**の Fuse Online の設定に使用するカスタムリソース属性の説明 の説明どおりに、Fuse Online のカスタムリソースを更新し、すべてのインテグレーションの CPU およびメモリー属性のデフォルト値を設定することができます。

## 8.9. インテグレーションの削除

インテグレーションの削除後、Fuse Online はそのインテグレーションの履歴を保持します。削除されたインテグレーションのバージョンをインポートする場合、Fuse Online は削除されたインテグレーションの履歴をインポートされたインテグレーションと関連付けます。

**手順**

1. Fuse Online パネルの左側にある **Integrations** をクリックします。
2. インテグレーションのリストで、開始するインテグレーションのエントリーの右にある  をクリックし、**Delete** を選択します。
3. ポップアップの **OK** をクリックし、インテグレーションの削除を確認します。

## 8.10. インテグレーションの別の環境へのコピー

開発、ステージング、および実稼働環境全体でインテグレーションを実行するには、インテグレーションをエクスポートおよびインポートします。環境は、すべて単一の OpenShift クラスターに配置することができ、複数の OpenShift クラスター全体に分散することもできます。

ここで説明する手順では、Fuse Online コンソールでインテグレーションをエクスポートおよびインポートします。

Fuse Online をオンサイトで OpenShift Container Platform 上で実行している場合、特定のインテグレーションをエクスポートおよびインポートする必要がある CI/CD (Continuous Integration/Continuous Deployment) パイプラインが存在する場合があります。これを行う方法については、外部ツールを使用した CI/CD の Fuse Online インテグレーションのエクスポート/インポート を参照してください。

以下のトピックを参照してください。

- 「インテグレーションのコピー」
- 「インテグレーションのエクスポート」
- 「インテグレーションのインポート」

### 8.10.1. インテグレーションのコピー

各 Fuse Online インストールは、インテグレーションのエクスポート元となる環境です。インテグレーションのエクスポートによって、異なる Fuse Online 環境でインテグレーションを再作成するのに必要な情報が含まれる zip ファイルがダウンロードされます。

環境では、各インテグレーションは 1 つの **Draft** バージョンのみを持つことができます。

インテグレーションのインポート結果は以下によって異なります。

- インテグレーションが以前インポートされたかどうか。
- インテグレーションが使用するコネクションが以前インポートされたかどうか。

Fuse Online は、各インテグレーションと各コネクションの内部 ID を使用し、インポート先の環境にすでに存在するかどうかを判断します。インテグレーション名またはコネクション名を変更する場合、Fuse Online は異なる名前を持つ同じインテグレーションまたはコネクションとして認識します。

以下の表は、インレグレーションのインポートで可能な結果を表しています。

| インポートする環境 | インポートオペレーションによる動作 |
| --- | --- |
| インテグレーションはこれまでインポートされていない。 | インテグレーションを作成します。インテグレーションは **Draft** 状態です。 |

| インポートする環境 | インポートオペレーションによる動作 |
| --- | --- |
| インテグレーションは以前インポートされた。 | Fuse Online はインテグレーションを更新します。更新されたインテグレーションは **Draft** 状態です。このインテグレーションの **Draft** バージョンがある場合、これは失われます。 |
| インポートされたインテグレーションは、インポートオペレーションの前に環境に存在しなかったコネクションを使用する。 | Fuse Online は、シークレット以外が同じ設定のコネクションを作成します。新しいコネクションを確認する必要があります。コネクションが新しい環境に対して完全に設定されていない場合、不足している設定を追加する必要があります。たとえばこのような場合、Fuse Online 環境をコネクションがアクセスするアプリケーションのクライアントとして登録し、シークレットの設定を取得する必要がある場合があります。 |

## 8.10.2. インテグレーションのエクスポート

Fuse Online がインテグレーションをエクスポートする場合、zip ファイルをローカルの **Downloads** フォルダーにダウンロードします。この zip ファイルには、異なる Fuse Online 環境でインテグレーションを再作成するために必要な情報が含まれます。

インテグレーションのエクスポートは、インテグレーションをバックアップする方法でもあります。しかし、Fuse Online はインテグレーションのバージョンを維持するため、バックアップコピーを保持するのにインテグレーションのエクスポートは必要ありません。

**手順**

1. Fuse Online の左パネルで **Integrations** をクリックします。
2. インテグレーションのリストで、エクスポートするインテグレーションのエントリーを特定します。
3. エントリーの右側にある  をクリックし、**Export** を選択します。

**次のステップ**

インテグレーションを別の Fuse Online 環境にインポートするには、その環境を開いて、エクスポートした zip ファイルをインポートします。

## 8.10.3. インテグレーションのインポート

Fuse Online 環境で、別の Fuse Online 環境からエクスポートされたインテグレーションをインポートできます。インテグレーションのエクスポートは、インテグレーションをインポートするためにアップロードする zip ファイルをダウンロードします。

**前提条件**

- 別の Fuse Online 環境からエクスポートされたインテグレーションが含まれる zip ファイルが必要です。

**手順**

**手順**

1. インテグレーションをインポートする Fuse Online 環境を開きます。
2. 左側のパネルで **Integrations** をクリックします。
3. 右上の **Import** をクリックします。
4. エクスポートしたインテグレーション zip ファイルを 1 つまたは複数ドラッグアンドドロップするか、エクスポートしたインテグレーションが含まれる zip ファイルに移動して選択します。Fuse Online はファイルをインポートし、適切にインポートが完了したときにメッセージを表示します。
5. 左側のパネルで **Integrations** をクリックします。
6. インテグレーションのリストで、インポートしたインテグレーションのエントリーにある **View** をクリックします。
7. 設定が必要な通知がある場合は、インテグレーションの概要の右上にある **Edit integration** をクリックします。
8. 設定が必要な各コネクションに対して、以下を行います。
   a. **Configure** ボタンをクリックして詳細を表示します。
   b. 必要に応じてコネクションの詳細を入力または変更します。このページのすべてのフィールドが正しく、セキュリティー設定のみが必要になることがあります。
   c. **Next** をクリックします。
9. 左側のパネルで **Settings** をクリックします。
**Settings** ページには、OAuth プロトコルを使用するアプリケーションのエントリーが表示されます。
10. 設定が必要で、OAuth プロトコルを使用するアプリケーションにアクセスする各コネクションに対して、Fuse Online 環境をアプリケーションに登録します。手順はアプリケーションごとに異なります。以下の該当するトピックを参照してください。
    - Dropbox での登録
    - Google での登録
    - Jira での登録
    - REST API での登録
    - Salesforce での登録
    - SAP Concur への接続
    - Twitter への接続
11. 左側のパネルで **Connections** をクリックし、設定が必要なコネクションがないことを確認します。
12. 左側のパネルで **Integrations** をクリックします。インテグレーションのリストでは、インポートされたインテグレーションはエントリーの左上隅に緑の三角が表示されます。

13. インテグレーションのリストで、インポートしたインテグレーションのエントリーの右側にある  をクリックし、**Edit** を選択します。

14. 右上の **Save** をクリックするか、インポートしたインテグレーションの実行を開始する場合は **Publish** をクリックします。Fuse Online は、インテグレーションをドラフトとして保存するか、パブリッシュするかに関わらず、更新されたコネクションを使用するようインテグレーションを更新します。

# 第9章 FUSE ONLINE のカスタマイズ

Fuse Online は、一般的なアプリケーションやサービスへの接続に使用できるコネクターを多数提供します。一般的な方法でデータを処理する組み込みの手順も複数あります。しかし、Fuse Online が必要な機能を提供しない場合は、要件について開発者と話し合う必要があります。経験のある開発者は、以下を提供してインテグレーションのカスタマイズに協力できます。

- REST API クライアントのコネクターを作成するために Fuse Online が使用できる OpenAPI ドキュメント
  このスキーマを Fuse Online にアップロードすると、Fuse Online はスキーマにしたがってコネクターを作成します。その後、コネクターを使用してインテグレーションに追加できるコネクションを作成します。たとえば、多くのインターネットショップの Web サイトは、開発者が OpenAPI ドキュメントでキャプチャーできる REST API クライアントインターフェイスを提供します。

- REST API サービスを定義する OpenAPI ドキュメント。
  このスキーマを Fuse Online にアップロードします。Fuse Online は API サービスを利用可能にし、API 呼び出しの URL を提供します。これにより、API 呼び出しでインテグレーションの実行をトリガー できます。

- SOAP クライアントのコネクターの作成に Fuse Online が使用する WSDL ファイル。

- Fuse Online エクステンションを実装する **JAR** ファイル。エクステンションは以下の 1 つになります。

  - コネクション間のインテグレーションデータを操作する 1 つ以上のステップ。

  - アプリケーションまたはサービスのコネクター。

  - プロプライエタリー SQL データベースの JDBC ドライバーなどのライブラリーリソース。この **JAR** ファイルを Fuse Online にアップロードすると、Fuse Online はエクステンションによって提供されるカスタム機能を利用可能にします。

詳細は以下のトピックを参照してください。

- 「REST API クライアントコネクターの開発」
- 「API クライアントコネクターの追加および管理」
- 「Fuse Online エクステンションの開発」
- 「エクステンションの追加および管理」

## 9.1. REST API クライアントコネクターの開発

Fuse Online は、HTTP (Hypertext Transfer Protocol) をサポートする REST API (Representational State Transfer Application Programming Interface) のコネクターを作成できます。これには、接続する REST API を記述する有効な OpenAPI 3 (または 2) ドキュメントが Fuse Online に必要です。API サービスプロバイダーが OpenAPI ドキュメントを利用可能にしない場合、経験のある開発者が OpenAPI ドキュメントを作成する必要があります。

REST API コネクターを開発するための情報および手順は、以下を参照してください。

- 「REST API クライアントコネクターの要件」
- 「REST API クライアントコネクターの OpenAPI スキーマのガイドライン」

- 「クライアントクレデンシャルをパラメーターで提供」
- 「アクセストークンの自動更新」

### 9.1.1. REST API クライアントコネクターの要件

OpenAPI スキーマを Fuse Online にアップロードすると、REST API へのコネクターが利用可能になります。コネクターを選択すると REST API クライアントコネクションを作成できます。その後、新しいインテグレーションを作成し、REST API クライアントコネクションを追加するか、既存のインテグレーションを編集して REST API クライアントコネクションを追加します。

Fuse Online コネクターは OAuth 2.0、HTTP の BASIC 認証、および API キーをサポートします。REST API へのアクセスに TLS (Transport Layer Security) が必要な場合、API は認められた認証局 (CA) が発行する有効な証明書を使用する必要があります。

OAuth を使用する REST API には、クライアントコールバック URL を入力とする承認 URL が必要です。Fuse Online がコネクターを作成した後、コネクターを使用してコネクションを作成する前に、その URL にアクセスして Fuse Online 環境を REST API のクライアントとして登録する必要があります。これにより、Fuse Online 環境による REST API へのアクセスが承認されます。登録の一環として、Fuse Online コールバック URL を提供します。詳細は、Fuse Online のアプリケーションおよびサービスへの接続、Fuse Online を REST API クライアントとして登録 の説明を参照してください。

OAuth を使用する REST API では、Fuse Online は承認の取得に **Authorization Code** Grant フローのみをサポートします。コネクターを作成するためにアップロードする OpenAPI ドキュメントの **securityDefinitions** オブジェクトで、**flow** 属性を **accessCode** に設定する必要があります。例を以下に示します。

```
securityDefinitions:
  OauthSecurity:
    type: oauth2
    flow: accessCode
    authorizationUrl: 'https://oauth.simple.api/authorization'
    tokenUrl: 'https://oauth.simple.api/token'
```

**flow** を **implicit**、**password**、または **application** に設定しないでください。

REST API クライアントコネクターの OpenAPI スキーマは、循環 (Cyclic) スキーマ参照を持つことができません。たとえば、リクエストまたは応答ボディーを指定する JSON スキーマは、そのスキーマ自体を全体的に参照することはできず、任意数の中間スキーマを介してそれ自体を部分的に参照することもできません。

Fuse Online は、HTTP 2.0 プロトコルをサポートする REST API のコネクターを作成できません。

### 9.1.2. REST API クライアントコネクターの OpenAPI スキーマのガイドライン

Fuse Online が REST API クライアントコネクターを作成するとき、OpenAPI ドキュメントの各リソースオペレーションをコネクションアクションにマップします。アクション名とアクションの説明は、OpenAPI ドキュメントのドキュメントから提供されます。

OpenAPI ドキュメントが提供する詳細が多いほど、API への接続時に Fuse Online が提供するサポートも多くなります。たとえば、API 定義はリクエストおよび応答のデータタイプを宣言する必要はありません。タイプ宣言がない場合、Fuse Online は対応するコネクションアクションをタイプレス (タイプがない) として定義します。ただし、インテグレーションでは、タイプレスアクションを実行する API コネクションの直前および直後に、データマッピングステップを追加することはできません。

これに対処するには、OpenAPI ドキュメントにより多くの情報を追加します。API コネクションが実行するアクションにマップする OpenAPI リソースオペレーションを特定します。OpenAPI ドキュメントにて、各オペレーションのリクエストおよび応答タイプを指定する YAML または JSON スキーマがあることを確認します。

スキーマをアップロードした後、OpenAPI ドキュメントをベースとした API 設計のビジュアルエディターである API Designer で、スキーマを確認および編集する機会が与えられます。詳細の追加や更新の保存を行うことができ、Fuse Online は更新に対応する API クライアントコネクターを作成します。Fuse Online がクライアントコネクターを作成した後、OpenAPI ドキュメントを編集できなくなります。変更を実装するには、新しいクライアントコネクターを作成する必要があります。

API の OpenAPI ドキュメントが、**application/json** コンテンツタイプおよび **application/xml** コンテンツタイプのサポートを宣言する場合、コネクターは JSON 形式を使用します。OpenAPI ドキュメントが、**application/json** および **application/xml** の両方を定義する **consumes** または **produces** パラメーターを指定する場合、コネクターは JSON 形式を使用します。

### 9.1.3. クライアントクレデンシャルをパラメーターで提供

Fuse Online が OAuth2 アプリケーションへアクセスするための承認の取得を試みると、HTTP Basic 認証を使用してクライアントのクレデンシャルを提供します。必要な場合は、このデフォルトの動作を変更し、Fuse Online が HTTP Basic 認証を使用する代わりに、クライアントクレデンシャルをパラメーターとしてプロバイダーに渡すことができます。これは、OAuth アクセストークンの取得に使用される **tokenUrl** エイドポイントの使用に影響します。

**重要**

これは テクノロジープレビュー の機能です。

Fuse Online がクライアントクレデンシャルをパラメーターとして渡すよう指定するには、OpenAPI ドキュメントの **securityDefinitions** セクションで、設定が **true** の **x-authorize-using-parameters** ベンダーエクステンションを追加します。以下の例では、最後の行は **x-authorize-using-parameters** を指定しています。

```
securityDefinitions:
  concur_oauth2:
    type: 'oauth2'
    flow: 'accessCode'
    authorizationUrl: 'https://example.com/oauth/authorize'
    tokenUrl: 'https://example.com/oauth/token'
    scopes:
      LIST: Access List API
    x-authorize-using-parameters: true
```

**x-authorize-using-parameters** ベンダーエクステンションの設定は **true** または **false** です。

- **True** は、クライアントクレデンシャルがパラメーターであることを示しています。
- デフォルトは **false** で、Fuse Online は HTTP Basic 認証を使用してクライアントクレデンシャルを提供します。

### 9.1.4. アクセストークンの自動更新

アクセストークンに有効期限がある場合、そのトークンを使用してアプリケーションに接続する Fuse Online インテグレーションは、トークンの期限が切れると、正常に実行を停止します。新しいアクセストークンを取得するには、アプリケーションに再接続するか、アプリケーションで再登録する必要があ

ります。

必要な場合は、このデフォルトの動作を変更し、Fuse Online が以下の状況で新しいアクセストークンを自動的にリクエストできるようにします。

- 期限切れになった場合。
- 指定した HTTP 応答ステータスコードが受信された場合。

**重要**

これは テクノロジープレビュー の機能です。

前述の状況で Fuse Online が新しいアクセストークンを自動的に取得するよう指定するには、OpenAPI ドキュメントの **securityDefinitions** セクションで **x-refresh-token-retry-statuses** ベンダーエクステンションを追加します。このエクステンションの設定は、HTTP 応答ステータスコードを指定するコンマ区切りのリストです。アクセストークンの期限が切れたり、Fuse Online が OAuth2 プロバイダーから受信したメッセージにこれらの応答ステータスコードの 1 つがある場合、Fuse Online は自動的に新しいアクセストークンの取得を試みます。以下の例では、最後の行は **x-refresh-token-retry-statuses** を指定します。

```
securityDefinitions:
  concur_oauth2:
    type: 'oauth2'
    flow: 'accessCode'
    authorizationUrl: 'https://example.com/oauth/authorize'
    tokenUrl: 'https://example.com/oauth/token'
    scopes:
      LIST: Access List API
    x-refresh-token-retry-statuses: 401,402,403
```

**注記**

場合によっては API オペレーションが失敗し、その失敗によりアクセストークンが更新されます。この場合、新しいアクセストークンの取得に成功しても、API オペレーションが失敗します。つまり、Fuse Online は新しいアクセストークンを受信した後に、失敗した API オペレーションを再試行しません。

## 9.2. API クライアントコネクターの追加および管理

Fuse Online は以下の API クライアントコネクターを作成できます。

- OpenAPI ドキュメントからの REST API クライアントコネクター。OpenAPI ドキュメントの内容に関する情報は REST API クライアントコネクターの開発 を参照してください。
- WSDL ファイルからの SOAP API クライアントコネクター。

REST API クライアントコネクターを追加および管理するための情報および手順は以下を参照してください。

- 「REST API クライアントコネクターの作成」
- 「SOAP API クライアントコネクターの作成」

- 「新規コネクターを作成して API クライアントコネクターを更新」
- 「API クライアントコネクターの削除」

API クライアントコネクターの作成後、このコネクターを使用するには、Fuse Online のアプリケーションおよびサービスへの接続の API クライアントへの接続 の詳細を参照してください。

### 9.2.1. REST API クライアントコネクターの作成

OpenAPI ドキュメントをアップロードし、Fuse Online が REST API クライアントコネクターを作成できるようにします。

#### 前提条件

Fuse Online が作成するコネクターの OpenAPI ドキュメントが必要です。

#### 手順

1. Fuse Online のナビゲーションパネルで、**Customizations** > **API Client Connectors** をクリックします。すでに利用可能な API クライアントコネクターが表示されます。
2. **Create API Connector** をクリックします。
3. **Create API Connector** ページで、以下のいずれかを行います。
   - 点線のボックスをクリックし、アップロードする OpenAPI ファイルを選択します。
   - **Use a URL** を選択し、入力フィールドに OpenAPI ドキュメントの URL を貼り付けます。
4. **Next** をクリックします。無効なコンテンツや不足しているコンテンツがある場合、必要な修正に関する情報が Fuse Online に表示されます。アップロードする別の OpenAPI ファイルを選択するか、**Cancel** をクリックします。OpenAPI ファイルを訂正し、更新されたファイルをアップロードします。
   スキーマが有効な場合、Fuse Online はコネクターが提供する操作の概要を表示します。これには、操作の定義に関するエラーおよび警告が含まれる可能性があります。
5. 概要が適切であれば **Next** をクリックします。
   OpenAPI ドキュメントを訂正する場合は、**Review/Edit** をクリックして API Designer エディターを開きます。必要に応じてスキーマを更新します。API エディターの使用に関する詳細は、API Designer を使用した API 定義の設計および開発 を参照してください。終了したら、変更を **保存** し、更新を新しい API クライアントコネクターに反映します。**Next** をクリックして API クライアントコネクターの作成を続行します。

   OpenAPI ドキュメントの URL を提供する場合、Fuse Online はドキュメントをアップロードできても、開いて編集できないことがあります。通常、これはファイルのホストの設定が原因です。スキーマを開いて編集するには、ファイルのホストに以下が必要です。

   - **https** の URL。(**http** の URL は動作しません)
   - 有効な CORS。
6. API のセキュリティー要件を示します。Fuse Online は OpenAPI 定義を読み取り、API のセキュリティー要件を満たすためにコネクターの設定に必要な情報を判断します。Fuse Online は以下のいずれかを表示できます。
   a. **No Security** (セキュリティーなし)

b. **HTTP Basic Authorization** (HTTP Basic 認証) - API サービスが HTTP Basic 認証を使用する場合は、このチェックボックスを選択します。後で、このコネクターを使用してコネクションを作成するときに、Fuse Online はユーザー名とパスワードの入力を要求します。

c. **OAuth 2.0** – Fuse Online は以下を入力するよう要求します。

i. **Authorization URL** は、Fuse Online を API のクライアントとして登録する場所です。登録によって Fuse Online の API へのアクセスが承認されます。Fuse Online のアプリケーションおよびサービスへの接続の Fuse Online を REST API クライアントとして登録 を参照してください。OpenAPI ドキュメントまたは API の他のドキュメントはこの URL を指定する必要があります。指定しない場合、サービスプロバイダーに連絡してこの URL を取得する必要があります。

ii. OAuth 承認には、**アクセストークン URL** が必要です。この場合も、OpenAPI ドキュメントまたは API の他のドキュメントがこの URL を提供する必要があります。提供しない場合、サービスプロバイダーに連絡する必要があります。

d. **API Key** - API サービスに API キーが必要な場合、Fuse Online はコネクターの作成に必要な情報を要求します。プロンプトは OpenAPI 定義に基づきます。たとえば、API キーがメッセージヘッダーまたはクエリーパラメーターにあるかを示す必要がある可能性があります。OpenAPI 定義が API キーのセキュリティーと他のセキュリティータイプを指定する場合、チェックボックスを選択し、このコネクターを基にしてコネクションに API キーセキュリティーを使用することを示します。後で、このコネクターを使用してコネクションを作成するときに、Fuse Online は API キーの値を入力するよう要求します。

7. **Next** をクリックします。Fuse Online は OpenAPI ドキュメントによって示されるコネクターの名前、説明、ホスト、およびベース URL を表示します。このコネクターから作成するコネクションの場合は以下を行います。

- Fuse Online は、コネクションのエンドポイントを定義するため、ホストおよびベース URL 値を連結します。たとえば、ホストが **https://example.com** で、ベース URL が **/api/v1** の場合、コネクションエンドポイントは **https://example.com/api/v1** になります。

- Fuse Online は OpenAPI ドキュメントをデータマッピングステップに適用します。OpenAPI ドキュメントが複数のスキーマをサポートする場合、Fuse Online は TLS (HTTPS) スキーマを使用します。

8. コネクターの詳細を確認し、必要に応じてコネクターのアイコンをアップロードします。アイコンをアップロードしない場合、Fuse Online はアイコンを生成します。後でアイコンをアップロードできます。Fuse Online でインテグレーションのフローが表示されるときに、コネクターから作成されたコネクションを表すコネクターのアイコンが表示されます。

9. OpenAPI ファイルから取得した値を上書きするには、変更するフィールド値を編集します。

**重要**

Fuse Online がコネクターを作成した後、**これを変更することはできません**。変更を反映にするには、Fuse Online が新しいコネクターを作成できるよう、更新された OpenAPI ドキュメントをアップロードしたり、同じスキーマをアップロードして API エディターで変更する必要があります。その後、新しい API クライアントコネクターを作成するプロセスを続行します。

10. コネクターの詳細が適切であれば、**Save** をクリックします。Fuse Online API Client Connectors のリストに新しいコネクターが表示されます。

### 次のステップ

新しい API コネクターの使用に関する詳細は、Fuse Online のアプリケーションおよびサービスへの接続の REST API への接続 を参照してください。

### 9.2.2. SOAP API クライアントコネクターの作成

WSDL ファイルをアップロードし、Fuse Online で SOAP API クライアントコネクターを作成できるようにします。

インラインと外部 (WSDL URL) の両方が、一意の名前空間を持つ複数のスキーマをサポートします。

#### 前提条件

Fuse Online で作成する SOAP クライアントコネクターの WSDL ファイルが必要です。

#### 手順

1. Fuse Online のナビゲーションパネルで、**Customizations** > **API Client Connectors** をクリックします。すでに利用可能な API クライアントコネクターが表示されます。
2. **Create API Connector** をクリックします。
3. **Create API Connector** ページで、以下のいずれかを行います。
   - 点線のボックスをクリックし、アップロードする WSDL (**.wsdl**) ファイルを選択します。(**ファイルのアップロード** フォームを使用して) コネクターに直接インポートする WSDL ファイルで参照されるディスクベースの外部スキーマはサポートされていないことに注意してください。アップロードされた WSDL ファイルは、インラインスキーマを使用する **必要があります**。
   - **Use a URL** を選択し、入力フィールドに WSDL (**.wsdl**) ファイルの URL を貼り付けます。URL ベースの WSDL は、WSDL でホストされる外部スキーマをサポートすることに注意してください。また、URL ベースの WSDL のみが、WSDL のベースパスからの相対 URL に基づく外部スキーマをサポートします。WSDL URL は、実行時に解析と検証のために SOAP コネクターで使用できる **必要** があります。したがって、WSDL とスキーマが永続的な URL でホストされていることを確認してください。
4. **Next** をクリックします。
5. **Specify service and port** ページで、サービスおよびポートを確認します。
6. **Next** をクリックします。無効なコンテンツや不足しているコンテンツがある場合、必要な修正に関する情報が Fuse Online に表示されます。アップロードする別の WSDL ファイルを選択するか、**Cancel** をクリックします。WSDL ファイルを修正し、更新したファイルをアップロードします。スキーマが有効な場合、Fuse Online には API 定義 (名前および説明) の概要と、インポートされた要素のリスト (操作の数など) が表示されます。
7. **Next** をクリックします。
8. WSDL エンドポイントを呼び出すときに使用するセキュリティー要件を示します。Fuse Online は API 定義を読み取り、API のセキュリティー要件を満たすためにコネクターの設定に必要な情報を判断します。Fuse Online は以下のいずれかを表示できます。
   - **なし** (セキュリティーなし)
   - **HTTP Basic Authorization** (HTTP Basic 認証) - API サービスが HTTP Basic 認証を使用する場合は、このチェックボックスを選択します。後で、このコネクターを使用してコネクションを作成するときに、Fuse Online はユーザー名とパスワードの入力を要求します。

- **WS-Security Username Token** – Fuse Online は以下の情報の入力を求めます。

  a. **Timestamp** - Fuse Online でタイムスタンプを WS-Security ヘッダーに追加する場合は、このオプションを選択します。

  b. **Password Type** - **Digest**、**Text**、または **None** を選択します。
  **Text** または **Digest** を選択した場合:

  - **username** と **password** を指定します。
  - Fuse Online で Nonce 要素を WS-Security Username Token ヘッダーに追加する場合は、**Username Token Nonce** を選択します。
  - Fuse Online で "Created" タイムスタンプ要素を WS-Security Username Token ヘッダーに追加する場合は、**Username Token Created** を選択します。

9. **Next** をクリックします。コネクターの名前、説明、および WSDL エンドポイントアドレスが Fuse Online に表示されます。

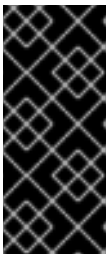
   a. 必要に応じて、コネクターのアイコンをアップロードします。後でアイコンをアップロードすることもできます。
   **注記:** 本リリースでは、アイコンをアップロードしないと、Fuse Online でアイコンが生成**されません**。

   Fuse Online でインテグレーションのフローが表示されるときに、コネクターから作成されたコネクションを表すコネクターのアイコンが表示されます。

   b. **Name** には、このコネクションを別のコネクションと区別するために使用する名前を入力します。

   c. 必要に応じて、**Description** にこのコネクションに関する便利な情報を入力します。

10. コネクターの詳細を確認し、WSDL ファイルから取得した値をオーバーライドします。フィールド値を変更します。



**重要**

Fuse Online がコネクターを作成した後、**これを変更することはできません**。変更を反映にするには、Fuse Online が新しいコネクターを作成できるよう、更新された OpenAPI ドキュメントをアップロードしたり、同じスキーマをアップロードして API エディターで変更する必要があります。その後、新しい API クライアントコネクターを作成するプロセスを続行します。

11. コネクターの詳細が適切であれば、**Save** をクリックします。Fuse Online API Client Connectors のリストに新しいコネクターが表示されます。

### 次のステップ

新しい API コネクターの使用に関する詳細は、Fuse Online のアプリケーションおよびサービスへの接続の REST API への接続 を参照してください。

## 9.2.3. 新規コネクターを作成して API クライアントコネクターを更新

API クライアントコネクターを作成した OpenAPI ドキュメントまたは WSDL ファイルに更新があり、API クライアントコネクターがそれらの更新を使用するようにするには、新しい API クライアントコネクターを作成する必要があります。API クライアントコネクターを直接更新することはできません。新

しい API クライアントコネクターの作成後に、そのコネクターを使用して新しいコネクションを作成し、古いコネクターから作成されたコネクションを使用する各インテグレーションを編集します。

**前提条件**

以下のいずれかを行う準備をする必要があります。

- REST API クライアントコネクターの場合:
  - 更新された OpenAPI ドキュメントをアップロードします。
  - 古いスキーマを再度アップロードし、API Designer で更新します。
- SOAP API クライアントコネクターの場合: 更新された WSDL ファイルをアップロードします。

**手順**

1. 更新された OpenAPI ドキュメントまたは WSDL ファイルを基にして新しい API クライアントコネクターを作成します。古いコネクターと新しいコネクターを区別しやすくするため、コネクター名またはコネクターの説明にバージョン番号を指定することもできます。
   REST API クライアントコネクターの開発 を参照してください。
2. 新しいコネクターから新しいコネクションを作成します。ここでも、古いコネクターから作成されたコネクションと、新しいコネクターから作成されたコネクションを簡単に区別できるようにすることができます。コネクション名またはコネクションの説明のバージョン番号を使用すると便利です。
3. 古いコネクションを削除し、新しいコネクションを追加して、古いコネクターから作成されたコネクションを使用する各インテグレーションを編集します。
4. 更新された各インテグレーションをパブリッシュします。
5. 必須ではありませんが、古いコネクターおよびコネクションを削除することが推奨されます。

### 9.2.4. API クライアントコネクターの削除

コネクターから作成されたコネクションがある場合や、このコネクションがインテグレーションで使用される場合は、コネクターを削除することはできません。API クライアントコネクターの削除後、そのコネクターから作成されたコネクションを使用することはできません。

**手順**

1. 左側のパネルで **Customizations** > **API Client Connectors** をクリックします。
2. 削除するコネクター名の右側にある **Delete** をクリックします。
3. コネクターを削除する場合は、確認ポップアップの **Delete** をクリックします。

## 9.3. FUSE ONLINE エクステンションの開発

Fuse Online がインテグレーションの作成に必要な機能を提供しない場合、開発者が必要な動作を提供するエクステンションを作成することができます。Syndesis エクステンションリポジトリー https://github.com/syndesisio/syndesis-extensions にエクステンションの例が含まれています。

ビジネスインテグレーターは、エクステンションを作成する開発者と要件を共有します。開発者は、エクステンションが含まれる **.jar** ファイルを提供します。ビジネスインテグレーターは Fuse Online に

**.jar** ファイルをアップロードして、カスタムコネクター、カスタムステップ、またはライブラリーリソースを Fuse Online で使用できるようにします。

Red Hat Developer Studio への Fuse Tooling プラグインは、ステップのエクステンションまたはコネクターのエクステンションの開発に役立つウィザードを提供します。Developer Studio またはその他の IDE で、ステップエクステンションまたはコネクターエクステンションを開発するかどうかは、個人的に選択できます。Developer Studio プラグインの使用に関する詳細は、Fuse Online インテグレーション用エクステンションの開発 を参照してください。

選択した IDE でエクステンションを開発するための手順の概要、要件の説明、および追加の例については以下を参照してください。

- 「エクステンション開発の一般的な手順」
- 「エクステンションの種類の説明」
- 「エクステンションコンテンツおよび構造の概要」
- 「エクステンション定義の JSON ファイルの要件」
- 「ユーザーインターフェイスプロパティーの説明」
- 「エクステンションをサポートする Maven プラグインの説明」
- 「エクステンションでデータシェイプを指定する方法」
- 「ステップエクステンションの開発例」
- 「コネクターエクステンションの開発例」
- 「ライブラリーエクステンションの開発方法」
- 「JDBC ドライバーライブラリーエクステンションの作成」

### 9.3.1. エクステンション開発の一般的な手順

エクステンションの開発を開始する前に、実行する必要のあるタスクについて理解しておく必要があります。

**前提条件**

- Maven を理解している必要があります。
- コネクターを提供するエクステンションを開発したり、コネクション間のデータを操作するインテグレーションステップを提供するエクステンションを開発する場合は、Camel を理解している必要があります。
- プログラミングの経験が必要です。

**注意**

インテグレーション Pod は、フラットなクラスパスを持つ Java プロセスで実行されます。バージョンの競合を避けるため、エクステンションが使用する依存関係が、以下のすべてのソースからインポートされた BOM (Bill of Materials) に対応することを確認してください。

- **org.springframework.boot:spring-boot-dependencies:$SPRING_BOOT_VERSION**
- **org.apache.camel:camel-spring-boot-dependencies:$CAMEL_VERSION**
- **io.syndesis.integration:integration-bom:$syndesis_VERSION**

インポートされた BOM の一部ではない追加の依存関係がある場合、以下を行う必要があります。

- **lib** ディレクトリーにあるエクステンション JAR ファイルでパッケージ化する必要があります。
- エクステンションの JSON 記述子ファイルの **dependencies** プロパティーから省略します。

**手順**

1. 拡張された機能の動作の理解します。機能要件を理解するために、ビジネスの関係者と話し合います。
2. ステップエクステンション、コネクターエクステンション、またはライブラリーエクステンションを開発する必要があるかどうかを判断します。
3. エクステンションを開発する Maven プロジェクトを設定します。
4. ステップエクステンションを開発する場合は以下を行います。
    a. Camel ルートとして実装するか、または Syndesis **Step** API を使用して実装するかを決定します。Syndesis API に関する情報は http://javadoc.io/doc/io.syndesis.extension/extension-api を参照してください。
    b. Camel ルートとしてエクステンションを実装する場合は、XML フラグメント、**RouteBuilder** クラス、または Bean を実装するかどうかを決定します。
    c. Maven プロジェクトで、**schemaVersion**、エクステンション **name**、**extensionId** などの必要なメタデータを指定します。
5. 機能を実装するクラスを作成します。
6. 依存関係をプロジェクトの **pom.xml** ファイルに追加します。
7. コネクターおよびライブラリーエクステンションと、XML に実装するステップエクステンションに、エクステンションを定義する JSON ファイルを作成します。
Java に実装するステップエクステンションでは、Maven プロジェクトで対応するデータ構造の値を指定するときに Maven は JSON エクステンション定義ファイルを生成できます。
8. Maven を実行してエクステンションを構築し、エクステンションの JAR ファイルを作成します。
9. JAR ファイルを Fuse Online 開発環境にアップロードして、エクステンションをテストします。
10. エクステンションをパッケージ化する JAR ファイルを Fuse Online 本番環境にアップロードす

るビジネス関係者に、そのファイルを提供します。JAR ファイルを提供するときに、Fuse Online の Web インターフェイスに表示される情報以外の設定について、ビジネス関係者に説明します。

### 9.3.2. エクステンションの種類の説明

エクステンションは以下のいずれかを定義します。

- コネクション間のインテグレーションデータを操作する 1 つ以上のカスタムステップ。各カスタムステップは 1 つのアクションを実行します。これは、ステップエクステンションです。
- インテグレーションランタイムが使用するライブラリーリソース。たとえば、ライブラリーエクステンションは、Oracle などのプロプライエタリー SQL データベースに接続するための JDBC ドライバーを提供できます。
- 統合する特定のアプリケーションまたはサービスへのコネクションを作成するための単一のカスタムコネクター。これは、コネクターエクステンションです。

**注記**

Fuse Online は OpenAPI ドキュメントを使用して REST API クライアントのコネクターを作成できます。REST API クライアントコネクターの開発 を参照してください。

ビジネスインテグレーターは、エクステンションを作成する開発者と要件を共有します。開発者は、エクステンションが含まれる **.jar** ファイルを提供します。ビジネスインテグレーターは Fuse Online で **.jar** ファイルをアップロードし、カスタムコネクター、カスタムステップ、または Fuse Online 内で使用できるライブラリーリソースを作成します。

Fuse Online にアップロードするエクステンション **.jar** ファイルには、常に 1 つのエクステンションが含まれます。

コネクション間のデータを操作するステップを提供するエクステンションのアップロード例および使用例については、 AMQ から REST API へのサンプルインテグレーションのチュートリアル を参照してください。

### 9.3.3. エクステンションコンテンツおよび構造の概要

エクステンションとは、**.jar** ファイルにパッケージ化されるクラス、依存関係、およびリソースのコレクションです。

Fuse Online は Spring Boot を使用してエクステンションをロードします。そのため、Spring Boot の実行可能な JAR 形式に従い、エクステンションをパッケージ化する必要があります。たとえば、**ZipEntry.STORED()** を使用して、ネストされた JAR ファイルを保存するようにします。

エクステンションをパッケージ化する **.jar** ファイルの構造は次のとおりです。

```
extension.jar
|
+- META-INF
| |
| +- syndesis
|   |
|   +- syndesis-extension-definition.json (1)
|
```

```
+- mycompany
| |
| +-project
|   |
|   +-YourClasses.class (2)
|
+- lib (3)
  |
  +-dependency1.jar
  |
  +-dependency2.jar
```

(1) エクステンションを定義するデータ構造を指定する JSON スキーマファイル。これは、エクステンション定義の JSON ファイルです。

(2) エクステンションが提供する動作を実装する Java クラス。

(3) カスタム機能をビルドおよび実行するために必要な追加の依存関係。

### 9.3.4. エクステンション定義の JSON ファイルの要件

各エクステンションには、名前、説明、サポートされるアクション、依存関係などのデータ構造の値を指定してエクステンションを定義する **.json** ファイルが必要です。以下の表は、Maven がエクステンション定義 JSON ファイルを生成するかどうかと、必要になるデータ構造をエクステンションタイプ別に示しています。

| エクステンションタイプ | Maven によるエクステンション定義の生成 | 必要なデータ構造 |
|---|---|---|
| Java のステップエクステンション | 可能 | **schemaVersion**<br>**name**<br>**description**<br>**version**<br>**extensionId**<br>**extensionType**<br>**actions**<br>**dependencies** * |
| XML のステップエクステンション | 不可能 | **schemaVersion**<br>**name**<br>**description**<br>**version**<br>**extensionId**<br>**extensionType**<br>**actions**<br>**dependencies** * |

| エクステンションタイプ | Maven によるエクステンション定義の生成 | 必要なデータ構造 |
|---|---|---|
| コネクターエクステンション | 不可能 | **schemaVersion**<br>**name**<br>**description**<br>**version**<br>**extensionId**<br>**extensionType**<br>**properties**<br>**actions**<br>**dependencies** *<br>**componentScheme**<br>**connectorCustomizers**<br>**connectorFactory** |
| ライブラリーエクステンション | 不可能 | **schemaVersion**<br>**name**<br>**description**<br>**version**<br>**extensionId**<br>**extensionType**<br>**dependencies** * |

* **dependencies** の指定は必ずしも必要ではありませんが、ほとんどの場合で指定する必要がある依存関係があります。

通常、エクステンション定義ファイルのレイアウトは次のようになります。

```
{
  "schemaVersion": "v1",
  "name": "",
  "description": "",
  "version": "",
  "extensionId": "",
  "extensionType": "",
  "properties": {
  },
  "actions": [
  ],
  "dependencies": [
  ],
}
```

- **schemaVersion** は、エクステンション定義スキーマのバージョンを定義します。Syndesis は内部的に **schemaVersion** を使用して、エクステンション定義を内部モデルにマップする方法を決定します。これにより、古いバージョンの Syndesis に対して開発されたエクステンションを、より新しいバージョンの Syndesis にデプロイすることが可能になります。

- **name** は、エクステンションの名前です。Fuse Online にエクステンションをアップロードするときに、この名前が表示されます。

- **description** は、指定する便利な情報です。Fuse Online はこの値を操作しません。

- **version** は、エクステンションへの更新を区別するのに便利です。Fuse Online はこの値を操作しません。
- **extensionId** は、エクステンションの一意の ID を定義します。これは、少なくとも Syndesis 環境全体で一意である必要があります。
- **extensionType** は、エクステンションが提供するものを Syndesis に示します。Syndesis バージョン 1.3 時点で、以下のエクステンションタイプがサポートされます。
  - **Steps**
  - **コネクター**
  - **Libraries**
- コネクターエクステンションのトップレベルで **properties** が必要です。このオブジェクトは、Fuse Online ユーザーがコネクターを選択してコネクションを作成する時に Fuse Online で何を表示するかを制御します。この **properties** オブジェクトには、コネクションを作成するための各フォームコントロールのプロパティーセットが含まれます。以下は例になります。

```
"myControlName": {
  "deprecated": true|false,
  "description": "",
  "displayName": "",
  "group": "",
  "kind": "",
  "label": "",
  "required": true|false,
  "secret": true|false,
  "javaType": "",
  "type": "",
  "defaultValue": "",
  "enum": {
  }
}
```

コネクターエクステンションでは、ネストされた **properties** オブジェクトはコネクションアクションを設定するための HTML フォームコントロールを定義します。ステップエクステンションでは、**actions** オブジェクトに **properties** オブジェクトが含まれます。**properties** オブジェクトは、ステップを設定するための各フォームコントロールのプロパティーセットを定義します。ユーザーインターフェイスプロパティーの説明 も参照してください。

- **actions** はコネクターが実行できるオペレーションや、コネクション間のステップが実行できるオペレーションを定義します。コネクターおよびステップエクステンションのみが指定するアクションを使用します。アクションの指定形式は以下のようになります。

```
{
   "id": "",
   "name": "",
   "description": "",
   "actionType": "step|connector",
   "descriptor": {
   }
}
```

- **id** は、アクションの一意の ID です。これは、少なくとも Syndesis 環境内で一意である必要があります。
- **name** は、Fuse Online に表示されるアクション名です。インテグレーターは、この値をコネクションアクションの名前またはコネクション間のインテグレーションデータを操作するステップの名前として解釈します。
- **description** は、Fuse Online に表示されるアクションの説明です。インテグレーターがアクションの動作を理解するよう、このフィールドを使用します。
- **actionType** は、アクションがコネクションまたはコネクション間のステップによって実行されるかどうかを示します。
- **descriptor** は、**kind**、**entrypoint**、**inputDataType**、**outputDatatype** などのネストされた属性を指定します。

- **dependencies** は、このエクステンションが必要とする Fuse Online が提供するリソースを定義します。
  以下のように依存関係を定義します。

```
{
  "type": "MAVEN",
  "id"   : "org.apache.camel:camel-telegram:jar:2.21.0"
}
```

  - **type** は依存関係のタイプを示します。**MAVEN** を指定します。(他のタイプは今後サポートされる予定です)
  - **id** は、Maven 依存関係 (Maven GAV) の ID です。

### 9.3.5. ユーザーインターフェイスプロパティーの説明

コネクターエクステンションおよびステップエクステンションで、エクステンション定義 JSON ファイルまたは Java クラスファイルにユーザーインターフェイスプロパティーを指定します。これらのプロパティーの設定は、Fuse Online ユーザーがコネクションの作成、コネクションアクションの設定、またはエクステンションによって提供されるステップの設定を行うときに Fuse Online が表示する HTML フォームコントロールを定義します。

Fuse Online コンソールのエクステンションのユーザーインターフェイスに表示する各フォームコントロールのプロパティーを指定する必要があります。各フォームコントロールで、一部またはすべてのプロパティーを任意の順序で指定します。

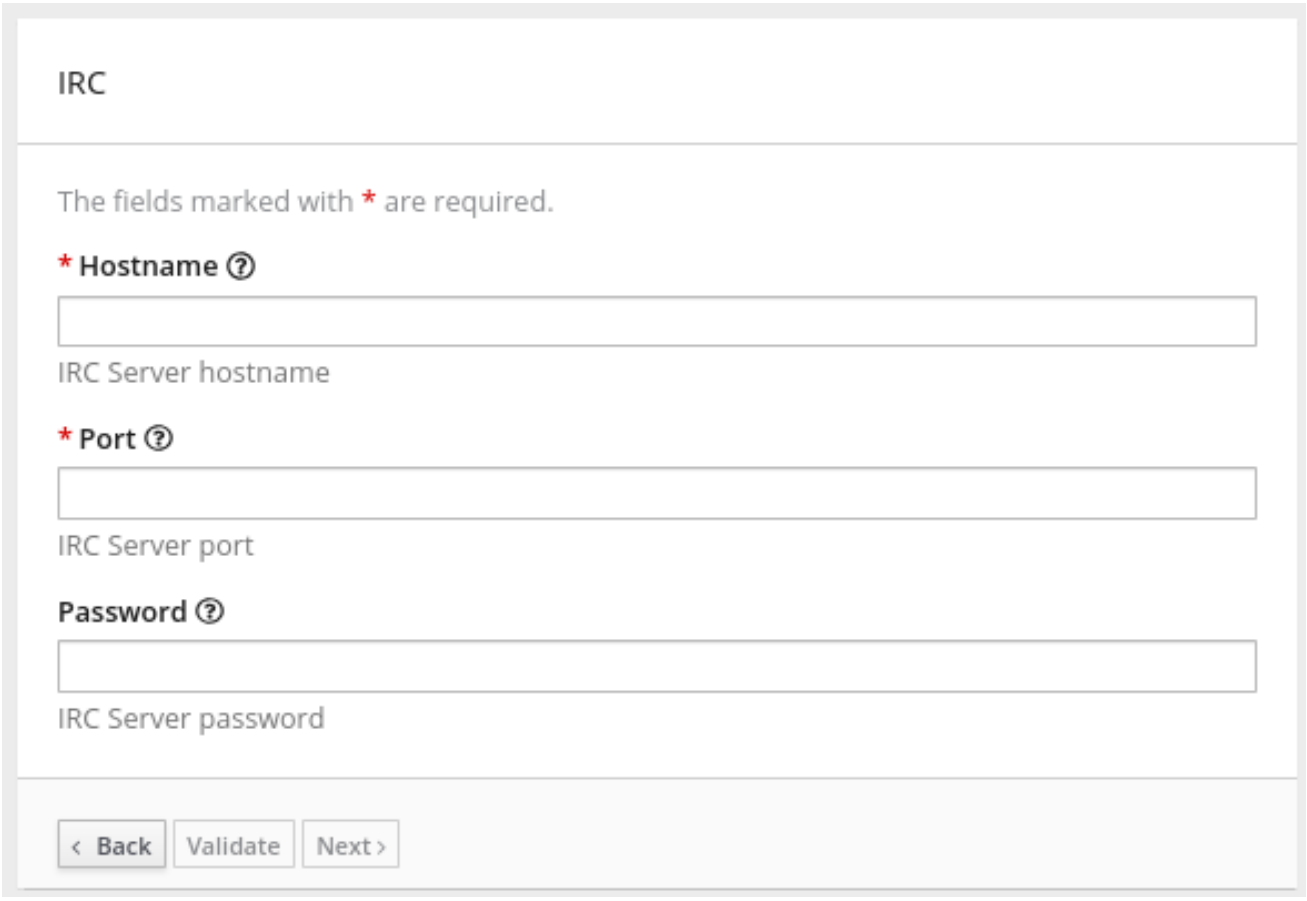
#### ユーザーインターフェイスプロパティーの指定例

IRC コネクターの一部である JSON ファイルでは、トップレベルの **properties** オブジェクトは、Fuse Online ユーザーがコネクションを作成するための IRC コネクターを選択した後に HTML フォームコントロールを定義します。3 つのフォームコントロールには、**hostname**、**password**、および **port** の 3 つのセットのプロパティー定義があります。

```
"properties": {
  "hostname": {
    "description": "IRC Server hostname",
    "displayName": "Hostname",
    "labelHint": "Hostname of the IRC server to connect to",
    "order": "1",
    "required": true,
```

```
    "secret": false,
    "type": "string"
   },
   "password": {
    "description": "IRC Server password",
    "displayName": "Password",
    "labelHint": "Required if IRC server requires it to join",
    "order": "3",
    "required": false,
    "secret": true,
    "type": "string"
   },
   "port": {
    "description": "IRC Server port",
    "displayName": "Port",
    "labelHint": "Port of the IRC server to connect to",
    "order": "2",
    "required": true,
    "secret": false,
    "tags": [],
    "type": "int"
   }
  },
```

これらのプロパティー指定を基にして、Fuse Online ユーザーが IRC コネクターを選択すると、Fuse Online に以下のダイアログが表示されます。ユーザーが 2 つの必須フィールドに値を入力し、**Next** をクリックすると、Fuse Online は Fuse Online ユーザーが入力する値で IRC コネクションを作成します。




**エクステンション定義 JSON ファイルの properties オブジェクト**

コネクターエクステンションでは以下が行われます。

- トップレベルの **properties** オブジェクトが必要です。このオブジェクトは、Fuse Online ユーザーがコネクターを選択してコネクションを作成する時に Fuse Online で何を表示するかを制御します。この **properties** オブジェクトには、コネクションを作成するための各フォームコントロールのプロパティーセットが含まれます。

- **action** オブジェクトでは、アクションごとに **properties** オブジェクトがあります。これらの **properties** オブジェクトごとに、そのアクションを設定するための各フォームコントロールのプロパティーセットがあります。

ステップエクステンションでは、**actions** オブジェクトに **properties** オブジェクトが含まれます。**properties** オブジェクトは、ステップを設定するための各フォームコントロールのプロパティーセットを定義します。JSON 階層は、以下のようになります。

```
"actions": [
  {
    ...

    "propertyDefinitionSteps": [
      {
        ...

        "properties":
          {
            "control-ONE": {
              "type": "string",
              "displayName": "Topic Name",
              "order": "2",
              ...,
              }

            "control-TWO": {
              "type": "boolean",
              "displayName": "Urgent",
              "order": "3",
              ...
              }

            "control-THREE": {
              "type": "textarea",
              "displayName": "Comment",
              "order": "1",
              ...,
              }
} } ]
```

### Java ファイルのユーザーインターフェイスプロパティー

Java ファイルでユーザーインターフェイスのフォームコントロールを定義するには、コネクション、アクション、またはステップのユーザー設定を定義する各クラスファイルに **io.syndesis.extension.api.annotations.ConfigurationProperty** をインポートします。Fuse Online コンソールが表示する各フォームコントロールに対して、**@ConfigurationProperty** アノテーションと後続のプロパティーリストを指定します。指定できるプロパティーに関する詳細は、このセクションの最後にあるユーザーインターフェイスプロパティーの参照テーブルを参照してください。

以下のコードは、単一のフォームコントロールのプロパティー定義を表しています。このコードは、**RouteBuilder** で Camel ルートを開発する例になります。

```
public class LogAction extends RouteBuilder {
    @ConfigurationProperty(
        name = "prefix",
        description = "The Log body prefix message",
        displayName = "Log Prefix",
        type = "string")
```

以下のコードは、2 つのコントロールのプロパティー定義を表しています。このコードは、Syndesis Step API をした例です。

```
@Action(id = "split", name = "Split", description = "Split your exchange")
public class SplitAction implements Step {

    @ConfigurationProperty(
        name = "language",
        displayName = "Language",
        description = "The language used for the expression")
    private String language;

    @ConfigurationProperty(
        name = "expression",
        displayName = "Expression",
        description = "The expression used to split the exchange
     private String language;
```

### コントロールフォーム入力ファイルの説明

各 HTML フォームコントロールのプロパティーセットでは、**type** プロパティーが Fuse Online が表示するフォームコントロールの入力タイプを定義します。HTML フォームの入力タイプに関する詳細は、https://www.w3schools.com/html/html_form_input_types.asp を参照してください。

以下の表に、Fuse Online フォームコントロールの可能な入力タイプを示します。制御のプロパティーのセットで、不明な **type** の値を指定すると、Fuse Online は 1 行のテキストを許可する入力フィールドを表示します。デフォルトは **"type": "text"** です。

| type プロパティーの値 | HTML | Fuse Online が表示するもの |
|---|---|---|
| **boolean** | **<input type="checkbox">** | ユーザーが選択可能または不可能なチェックボックス。 |

| type プロパティーの値 | HTML | Fuse Online が表示するもの |
|---|---|---|
| **duration** | Fuse Online ユーザーが時間の単位を選択できるようにするカスタム制御。ミリ秒、秒、分、時、または日を選択できます。また、ユーザーは番号を入力し、Fuse Online はミリ秒数を返します。例:<br>**"properties": {**<br>**"period": {**<br>**"type": "duration"**<br>**"defaultValue": 60000,**<br>**"description": "Period",**<br>**"displayName": "Period",**<br>**"labelHint": "Delay between integration executions.",**<br>**"required": true,**<br>**"secret": false,**<br>**}**<br>**}** | |
| **hidden** | **<input type="hidden">** | このフィールドは Fuse Online コンソールに表示されません。他のプロパティーを使用して、テキストデータなどのこのフィールドに関連するデータを指定できます。Fuse Online ユーザーはこのデータを表示または変更できませんが、ユーザーが Fuse Online ページの **View Source** を選択すると、ソース表示に非表示のフィールドが表示されます。そのため、セキュリティーの目的で非表示フィールドを使用しないでください。 |
| **int**、**integer**、**long**、**number** | **<input type="number">** | 数字を許可する入力フィールドです。 |
| **password** | **<input type="password">** | ユーザーが入力する文字を通常はアスタリスクでマスクする入力フィールド。 |
| **select** | **<select>** 要素の例:<br>**<select name="targets">**<br>**<option value="queue">Queue</option>**<br>**<option value="topic">Topic</option>**<br>**</select>** | フォームコントロールの **enum** プロパティーで指定するラベルと値のペアのエントリーが含まれるドロップダウンメニュー。 |
| **text**、**string**、または未知の値 | **<input type="text">`** | 1 行のテキストを許可する入力フィールド。 |
| **textarea** | **<input type="textarea"** | textarea 要素が使用されます。 |

### コントロールフォームのユーザーインターフェイスプロパティーの説明

コネクターまたはステップエクステンションでは、Fuse Online コンソールに表示される各 HTML

フォームコントロールに対して、以下の表に説明があるプロパティーを 1 つ以上指定できます。HTML フォームの入力タイプに関する詳細は、https://www.w3schools.com/html/html_form_input_types.asp を参照してください。

| プロパティー名 | タイプ | 説明 |
|---|---|---|
| **type** | string | Fuse Online が表示するフォームコントロールの種類を制御します。詳細は、前述の表を参照してください。 |
| **cols** | number | **textarea** フィールドに設定された場合、textarea 制御で最初に表示される列数を制御します。 |
| **controlHint** または **controlTooltip** | string | 設定された場合、値はフォームコントロール要素の HTML **title** 属性にマッピングされます。**title** 属性を持つ他の要素と同様に、コントロールにカーソルを合わせるとツールチップが表示されます。ツールチップの内容は、**controlHint** または **controlTooltip** プロパティーの値から取得されます。 |
| **dataList** | array | **type** プロパティーの値が **text** の場合、Fuse Online は **dataList** プロパティーの値を使用してタイプアヘッドサポートを追加します。文字列の配列を指定します。 |
| **defaultValue** | **type** プロパティーの値によって異なります。 | Fuse Online は、最初にこの値をフォームフィールドに表示します。**defaultValue** プロパティーの設定のタイプは、**type** プロパティーの値と一致する必要があります。たとえば、**type** プロパティーが **number** に設定された場合、**defaultValue** 設定は number になります。ユーザーがこの初期フィールド値を変更しないと、Fuse Online は **defaultValue** を使用します。 |
| **description** | string | 設定されている場合、Fuse Online はこの値をフォームコントロールの下に表示します。通常、これはコントロールに関する短い便利なメッセージです。 |
| **displayName** | string | Fuse Online はこの値を表示します。 |
| **enum** | array | 設定した場合、Fuse Online は **type** プロパティーの設定をオーバーライドし、**select** コントロールを実装します。配列を **label** および **value** 属性のセットとして指定します。**label** 属性は、選択項目のラベルとしてユーザーインターフェイスに表示されます。**value** 属性は、対応する選択項目の値になります。 |

| プロパティー名 | タイプ | 説明 |
| --- | --- | --- |
| **labelHint** または **labelTooltip** | string | 設定されている場合、表示名の横に **?** アイコンが表示されます。Fuse Online ユーザーが **?** アイコンをクリックすると、**labelHint** プロパティーの値が表示されます。 |
| **max** | number | **number** フィールドが設定されている場合、許容される最大値を定義します。 |
| **min** | number | **number** フィールドが設定されている場合、許容される最小値を定義します。 |
| **multiple** | Boolean | **select** フィールドまたは **enum** プロパティーセットを持つフィールドに **true** が設定されている場合、Fuse Online は選択ドロップダウンの代わりに複数選択コントロールを表示します。 |
| **order** | number | Fuse Online コンソールでの制御の順序を決定します。Fuse Online は昇順を適用します。つまり、**"order": "1"** のコントロールが最初に表示されます。**order** プロパティーを指定しないと、Fuse Online は JSON ファイルが定義する順序でコントロールを表示します。 |
| **placeholder** | string | 設定された場合、ユーザーに想定される入力を分かりやすく示すため、この値を薄いフォントで入力フィールドに表示します。 |
| **required** | Boolean | **required** 属性がコントロールに設定されるかどうかを制御します。true の場合、Fuse Online ユーザーはこのコントロールの値を入力する必要があります。 |
| **raws** | number | **type** プロパティーの値が **textarea** の場合、**rows** プロパティーは textarea コントロールに最初に表示される行数を制御します。 |
| **secret** | Boolean | 指定された場合、Fuse Online はコントロールの **type** プロパティーの設定を **password** に変更します。 |

### 9.3.6. エクステンションをサポートする Maven プラグインの説明

**extension-maven-plugin** は、エクステンションを有効な Spring Boot モジュールとしてパッケージ化し、エクステンションの開発をサポートします。Java に実装するステップエクステンションでは、このプラグインでエクステンション定義 JSON ファイルを生成できます。

Maven プロジェクトの **pom.xml** ファイルに、以下のプラグイン宣言を追加します。

```
<plugin>
  <groupId>io.syndesis.extension</groupId>
  <artifactId>extension-maven-plugin</artifactId>
  <version>${syndesis.version}</version>
  <executions>
    <execution>
    <goals>
      <goal>generate-metadata</goal>
      <goal>repackage-extension</goal>
    </goals>
    </execution>
  </executions>
</plugin>
```

**extension-maven-plugin** は以下のゴールを定義します。

- **generate-metadata** は、生成された JAR ファイルに含まれる JSON エクステンション定義ファイルを以下のように生成します。

  a. Maven は **META-INF/syndesis/syndesis-extension-definition.json** ファイルにあるデータ構造の指定から開始します (このファイルがある場合)。
  XML でコーディングする場合、エクステンション定義 JSON ファイルを独自に定義する必要があり、そのファイルに必要なデータ構造をすべて指定する必要があります。

  コネクターまたはライブラリーエクステンションを開発する場合、エクステンション定義 JSON ファイルを独自に定義する必要があり、そのファイルに必要なデータ構造をすべて指定する必要があります。

  Java でステップエクステンションを開発する場合、以下を行うことが可能です。

  - エクステンション定義 JSON ファイルをユーザー自身が作成します。
  - Java コードで、必要なデータ構造すべてを定義するアノテーションを指定します。ユーザーはエクステンション定義 JSON ファイルを作成しません。
  - エクステンション定義 JSON ファイルを作成し、一部のデータ構造を指定します。

  b. Java で開発するステップエクステンションの場合、Maven はコードアノテーションから、不足している指定を取得します。

  c. Maven は、scope が **provided** で、**extension-bom** を介して管理される依存関係を指定する、依存関係リストを追加します。

- **repackage-extension** はエクステンションをパッケージ化します。
  - **extension-bom** を介して管理されない依存関係および関連する推移的依存関係は、生成された JAR の **lib** フォルダーにあります。
  - ライブラリーエクステンションの場合、scope が **system** の依存関係は、生成された JAR の **lib** フォルダーにあります。

たとえば、Maven プロジェクトに以下の **pom.xml** ファイルがあるとします。

```
<?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.company</groupId>
  <artifactId>my-extension</artifactId>
  <version>1.0.0</version>
  <name>MyExtension</name>
  <description>A Sample Extension</description>
  <packaging>jar</packaging>

  <dependencyManagement>
    <dependencies>
      <dependency>
        <groupId>io.syndesis.extension</groupId>
        <artifactId>extension-bom</artifactId>
        <version>1.3.10</version>
        <type>pom</type>
        <scope>import</scope>
      </dependency>
    </dependencies>
  </dependencyManagement>

  <dependencies>
    <dependency>
      <groupId>io.syndesis.extension</groupId>
      <artifactId>extension-api</artifactId>
      <scope>provided</scope>
    </dependency>
    <dependency>
      <groupId>com.github.lalyos</groupId>
      <artifactId>jfiglet</artifactId>
      <version>0.0.8</version>
    </dependency>
  </dependencies>

  <build>
    <plugins>
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-compiler-plugin</artifactId>
        <version>3.7.0</version>
        <configuration>
          <source>1.8</source>
          <target>1.8</target>
        </configuration>
      </plugin>
      <plugin>
        <groupId>io.syndesis.extension</groupId>
        <artifactId>extension-maven-plugin</artifactId>
        <version>1.3.10</version>
        <executions>
          <execution>
            <goals>
              <goal>generate-metadata</goal>
              <goal>repackage-extension</goal>
            </goals>
```

```
          </execution>
        </executions>
      </plugin>
    </plugins>
  </build>
</project>
```

この **pom.xml** ファイルを基にすると、生成されたエクステンション定義 JSON ファイルは以下のようになります。

```
{
  "name": "MyExtension",
  "description": "A Sample Extension",
  "extensionId": "com.company:my-extension",
  "version": "1.0.0",
  "dependencies": [ {
    "type": "MAVEN",
    "id": "io.syndesis.extension:extension-api:jar:1.3.10"
  } ],
  "extensionType": "Libraries",
  "schemaVersion": "v1"
}
```

生成されたアーカイブの構造と内容は次のようになります。

```
my-extension-1.0.0.jar
|
+- lib
|  |
|  + jfiglet-0.0.8.jar
|
+- META-INF
   |
   +- MANIFEST.MF
      |
      +- syndesis
         |
         +- syndesis-extension-definition.json
```

### 9.3.7. エクステンションでデータシェイプを指定する方法

データシェイプは、データマッパーによって使用されるデータタイプメタデータを保持します。データマッパーはこのメタデータを、データマッパーユーザーインターフェイスでソースおよびターゲットデータフィールドを表示するために使用する内部ドキュメントに変換します。コネクターまたはカスタムステップのエクステンション定義 JSON ファイルでは、各アクションの指定が入力データシェイプ (**inputDataShape**) および出力データシェイプ (**outputDataShape**) を定義します。

エクステンションを開発する場合、データマッパーがソースおよびターゲットフィールドを正しく処理および表示できるようにするデータシェイププロパティーを指定することが重要になります。以下のデータシェイププロパティーは、データマッパーの動作に影響します。

- **kind**
- **type**

- **specification**
- **name**
- **description**

### kind プロパティー

データシェイプ **kind** プロパティーは **DataShapeKinds** 列挙によって表されます。**kind** プロパティーで使用可能な値は次のとおりです。

- **any** は、データタイプが構造化されていないことを示します。たとえば、バイト配列やフリーフォーマットテキストであることがあります。データマッパーは、**kind** プロパティーが **any** に設定されている場合にデータシェイプを無視します。つまり、データはデータマッパーに表示されないため、このデータとフィールドの間にマッピングを作成することはできません。
しかし、**kind** プロパティーが **any** に設定された場合、カスタムコネクターから作成したコネクションを設定するときに、Fuse Online は入力および出力データタイプを指定するよう要求します。これは、コネクションをインテグレーションに追加するときに発生します。データシェイプのスキーマの種類、指定するスキーマの種類に対応するドキュメント、およびデータタイプの名前を指定できます。

- **none** はデータタイプがないことを示します。入力データシェイプでは、コネクションまたはステップがデータを読み取らないことを示します。出力データシェイプでは、コネクションまたはステップがデータを変更しないことを示します。たとえば、入力メッセージボディーが出力メッセージボディーに転送される場合は、**kind** プロパティーを **none** に設定すると、データが通過することを意味します。データマッパーは、**kind** が **none** に設定された場合にデータシェイプを無視します。つまり、データはデータマッパーに表示されないため、このデータとフィールドの間にマッピングを作成することはできません。

- **java** は、データタイプが Java クラスによって表されることを示します。**type** プロパティーの完全修飾クラス名を指定して、**"kind": "java"** 宣言に従います。以下は例になります。

```
"outputDataShape": {
    "kind": "java",
    "type": "org.apache.camel.component.telegram.model.IncomingMessage"
},
```

- **json-schema** はデータタイプが JSON スキーマによって表されることを示します。**kind** が **json-schema** に設定された場合、JSON スキーマをデータシェイプの **specification** プロパティーの値として指定します。以下は例になります。

```
"inputDataShape": {
  "description": "Person data",
  "kind": "json-schema",
  "name": "Person",
  "specification": "{\"$schema\":\"http://json-schema.org/draft-
04/schema#\",\"title\":\"Person\",\"type\":\"object\",\"properties\":{\"firstName\":{...}}}"
}
```

SAP Concur コネクターのコードには、JSON スキーマによって指定されるデータシェイプの例 が含まれています。

- **json-instance** は、データタイプが JSON インスタンスで表されることを示しています。**kind** が **json-instance** に設定された場合、JSON インスタンスをデータシェイプの **specification** プロパティーの値として指定します。以下は例になります。

```
"inputDataShape": {
  "description": "Person data",
  "kind": "json-instance",
  "name": "Person",
  "specification": "{\"firstName\":\"John\",...}"
}
```

- **xml-schema** は、データタイプが XML スキーマで表されることを示しています。**kind** が **xml-schema** に設定された場合、XML スキーマをデータシェイプの **specification** プロパティーの値として指定します。以下は例になります。

```
"inputDataShape": {
  "description": "Person data",
  "kind": "xml-schema",
  "name": "Person",
  "specification": "<?xml version=\"1.0\" encoding=\"UTF-8\" ?><xs:schema
xmlns:xs=\"http://www.w3.org/2001/XMLSchema\">...</xs:schema>"
}
```

- **xml-instance** は、データタイプが XML インスタンスによって表されることを示しています。**kind** が **xml-instance** に設定された場合、XML インスタンスをデータシェイプの **specification** プロパティーの値として指定します。以下に例を示します。

```
"inputDataShape": {
  "description": "Person data",
  "kind": "xml-instance",
  "name": "Person",
  "specification": "<?xml version=\"1.0\" encoding=\"UTF-8\" ?><Person>
<firstName>Jane</firstName></Person>"
}
```

- **csv-instance** は、データ型が CSV インスタンスによって表されることを示します。**kind** が **csv-instance** に設定されている場合は、CSV インスタンスをデータシェイプの **specification** プロパティーの値として指定します。以下に例を示します。

```
"inputDataShape": {
  "description": "Person data",
  "kind": "csv-instance",
  "name": "Person",
  "specification":  "John,Doe,120 Jefferson Street,Riverside, NJ, 08075"
}
```

**kind** が **csv-instance** に設定されている場合は、次の **boolean** (**true**/**false**) パラメーターを指定できます。

| ラベル | 名前。 | 説明 |
|---|---|---|
| ヘッダー名の重複を許可 | **allowDuplicateHeaderNames** | CSV データのヘッダー行で名前の重複を許可します。 |
| 欠落している列名を許可 | **allowMissingColumnNames** | CSV データのヘッダー行を解析するときに、欠落している列名を許可します。 |

| ラベル | 名前。 | 説明 |
| --- | --- | --- |
| コメントマーカー | **commentMarker** | CSV データのコメント行の開始を示す文字を指定します。 |
| デリミタ | **delimiter** | CSV データの値を区切る文字 (通常は ";"、","、または "\t") を指定します。 |
| Escape | **escape** | CSV データのエスケープ文字を指定します。 |
| ヘッダーとしての最初のレコード | **firstRecordAsHeader** | CSV データの最初のレコードをヘッダー行として使用します。 |
| 空行を無視 | **ignoreEmptyLines** | CSV データの空行を無視します。 |
| ヘッダーの大文字小文字を無視 | **ignoreHeaderCase** | CSV データのヘッダー行の大文字と小文字を無視します。 |
| 周囲のスペースを無視 | **ignoreSurroundingSpaces** | CSV データを囲む空白文字は無視されます。 |
| Null 文字列 | **nullString** | CSV データで null との間で変換するときに使用する文字列を指定します。 |
| ヘッダーレコードをスキップ | **skipHeaderRecord** | CSV データのヘッダーレコードをスキップします。 |

### type プロパティー

**kind** プロパティーの値が **java** の場合、**"kind": "java"** 宣言の後に完全修飾 Java クラス名を指定する **type** 宣言が続きます。以下は例になります。

```
"outputDataShape": {
    "kind": "java",
    "type": "org.apache.camel.component.telegram.model.IncomingMessage"
},
```

**kind** プロパティーが **java** 以外に設定された場合、**type** プロパティーの設定は無視されます。

### specification プロパティー

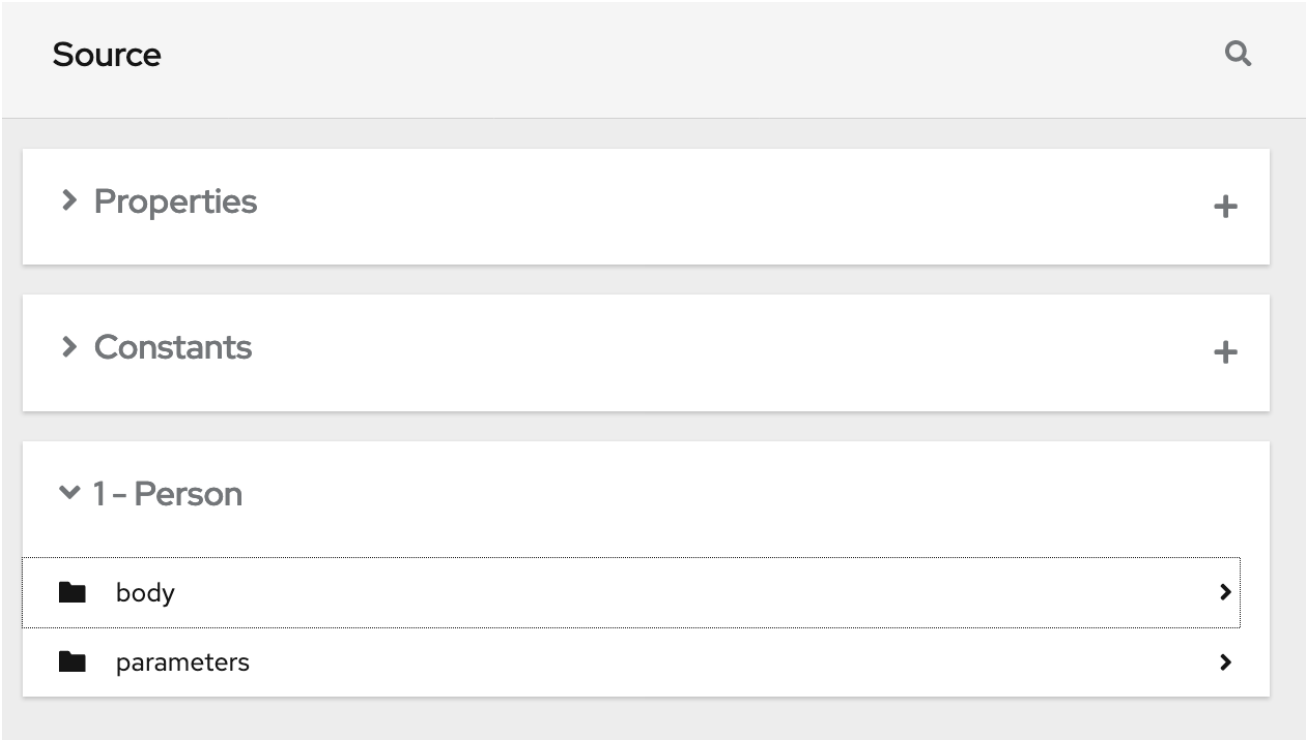
**kind** プロパティーの設定は、以下の表のように、**specification** プロパティーの設定を決定します。

| kind プロパティーの設定 | specification プロパティーの設定 |
| --- | --- |

| kind プロパティーの設定 | specification プロパティーの設定 |
| --- | --- |
| **java** | Java インスペクションの結果。<br><br>Java で作成する各エクステンションに、**extension-maven-plugin** を使用して、最低でも Java インスペクションの結果を取得します。プラグインは、Java インスペクションの結果を **specification** プロパティーの設定として JSON エクステンション定義ファイルに挿入します。これは、Fuse Online のデータマッピングに必要な Java インスペクションの結果を取得する唯一の方法です。<br><br>Java で記述されたステップエクステンションの場合、**extension-maven-plugin** は JSON エクステンション定義ファイルを生成し、必要なコンテンツを追加します。コネクターエクステンションでは、**extension-maven-plugin** は Java インスペクションの結果を JSON エクステンション定義ファイルに挿入しますが、プラグインが挿入しない必要なコンテンツを手作業で追加する必要があります。 |
| **json-schema** | 実際の JSON スキーマのドキュメント。設定をドキュメントへの参照とすることはできず、参照の手段として JSON スキーマが他の JSON スキーマドキュメントを示すことはできません。 |
| **json-instance** | サンプルデータが含まれる実際の JSON ドキュメント。データマッパーは、サンプルデータからデータタイプを取得します。この設定をドキュメントへの参照とすることはできません。 |
| **xml-schema** | 実際の XML スキーマドキュメント。設定をドキュメントへの参照とすることはできず、参照の手段として XML スキーマが他の XML スキーマドキュメントを示すことはできません。 |
| **xml-instance** | 実際の XML インスタンスドキュメントです。この設定をドキュメントへの参照とすることはできません。 |
| **csv-instance** | 実際の CSV インスタンスドキュメント。この設定をドキュメントへの参照とすることはできません。 |
| **any** | **specification** プロパティーは必要ありません。設定は無視されます。 |
| **none** | **specification** プロパティーは必要ありません。設定は無視されます。 |

**name プロパティー**

データシェイプの **name** プロパティーは、データタイプの人間が判読できる名前を指定します。データマッパーは、この名前をデータフィールドのラベルとしてユーザーインターフェイスに表示します。以下のイメージの **Person** は、**name** プロパティーの値が表示される場所の例になります。




この名前は、Fuse Online フロービジュアライゼーションのデータタイプインジケーターにも表示されます。

#### description プロパティー

データシェイプの **description** プロパティーは、データマッパーユーザーインターフェイスのデータタイプ名にカーソルを合わせたときにツールチップとして表示されるテキストを指定します。

### 9.3.8. ステップエクステンションの開発例

ステップエクステンションは、1つまたは複数のカスタムステップを実装します。各カスタムステップは、コネクション間のインテグレーションデータを処理するために1つのアクションを実装します。以下の例は、ステップエクステンション開発の代替を示します。

- 「XML フラグメントでの Camel ルートの開発例」
- 「**RouteBuilder** を使った Camel ルートの開発例」
- 「**RouteBuilder** および Spring Boot を使った Camel ルートの開発例」
- 「Camel Bean の使用例」
- 「Syndesis Step API の使用例」

Syndesis は、**syndesis-extension-plugin** とともに使用できる、カスタム Java アノテーションを提供します。ステップエクステンションまたはコネクターエクステンションを Java で実装する場合、Maven がアクション定義をエクステンション定義 JSON ファイルに追加できるようにするアノテーションを指定できます。アノテーションの処理を有効にするには、以下の依存関係を Maven プロジェクトに追加します。

```
<dependency>
```

```
  <groupId>io.syndesis.extension</groupId>
  <artifactId>extension-annotation-processor</artifactId>
  <optional>true</optional>
</dependency>
```

Spring Boot はインテグレーションランタイムであるため、Bean を Camel コンテキストにインジェクトするには、Spring Boot の標準の方法に従う必要があります。たとえば、自動設定クラスを作成 し、そこに Bean を作成します。ただし、デフォルトの動作では、エクステンションコードはパッケージスキャンの対象ではありません。したがって、ステップエクステンションの **META-INF/spring.factories** ファイルを作成し、内容を追加する必要があります。

#### 9.3.8.1. XML フラグメントでの Camel ルートの開発例

カスタムステップを開発するには、 **direct** などの入力がある Camel ルートである XML フラグメントとして、アクションを実装できます。Syndesis ランタイムは、このルートを他の Camel ルートを呼び出す同じ方法で呼び出します。

たとえば、任意の接頭辞を持つメッセージのボディーをログに記録するステップを作成する場合に以下の XML はこれを行う Camel ルートを定義します。

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

  <route id="log-body-with-prefix">
    <from uri="direct:log"/>
    <choice>
      <when>
        <simple>${header.prefix} != ''</simple>
        <log message="${header.prefix} ${body}"/>
      </when>
      <otherwise>
        <log message="Output ${body}"/>
      </otherwise>
    </choice>
  </route>

</routes>
```

XML でエクステンションを開発する場合は、エクステンション定義 JSON ファイルを独自に作成する必要があります。この XML フラグメントに対し、**src/main/resources/META-INF/syndesis/syndesis-extension-definition.json** ファイルは以下のようにアクションを定義できます。

```
{
  "actionType": "step",
  "id": "log-body-with-prefix",
  "name": "Log body with prefix",
  "description": "A simple body log with a prefix",
  "descriptor": {
    "kind": "ENDPOINT", (1)
    "entrypoint": "direct:log", (2)
```

```
  "resource": "classpath:log-body-action.xml", 3
  "inputDataShape": {
    "kind": "none"
  },
  "outputDataShape": {
    "kind": "none"
  },
 "propertyDefinitionSteps": [ {
  "description": "extension-properties",
  "name": "extension-properties",
  "properties": { 4
    "prefix": {
      "componentProperty": false,
      "deprecated": false,
      "description": "The Log body prefix message",
      "displayName": "Log Prefix",
      "javaType": "String",
      "kind": "parameter",
      "required": false,
      "secret": false,
      "type": "string"
    }
  }
 } ]
 }
}
```

1 アクションのタイプは **ENDPOINT** に設定されます。ランタイムは、Camel エンドポイントを呼び出して、このカスタムステップによって提供されるアクションを実行します。

2 呼び出す Camel エンドポイントは **direct:log** です。これはルートの **from** 指定です。

3 これは、XML フラグメントの場所になります。

4 これらは、このカスタムステップでアクションが定義したプロパティーで、このステップをインテグレーションに追加するインテグレーターに公開します。Fuse Online では、インテグレーターがユーザーインターフェイスで指定する値は、プロパティーと同じ名前を持つメッセージヘッダーにマップされます。この例では、インテグレーターは **Log Prefix** 表示名を持つ 1 つの入力フィールドを確認できます。詳細は、ユーザーインターフェイスプロパティーの説明 を参照してください。

**警告**

Syndesis は、完全な Camel XML 設定をサポートしません。Syndesis は <routes> タグのみをサポートします。

#### 9.3.8.2. RouteBuilder を使った Camel ルートの開発例

カスタムステップを実装するには、**RouteBuilder** クラスを利用してアクションを Camel ルートとして開発します。このようなルートには、**direct** などの入力があります。Syndesis は、他の Camel ルートの呼び出しと同じ方法でこのルートを呼び出します。

任意の接頭辞でメッセージのボディーをログに記録するステップを作成する例を実装するには、以下のように記述します。

```
import org.apache.camel.builder.RouteBuilder;

import io.syndesis.extension.api.annotations.Action;
import io.syndesis.extension.api.annotations.ConfigurationProperty;

@Action( 1
    id = "log-body-with-prefix",
    name = "Log body with prefix",
    description = "A simple body log with a prefix",
    entrypoint = "direct:log")
public class LogAction extends RouteBuilder {
    @ConfigurationProperty( 2
        name = "prefix",
        description = "The Log body prefix message",
        displayName = "Log Prefix",
        type = "string")
    private String prefix;

    @Override
    public void configure() throws Exception {
        from("direct::start") 3
            .choice()
                .when(simple("${header.prefix} != ""))
                    .log("${header.prefix} ${body}")
                .otherwise()
                    .log("Output ${body}")
            .endChoice();
    }
}
```

1. **@Action** アノテーションは、アクション定義を示します。
2. **@ConfigurationProperty** アノテーションは、ユーザーインターフェイスのフォームコントロールの定義を示します。詳細は、ユーザーインターフェイスプロパティーの説明 を参照してください。
3. これはアクションの実装です。

この Java コードは Syndesis アノテーションを使用するため、**extension-maven-plugin** はアクション定義を自動的に生成できます。エクステンション定義 JSON ファイルのアクション定義は以下のようになります。

```
{
  "id": "log-body-with-prefix",
  "name": "Log body with prefix",
  "description": "A simple body log with a prefix",
  "descriptor": {
    "kind": "ENDPOINT", 1
```

```
  "entrypoint": "direct:log", (2)
  "resource": "class:io.syndesis.extension.log.LogAction", (3)
  "inputDataShape": {
    "kind": "none"
  },
  "outputDataShape": {
    "kind": "none"
  },
  "propertyDefinitionSteps": [ {
    "description": "extension-properties",
    "name": "extension-properties",
    "properties": { (4)
      "prefix": {
        "componentProperty": false,
        "deprecated": false,
        "description": "The Log body prefix message",
        "displayName": "Log Prefix",
        "javaType": "java.lang.String",
        "kind": "parameter",
        "required": false,
        "secret": false,
        "type": "string",
        "raw": false
      }
    }
  } ]
 },
 "actionType": "step"
}
```

(1) アクションのタイプは **ENDPOINT** です。ランタイムは Camel エンドポイントを呼び出して、このステップが実装するアクションを実行します。

(2) これが呼び出す Camel エンドポイントです。これはルートの **from** 指定です。

(3) これは、**RoutesBuilder** を実装するクラスです。

(4) これらは、このカスタムステップでアクションが定義したプロパティーで、このステップをインテグレーションに追加するインテグレーターに公開します。Fuse Online では、インテグレーターがユーザーインターフェイスで指定する値は、プロパティーと同じ名前を持つメッセージヘッダーにマップされます。この例では、インテグレーターは **Log Prefix** 表示名を持つ 1 つの入力フィールドを確認できます。詳細は、ユーザーインターフェイスプロパティーの説明 を参照してください。

#### 9.3.8.3. RouteBuilder および Spring Boot を使った Camel ルートの開発例

カスタムステップを実装するには、**RouteBuilder** クラスと Spring Boot を利用してアクションを Camel ルートとして開発します。この例では、Spring Boot は **RouteBuilder** オブジェクトを Camel コンテキストに登録するファシリティーです。Syndesis は、他の Camel ルートの呼び出しと同じ方法でこのルートを呼び出します。

任意の接頭辞でメッセージのボディーをログに記録するステップを作成する例を実装するには、以下のように記述します。

```
import io.syndesis.extension.api.annotations.Action;
import io.syndesis.extension.api.annotations.ConfigurationProperty;
import org.apache.camel.builder.RouteBuilder;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

@Configuration
public class ActionsConfiguration {

    @Action( 1
        id = "log-body-with-prefix",
        name = "Log body with prefix",
        description = "A simple body log with a prefix",
        entrypoint = "direct:log")
    @ConfigurationProperty( 2
        name = "prefix",
        description = "The Log body prefix message",
        displayName = "Log Prefix",
        type = "string")
    @Bean 3
    public RouteBuilder logBodyWithprefix() {
        return new RouteBuilder() {
            @Override
            public void configure() throws Exception {
                from("direct::start") 4
                    .choice()
                        .when(simple("${header.prefix} != ''"))
                            .log("${header.prefix} ${body}")
                        .otherwise()
                            .log("Output ${body}")
                    .endChoice();
            }
        };
    }
}
```

1. **@Action** アノテーションは、アクション定義を示します。
2. **@ConfigurationProperty** アノテーションは、ユーザーインターフェイスのフォームコントロールの定義を示します。詳細は、ユーザーインターフェイスプロパティーの説明 を参照してください。
3. **RouteBuilder** オブジェクトを Bean として登録します。
4. これはアクションの実装です。

この Java コードは Syndesis アノテーションを使用するため、**extension-maven-plugin** はアクション定義を自動的に生成できます。エクステンション定義 JSON ファイルのアクション定義は以下のようになります。

```
{
  "id": "log-body-with-prefix",
  "name": "Log body with prefix",
  "description": "A simple body log with a prefix",
```

```
"descriptor": {
  "kind": "ENDPOINT", (1)
  "entrypoint": "direct:log", (2)
  "inputDataShape": {
    "kind": "none"
  },
  "outputDataShape": {
    "kind": "none"
  },
  "propertyDefinitionSteps": [ {
    "description": "extension-properties",
    "name": "extension-properties",
    "properties": { (3)
      "prefix": {
        "componentProperty": false,
        "deprecated": false,
        "description": "The Log body prefix message",
        "displayName": "Log Prefix",
        "javaType": "java.lang.String",
        "kind": "parameter",
        "required": false,
        "secret": false,
        "type": "string",
        "raw": false
      }
    }
  } ]
},
"actionType": "step"
}
```

(1) アクションのタイプは **ENDPOINT** です。ランタイムは Camel エンドポイントを呼び出して、このステップが実装するアクションを実行します。

(2) これが呼び出す Camel エンドポイントです。これはルートの **from** 指定です。

(3) これらは、このカスタムステップでアクションが定義したプロパティーで、このステップをインテグレーションに追加するインテグレーターに公開します。Fuse Online では、インテグレーターがユーザーインターフェイスで指定する値は、プロパティーと同じ名前を持つメッセージヘッダーにマップされます。この例では、インテグレーターは **Log Prefix** 表示名を持つ 1 つの入力フィールドを確認できます。詳細は、ユーザーインターフェイスプロパティーの説明 を参照してください。

**重要**

Spring Boot が設定クラスを検出できるようにするには、以下のように **META-INF/spring.factories** という名前のファイルにこれらのクラスを記載する必要があります。

**org.springframework.boot.autoconfigure.EnableAutoConfiguration=com.company.ActionsConfiguration**

Spring Boot では、設定クラスに最終的に登録するすべての Bean を Camel コンテキストで利用できます。詳細は、独自の自動設定を作成 するための Spring Boot ドキュメントを参照してください。

#### 9.3.8.4. Camel Bean の使用例

アクションを Camel Bean プロセッサーとして開発すると、カスタムステップを実装できます。任意の接頭辞でメッセージのボディーをログに記録するステップを作成する例を実装するには、以下のように記述します。

```
import org.apache.camel.Body;
import org.apache.camel.Handler;
import org.apache.camel.Header;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import io.syndesis.extension.api.annotations.Action;
import io.syndesis.extension.api.annotations.ConfigurationProperty;

@Action(
    id = "log-body-with-prefix",
    name = "Log body with prefix",
    description = "A simple body log with a prefix")
public class LogAction  {
    private static final Logger LOGGER = LoggerFactory.getLogger(LogAction.class);

    @ConfigurationProperty(
        name = "prefix",
        description = "The Log body prefix message",
        displayName = "Log Prefix",
        type = "string")
    private String prefix;

    @Handler (1)
    public void process(@Header("prefix") String prefix, @Body Object body) {
        if (prefix == null) {
            LOGGER.info("Output {}", body);
        } else {
            LOGGER.info("{} {}", prefix, body);
        }
    }
}
```

(1) これは、アクションを実装する関数です。

この Java コードは Syndesis アノテーションを使用するため、**extension-maven-plugin** はアクション定義を自動的に生成できます。エクステンション定義 JSON ファイルのアクション定義は以下のようになります。

```
{
  "id": "log-body-with-prefix",
  "name": "Log body with prefix",
  "description": "A simple body log with a prefix",
  "descriptor": {
    "kind": "BEAN", (1)
    "entrypoint": "io.syndesis.extension.log.LogAction::process", (2)
    "inputDataShape": {
      "kind": "none"
```

```
    },
    "outputDataShape": {
      "kind": "none"
    },
    "propertyDefinitionSteps": [ {
      "description": "extension-properties",
      "name": "extension-properties",
      "properties": {
        "prefix": { 3
          "componentProperty": false,
          "deprecated": false,
          "description": "The Log body prefix message",
          "displayName": "Log Prefix",
          "javaType": "java.lang.String",
          "kind": "parameter",
          "required": false,
          "secret": false,
          "type": "string",
          "raw": false
        }
      }
    } ]
  },
  "actionType": "step"
}
```

1 アクションのタイプは **BEAN** です。ランタイムは、Camel Bean プロセッサーを呼び出して、このカスタムステップのアクションを実行します。

2 この Camel Bean を呼び出します。

3 これらは、このカスタムステップでアクションが定義したプロパティーで、このステップをインテグレーションに追加するインテグレーターに公開します。Fuse Online では、インテグレーターがユーザーインターフェイスで指定する値は、プロパティーと同じ名前を持つメッセージヘッダーにマップされます。この例では、インテグレーターは **Log Prefix** 表示名を持つ 1 つの入力フィールドを確認できます。詳細は、ユーザーインターフェイスプロパティーの説明 を参照してください。

Bean を使用する場合、交換ヘッダーからユーザープロパティーを取得する代わりに、ユーザープロパティーを Bean にインジェクトすると便利である可能性があります。これには、インジェクトするプロパティーに getter および setter メソッドを実装します。アクション実装は以下のようになります。

```
import org.apache.camel.Body;
import org.apache.camel.Handler;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import io.syndesis.extension.api.annotations.Action;
import io.syndesis.extension.api.annotations.ConfigurationProperty;

@Action(
    id = "log-body-with-prefix",
    name = "Log body with prefix",
    description = "A simple body log with a prefix")
public class LogAction  {
```

```
    private static final Logger LOGGER = LoggerFactory.getLogger(LogAction.class);

    @ConfigurationProperty(
        name = "prefix",
        description = "The Log body prefix message",
        displayName = "Log Prefix",
        type = "string")
    private String prefix;

    public void setPrefix(String prefix) { 1
        this.prefix = prefix;
    }

    public String getPrefix() { 2
        return prefix;
    }

    @Handler
    public void process(@Body Object body) {
        if (this.prefix == null) {
            LOGGER.info("Output {}", body);
        } else {
            LOGGER.info("{} {}", this.prefix, body);
        }
    }
}
```

1 これは、プロパティーの setter メソッドです。

2 これは、プロパティーの getter メソッドです。

#### 9.3.8.5. Syndesis Step API の使用例

Syndesis **Step** API を使用することで、カスタムステップを実装できます。これにより、ランタイムルートの作成と対話することが可能になります。**ProcessorDefinition** クラスによって提供されるすべてのメソッドを使用でき、さらに複雑なルートを作成できます。Syndesis API に関する情報は http://javadoc.io/doc/io.syndesis.extension/extension-api を参照してください。

以下は、スプリットアクションを実装するために Syndesis **Step** API を使用するステップエクステンションの例になります。

```
import java.util.Map;
import java.util.Optional;

import io.syndesis.extension.api.Step;
import io.syndesis.extension.api.annotations.Action;
import io.syndesis.extension.api.annotations.ConfigurationProperty;
import org.apache.camel.CamelContext;
import org.apache.camel.model.ProcessorDefinition;
import org.apache.camel.util.ObjectHelper;
import org.apache.camel.Expression;
import org.apache.camel.builder.Builder;
import org.apache.camel.processor.aggregate.AggregationStrategy;
import org.apache.camel.processor.aggregate.UseOriginalAggregationStrategy;
```

```
import org.apache.camel.spi.Language;

@Action(id = "split", name = "Split", description = "Split your exchange")
public class SplitAction implements Step {

    @ConfigurationProperty(
        name = "language",
        displayName = "Language",
        description = "The language used for the expression")
    private String language;

    @ConfigurationProperty(
        name = "expression",
        displayName = "Expression",
        description = "The expression used to split the exchange")
    private String expression;

    public String getLanguage() {
        return language;
    }

    public void setLanguage(String language) {
        this.language = language;
    }

    public String getExpression() {
        return expression;
    }

    public void setExpression(String expression) {
        this.expression = expression;
    }

    @Override
    public Optional<ProcessorDefinition> configure(
            CamelContext context,
            ProcessorDefinition route,
            Map<String, Object> parameters) { (1)

        String languageName = language;
        String expressionDefinition = expression;

        if (ObjectHelper.isEmpty(languageName) && ObjectHelper.isEmpty(expressionDefinition)) {
            route = route.split(Builder.body());
        } else if (ObjectHelper.isNotEmpty(expressionDefinition)) {

            if (ObjectHelper.isEmpty(languageName)) {
                languageName = "simple";
            }

            final Language splitLanguage = context.resolveLanguage(languageName);
            final Expression splitExpression = splitLanguage.createExpression(expressionDefinition);
            final AggregationStrategy aggreationStrategy = new UseOriginalAggregationStrategy(null,
false);

            route = route.split(splitExpression).aggregationStrategy(aggreationStrategy);
```

```
        }

        return Optional.of(route);
    }
}
```

1 これは、カスタムステップが実行するアクションの実装です。

この Java コードは Syndesis アノテーションを使用するため、**extension-maven-plugin** はアクション定義を自動的に生成できます。エクステンション定義 JSON ファイルのアクション定義は以下のようになります。

```
{
  "id": "split",
  "name": "Split",
  "description": "Split your exchange",
  "descriptor": {
    "kind": "STEP", 1
    "entrypoint": "io.syndesis.extension.split.SplitAction", 2
    "inputDataShape": {
      "kind": "none"
    },
    "outputDataShape": {
      "kind": "none"
    },
    "propertyDefinitionSteps": [ {
      "description": "extension-properties",
      "name": "extension-properties",
      "properties": {
        "language": {
          "componentProperty": false,
          "deprecated": false,
          "description": "The language used for the expression",
          "displayName": "Language",
          "javaType": "java.lang.String",
          "kind": "parameter",
          "required": false,
          "secret": false,
          "type": "string",
          "raw": false
        },
        "expression": {
          "componentProperty": false,
          "deprecated": false,
          "description": "The expression used to split the exchange",
          "displayName": "Expression",
          "javaType": "java.lang.String",
          "kind": "parameter",
          "required": false,
          "secret": false,
          "type": "string",
          "raw": false
        }
      }
    } ]
```

```
  },
  "tags": [],
  "actionType": "step"
}
```

1 アクションのタイプは **STEP** です。

2 これは、**Step** インターフェイスを実装するクラスです。

#### その他のリソース

ユーザーインターフェイスプロパティーの詳細は、ユーザーインターフェイスプロパティーの説明 を参照してください。

### 9.3.9. コネクターエクステンションの開発例

Fuse Online が、インテグレーションで接続するアプリケーションやサービスのコネクターを提供しない場合、経験のある開発者は Fuse Online に新しいコネクターを提供するエクステンションを作成できます。このドキュメントは、初心者向けのコネクターエクステンションの開発に関する情報を提供します。コネクターの開発に関する詳細は、Syndesis コミュニティーの Web サイトにある Developing Syndesis connectors を参照してください。

**重要**

コネクターエクステンションでは、Java コードからエクステンション定義 JSON ファイルを自動的に生成できません。

コネクターは基本的に Camel コンポーネントのプロキシーになります。コネクターは基盤のコンポーネントを設定し、エクステンション定義で定義されるオプションや、Fuse Online Web インターフェイスが収集するユーザー提供のオプションに応じてエンドポイントを作成します。

コネクターエクステンションの定義は、以下の追加のデータ構造を使用して、ステップエクステンションに必要なエクステンション定義を拡張します。

- **componentScheme**
  コネクターが使用する Camel コンポーネントを定義します。コネクターまたはアクションに **componentScheme** を設定できます。**componentScheme** をコネクターとアクションの両方に設定した場合、アクションの設定が優先されます。

- **connectorCustomizers**
  ComponentProxyCustomizer クラスを実装するクラスの一覧を指定します。各クラスはコネクターの動作をカスタマイズします。たとえば、クラスはプロパティーが基礎となるコンポーネントやエンドポイントに適用される前にプロパティーを操作したり、クラスは事前または事後のエンドポイントロジックを追加する場合があります。各クラスに対して、**com.mycomponent.MyCustomizer** のように実装の完全クラス名を指定します。**connectorCustomizers** はアクションおよびコネクターに設定できます。Fuse Online は設定に応じて、カスタマイザーを最初にコネクターに適用した後、アクションに適用します。

- **connectorFactory**
  基盤のコンポーネント/エンドポイントの作成や設定を行う、ComponentProxyFactory クラスを実装するクラスを定義します。実装の完全クラス名を指定します。コネクターまたはアクションの **connectorFactory** を設定できます。アクションには優先順位があります。

#### カスタマイザーの例

以下のカスタマイザーの例は、個別のオプションから **DataSource** を設定します。

```
public class DataSourceCustomizer implements ComponentProxyCustomizer, CamelContextAware {
    private final static Logger LOGGER = LoggerFactory.getLogger(DataSourceCustomizer.class);

    private CamelContext camelContext;

    @Override
    public void setCamelContext(CamelContext camelContext) { (1)
        this.camelContext = camelContext;
    }

    @Override
    public CamelContext getCamelContext() { (2)
        return this.camelContext;
    }

    @Override
    public void customize(ComponentProxyComponent component, Map<String, Object> options) {
        if (!options.containsKey("dataSource")) {
            if (options.containsKey("user") && options.containsKey("password") &&
options.containsKey("url")) {
                try {
                    BasicDataSource ds = new BasicDataSource();

                    consumeOption(camelContext, options, "user", String.class, ds::setUsername); (3)
                    consumeOption(camelContext, options, "password", String.class, ds::setPassword); (4)
                    consumeOption(camelContext, options, "url", String.class, ds::setUrl); (5)

                    options.put("dataSource", ds);
                } catch (@SuppressWarnings("PMD.AvoidCatchingGenericException") Exception e) {
                    throw new IllegalArgumentException(e);
                }
            } else {
                LOGGER.debug("Not enough information provided to set-up the DataSource");
            }
        }
    }
}
```

(1)(2) Syndesis は **CamelContextAware** を実装することで Camel コンテキストをインジェクトし、カスタマイズメソッドを呼び出します。

(3)(4)(5) オプションを処理し、オプションマップから削除します。

### プロパティーのインジェクト例

カスタマイザーが Java Bean の慣例に従う場合、以下のようにプロパティーもインジェクトできます (前述の例を編集)。

```
public class DataSourceCustomizer implements ComponentProxyCustomizer, CamelContextAware {
    private final static Logger LOGGER = LoggerFactory.getLogger(DataSourceCustomizer.class);

    private CamelContext camelContext;
```

```
private String userName;
private String password;
private String url;

@Override
public void setCamelContext(CamelContext camelContext) { 1
    this.camelContext = camelContext;
}

@Override
public CamelContext getCamelContext() { 2
    return this.camelContext;
}

public void setUserName(String userName) { 3
  this.userName = userName;
}

public String getUserName() { 4
  return this.userName;
}

public void setPassword(String password) { 5
  this.password = password;
}

public String getPassword() { 6
  return this.password;
}

public void setUrl(String url) { 7
  this.url = url;
}

public String getUrl() { 8
  return this.url;
}

@Override
public void customize(ComponentProxyComponent component, Map<String, Object> options) {
    if (!options.containsKey("dataSource")) {
        if (userName != null && password != null && url != null) {
            try {
                BasicDataSource ds = new BasicDataSource();
                ds.setUserName(userName);
                ds.setPassword(password);
                ds.setUrl(url);

                options.put("dataSource", ds);
            } catch (@SuppressWarnings("PMD.AvoidCatchingGenericException") Exception e) {
                throw new IllegalArgumentException(e);
            }
        } else {
            LOGGER.debug("Not enough information provided to set-up the DataSource");
        }
```

```
        }
    }
}
```

1 2 3 Syndesis は **CamelContextAware** を実装することで Camel コンテキストをインジェクトし、カスタマイズメソッドを呼び出します。このサンプルコードは、**setCamelContext()** および **getCamelContext()** メソッドを上書きし、ユーザー名を設定します。

4 5 6 7 8 サンプルコードはインジェクトされたオプションを処理し、オプションマップから自動的に削除します。

### カスタマイザーを使用した before/after 論理の設定

以下の例のように、カスタマイザーを使用して before/after 論理を設定できます。

```
public class AWSS3DeleteObjectCustomizer implements ComponentProxyCustomizer {
  private String filenameKey;

   public void setFilenameKey(String filenameKey) {
     this.filenameKey = filenameKey;
   }

   public String getFilenameKey() {
      return this.filenameKey;
   }

   @Override
   public void customize(ComponentProxyComponent component, Map<String, Object> options) {
     component.setBeforeProducer(this::beforeProducer);
   }

   public void beforeProducer(final Exchange exchange) throws IOException {
     exchange.getIn().setHeader(S3Constants.S3_OPERATION, S3Operations.deleteObject);

     if (filenameKey != null) {
       exchange.getIn().setHeader(S3Constants.KEY, filenameKey);
     }
   }
}
```

### ComponentProxyComponent の動作のカスタマイズ

ComponentProxyFactory クラスは、基礎となるコンポーネント/エイドポイントの作成や設定を行います。**ComponentProxyFactory** が作成する ComponentProxyComponent オブジェクトの動作をカスタマイズするには、以下のメソッドのいずれかをオーバーライドします。

- **createDelegateComponent()**
  Syndesis は、プロキシーの開始時にこのメソッドを呼び出し、最終的に **componentScheme** オプションによって定義されたスキームでコンポーネントの専用インスタンスを作成します。

  このメソッドのデフォルトの動作は、いずれかのコネクター/アクションオプションがコンポーネントレベルで適用されるかどうかを判断します。同じオプションをエンドポイントに適用できない場合のみ、メソッドによってカスタムコンポーネントインスタンスが作成され、適用可能なオプションに応じて設定されます。

- **configureDelegateComponent()**
  Syndesis は、委譲されたコンポーネントインスタンスの追加の動作を設定するためにカスタムコンポーネントインスタンスが作成された場合に限り、このメソッドを呼び出します。

- **createDelegateEndpoint()**
  Syndesis は、プロキシーがエンドポイントを作成するときにこのメソッドを呼び出し、デフォルトで Camel カタログファシリティーを使用してエンドポイントを作成します。

- **configureDelegateEndpoint()**
  委譲されたエンドポイントの作成後に、Syndesis がこのメソッドを呼び出し、委譲されたエンドポイントインスタンスの追加動作を設定します。以下に例を示します。

```
public class IrcComponentProxyFactory implements ComponentProxyFactory {

    @Override
    public ComponentProxyComponent newInstance(String componentId, String
componentScheme) {
        return new ComponentProxyComponent(componentId, componentScheme) {
            @Override
            protected void configureDelegateEndpoint(ComponentDefinition definition, Endpoint
endpoint, Map<String, Object> options) throws Exception {
                if (!(endpoint instanceof IrcEndpoint)) {
                    throw new IllegalStateException("Endpoint should be of type IrcEndpoint");
                }

                final IrcEndpoint ircEndpoint = (IrcEndpoint)endpoint;
                final String channels = (String)options.remove("channels");

                if (ObjectHelper.isNotEmpty(channels)) {
                    ircEndpoint.getConfiguration().setChannel(
                        Arrays.asList(channels.split(","))
                    );
                }
            }
        };
    }
}
```

### 9.3.10. ライブラリーエクステンションの開発方法

ライブラリーエクステンションは、インテグレーションがランタイムに必要なリソースを提供します。ライブラリーエクステンションは、ステップやコネクターを Fuse Online に提供しません。

インテグレーションを保存するときに、必要の応じて、インテグレーションと含まれるようにするインポートされたライブラリーエクステンションを 1 つまたは複数選択できます。

ライブラリーエクステンションは、いかなるアクションも定義しません。以下は、ライブラリーエクステンションの定義例になります。

```
{
  "schemaVersion" : "v1",
  "name" : "Example Library Extension",
  "description" : "Syndesis Extension for adding a runtime library",
  "extensionId" : "io.syndesis.extensions:syndesis-library",
  "version" : "1.0.0",
```

```
  "tags" : [ "my-libraries-extension" ],
  "extensionType" : "Libraries"
}
```

ライブラリーエクステンションの例は https://github.com/syndesisio/syndesis-extensions でも参照できます。

ライブラリーエクステンションの構造は、アクションがないこと以外は、ステップまたはコネクターエクステンションの構造と同じです。

ライブラリーエクステンションを作成する Maven プロジェクトで、Maven リポジトリーから使用できない依存関係を追加するには、以下のように **system** 依存関係を指定します。

```
<dependency>
  <groupId>com.company</groupId>
  <artifactId>my-library-extension</artifactId>
  <version>1.0</version>
  <scope>system</scope>
  <systemPath>${project.basedir}/lib/my-library-extension.jar</systemPath>
</dependency>
```

### 9.3.11. JDBC ドライバーライブラリーエクステンションの作成

Apache Derby、MySQL、および PostgreSQL 以外の SQL データベースに接続するには、接続するデータベースの JDBC ドライバーをラップするライブラリーエクステンションを作成します。Fuse Online にこのエクステンションをアップロードした後、Fuse Online が提供する **Database** コネクターはドライバーにアクセスしてプロプライエタリーデータベースへのコネクションを検証および作成することができます。特定のデータベースに新しいコネクターを作成しません。

Syndesis オープンソースコミュニティーは、JDBC ドライバーをラップするエクステンションを作成するためのプロジェクトを提供します。

エクステンションで 1 つのドライバーのみをパッケージ化します。これにより、特定のデータベースを管理する一環として、エクステンションの管理が容易になります。ただし、複数のドライバーをラップするライブラリーエクステンションを作成することは可能です。

#### 前提条件

Syndesis プロジェクトを使用するには、GitHub アカウントが必要です。

#### 手順

1. 以下のいずれかを行って、接続するデータベースの JDBC ドライバーへアクセスできるようにします。
   a. ドライバーが Maven リポジトリーにあることを確認します。
   b. ドライバーをダウンロードします。
2. ブラウザーのタブで https://github.com/syndesisio/syndesis-extensions にアクセスします。
3. **syndesis-extensions** リポジトリーを GitHub アカウントにフォークします。
4. フォークからローカルのクローンを作成します。
5. **syndesis-extensions** のクローンで以下を行います。

a. このドライバーが Maven リポジトリーにない場合、**syndesis-library-jdbc-driver/lib** フォルダーにコピーします。

b. **syndesis-library-jdbc-driver/pom.xml** ファイルを編集します。

i. **Name** 要素の値を更新し、このエクステンションに選択する名前にします。

ii. **Description** 要素の値を更新し、このエクステンションに関する便利な情報を提供します。

iii. ドライバーを **syndesis-library-jdbc-driver/lib** にコピーした場合、**pom.xml** の **systemPath** がそのドライバーファイルを示すようにしてください。必要に応じて、**groupId**、**artifactId**、および **version** を変更し、ドライバーに応じた適切な値を反映するようにします。

iv. このドライバーが Maven リポジトリーにある場合は、その Maven 依存関係への参照が **pom.xml** ファイルにあることを確認します。

v. **pom.xml** ファイルの残りの内容を確認し、必要に応じて関連するメタデータを変更します。

c. **./mvnw -pl :syndesis-library-jdbc-driver clean package** を実行し、エクステンションをビルドします。

生成された **.jar** ファイルは **syndesis-library-jdbc-driver/target** フォルダーにあります。この **.jar** ファイルを Fuse Online でエクステンションとしてインポートします。

ライブラリーエクステンションのインポート後、Fuse Online にインテグレーションを保存するときに、必要に応じて、インポートされたライブラリーエクステンションを選択し、インテグレーションに関連付けることができます。

## 9.4. エクステンションの追加および管理

エクステンションを使用すると、Fuse Online にカスタマイズを追加できるため、アプリケーションを好きなように統合できます。エクステンションで提供されるカスタマイズを使い始めた後、これらのカスタマイズを使用するインテグレーションを特定できます。これは、エクステンションを更新または削除する前に便利です。

詳細は以下のセクションを参照してください。

- 「カスタム機能の使用」
- 「エクステンションを使用するインテグレーションの特定」
- 「エクステンションの更新」
- 「エクステンションの削除」

### 9.4.1. カスタム機能の使用

インテグレーションでカスタム機能を使用できるようにするには、エクステンションを Fuse Online にアップロードします。

**前提条件**

- 開発者によって、Fuse Online エクステンションが含まれる **.jar** ファイルが提供済みである必要があります。

**手順**

1. Fuse Online の左パネルで **Customizations** > **Extensions** をクリックします。
2. **Import Extension** をクリックします。
3. アップロードするエクステンションが含まれる **.jar** ファイルをドラッグアンドドロップまたは選択します。
   Fuse Online は、ファイルにエクステンションが含まれることを即座に検証しようとします。問題がある場合、Fuse Online はエラーに関するメッセージを表示します。エクステンションの開発者と協力して、更新された **.jar** ファイルを取得する必要があります。取得後、そのファイルをアップロードします。
4. エクステンションの詳細を確認します。
   Fuse Online はファイルの検証後、エクステンションの名前、ID、説明、およびタイプを抽出および表示します。タイプは、カスタムコネクター、コネクション間のデータを操作する 1 つ以上のカスタムステップ、またはランタイムライブラリーエクステンション (JDBC ドライバーを含む) がエクステンションによって定義されるかどうかを示します。

   コネクターエクステンションの場合、Fuse Online はこのカスタムコネクターから作成されたコネクションで使用できるアクションを表示します。エクステンションでは、開発者は、このコネクターから作成されたアプリケーションコネクションを表すために Fuse Online が使用するアイコンを提供することがあります。このアイコンは、エクステンションの詳細ページには表示されませんが、カスタムコネクターからコネクションを作成すると表示されます。エクステンションの開発者がエクステンションのアイコンを指定しなかった場合、Fuse Online はアイコンを生成します。

   ステップエクステンションでは、Fuse Online にエクステンションによって定義される各カスタムステップの名前が表示されます。

   ライブラリーエクステンションでは、インポートされた Maven 依存関係がインテグレーションランタイムクラスパスに含まれます。インポートされた Maven 依存関係が、インテグレーションですでに使用されている他の依存関係と競合しないようにする必要があります (JDBC ドライバーなどの他のライブラリーエクステンションを含む)。
5. **Import Extension** をクリックします。Fuse Online はカスタムコネクターまたはカスタムステップを使用できるようにし、エクステンションの詳細ページを表示します。

**その他のリソース**

- カスタムコネクターからのコネクションの作成
- カスタムステップの追加
- プロプライエタリーデータベースへの接続

### 9.4.2. エクステンションを使用するインテグレーションの特定

エクステンションを更新または削除する前に、そのエクステンションが提供するカスタマイズを使用するインテグレーションを特定する必要があります。

**手順**

1. Fuse Online の左パネルで **Customizations** > **Extensions** をクリックします。
2. エクステンションの一覧で、更新または削除するエクステンションのエントリーを見つけ、**Details** ボタンをクリックします。

**結果**

Fuse Online は、エクステンションに関する詳細を表示します。これには、エクステンションが提供するカスタマイズを使用するインテグレーションのリストが含まれます。

### 9.4.3. エクステンションの更新

開発者がエクステンションを更新する場合、更新された **.jar** ファイルをアップロードし、インテグレーションで更新を実装することができます。

**前提条件**

開発者は、以前にアップロードしたエクステンションの更新された **.jar** ファイルを提供する必要があります。

**手順**

1. Fuse Online の左側のパネルで **Customizations** > **Extensions** をクリックします。
2. 更新するエクステンションのエントリーの右側にある **Update** をクリックします。
3. 点線のボックスをクリックして、更新された **.jar** ファイルに移動および選択し、**Open** をクリックします。
4. エクステンションの詳細が正しいことを確認し、**Import Extension** をクリックします。
5. 更新されたエクステンションの詳細ページで、エクステンションに定義されたコネクターまたはカスタムステップを使用するインテグレーションを特定します。

更新されたエクステンションからカスタムコネクターまたはカスタムステップを使用する各インテグレーションを更新するのに何が必要であるかを確認するのはユーザー次第です。少なくとも、更新されたエクステンションに定義されたカスタマイズを使用する各インテグレーションを再パブリッシュする必要があります。

インテグレーションを編集し、カスタマイズの設定詳細を変更または追加する必要が場合があります。インテグレーションの更新方法を理解するには、エクステンションの開発者と対話する必要があります。

### 9.4.4. エクステンションの削除

実行中のインテグレーションがエクステンションによって提供されるステップを使用したり、エクステンションによって提供されたコネクターから作成されたコネクションを使用しても、そのエクステンションを削除できます。エクステンションの削除後に、そのエクステンションによって提供されたカスタマイズを使用するインテグレーションを開始することはできません。

**手順**

1. Fuse Online の左パネルで **Customizations** > **Extensions** をクリックします。
2. エクステンションの一覧で、削除するエクステンションのエントリーを見つけ、エントリーの右側に表示される **Delete** をクリックします。

### 結果

削除するエクステンションによって提供されるカスタマイズを使用する、停止したインテグレーションまたは下書きのインテグレーションが存在する可能性があります。このようなインテグレーションを実行するには、インテグレーションを編集して、カスタマイズを削除する必要があります。

### その他のリソース

- インテグレーションの使用を特定
- エクステンションの更新