Red Hat Fuse 7.1

Integrating Applications with Fuse Online

User's guide to integrating applications with Fuse Online

Last Updated: 2020-03-11

Red Hat Fuse 7.1 Integrating Applications with Fuse Online

User's guide to integrating applications with Fuse Online

Legal Notice

Abstract

Fuse Online provides integration as a service.

Table of Contents

PREFACE

Red Hat Fuse is a distributed, cloud-native integration solution that provides choices for architecture, deployment, and tools. Fuse Online is Red Hat's web-based Fuse distribution. Syndesis is the open source project for Fuse Online. Fuse Online runs on OpenShift Online, OpenShift Dedicated, and OpenShift Container Platform.

This guide provides information and instructions for using Fuse Online's web interface to integrate applications. The content is organized as follows:

- Chapter 1, *High level overview of Fuse Online*
- Chapter 2, *How to get ready to create integrations*
- Chapter 3, *What to expect when using Fuse Online for the first time*
- Chapter 4, *Connecting to applications that you want to integrate*
- Chapter 5, *Creating integrations*
- Chapter 6, *Mapping integration data to fields for the next connection*
- Chapter 7, *Customizing Fuse Online*
- Chapter 8, *Managing integrations*
- Chapter 9, *Installing Fuse Online on OpenShift Container Platform*

To learn how to use Fuse Online by creating sample integrations, see the sample integration tutorials.

To obtain support, in Fuse Online, in the upper right, click ℹ and then select **Support**.

CHAPTER 1. HIGH LEVEL OVERVIEW OF FUSE ONLINE

With Fuse Online, you can obtain data from an application or service, operate on that data if you need to, and then send the data to a completely different application or service. No coding is required to accomplish this.

The following topics provide a high-level overview of Fuse Online:

- Section 1.1, “How Fuse Online works”
- Section 1.2, “Who Fuse Online is for”
- Section 1.3, “Benefits of using Fuse Online”
- Section 1.4, “Descriptions of Fuse Online constructs”

1.1. HOW FUSE ONLINE WORKS

Fuse Online provides a web browser interface that lets you integrate two or more different applications or services without writing code. It also provides features that allow you to introduce code if it is needed for complex use cases.

Fuse Online lets you enable data transfer between different applications. For example, a business analyst can use Fuse Online to capture tweets that mention customers and then leverage the data obtained from Twitter to update Salesforce accounts. Another example is a service that makes stock trade recommendations. You can use Fuse Online to capture recommendations for buying or selling stocks of interest and forward those recommendations to a service that automates stock transfers.

To create and run a simple integration, the main steps are:

1. Create a connection to each application that you want to integrate.
2. Select the start connection. This connection is to the application that contains the data that you want to share with another application.
3. Select the finish connection. This connection is to the application that receives data from the start connection and that completes the integration.
4. Map data fields from the start connection to data fields in the finish connection.
5. Give the integration a name.
6. Click **Publish** to start running the integration.

The Fuse Online dashboard lets you monitor and manage integrations. You can see which integrations are running, and you can start, stop, and edit integrations.

1.2. WHO FUSE ONLINE IS FOR

Fuse Online is for business experts in, for example, finance, human resources, or marketing, who do not want to write code in order to share data between two different applications. Their use of a variety of software-as-a-service (SaaS) applications gives them an understanding of business requirements, workflows, and relevant data.

As a business user, you can use Fuse Online to:

- Capture tweets that mention your company, filter them, and create new contacts in your Salesforce environment when the tweet is from an unknown source.
- Identify Salesforce lead updates and then execute a SQL stored procedure to keep your related database up to date.
- Subscribe for orders received by an AMQ broker and then operate on those orders with a custom API.
- Obtain data from an Amazon S3 bucket and add it to a Dropbox folder.

These are just a few examples of what a business user can do without writing code.

1.3. BENEFITS OF USING FUSE ONLINE

Fuse Online has a number of benefits:

- Integrate data from different applications or services without writing code.
- Run the integration on OpenShift Online in the public cloud or on OpenShift Container Platform on site.
- Use the visual data mapper to map data fields in one application to data fields in another application.
- Leverage all the benefits of open source software. You can extend features, and customize interfaces. If Fuse Online does not provide a connector for an application or service that you want to integrate then a developer can create the connector that you need.

1.4. DESCRIPTIONS OF FUSE ONLINE CONSTRUCTS

To use Fuse Online, you create an integration by working with connectors, connections, actions, and steps. The following sections describe these constructs.

- Section 1.4.1, “About integrations”
- Section 1.4.2, “About Fuse Online connectors”
- Section 1.4.3, “About Fuse Online connections”
- Section 1.4.4, “About actions performed by connections”
- Section 1.4.5, “About integration steps”

1.4.1. About integrations

An integration is a set of ordered steps that Fuse Online executes. This set includes:

- A step that connects to an application to start the integration. This connection provides the initial data that the integration operates on. A subsequent connection can provide additional data.
- A step that connects to an application to complete the integration. This connection receives any data that was output from previous steps and finishes the integration.

- Optional additional steps that connect to applications between the start and finish connections. Depending on the position of the additional connection in the sequence of integration steps, an additional connection can do any or all of the following:
  - Provide additional data for the integration to operate on
  - Process the integration data
  - Output processing results to the integration
- Optional steps that operate on data between connections to applications. Typically, there is a step that maps data fields from the previous connection to data fields that the next connection uses.

In an integration, each step can operate on the data that is output from the previous steps. To determine the steps that you need in an integration, see Section 2.1, “Planning integrations”.

1.4.2. About Fuse Online connectors

Fuse Online provides a set of connectors. A connector represents a specific application that you want to obtain data from or send data to. Each connector is a template for creating a connection to that specific application. For example, you use the Salesforce connector to create a connection to Salesforce.

An application that you want to connect to might use the OAuth protocol to authenticate users. In this case, you register your Fuse Online environment as a client that can access that application. The registration is associated with the connector for that application. You need to register a particular Fuse Online environment only once with each application that uses OAuth. The registration extends to each connection that you create from that connector.

If Fuse Online does not provide a connector you need, a developer can create the required connector.

1.4.3. About Fuse Online connections

Before you can create an integration, you must create a connection to each application or service that you want to obtain data from or send data to. To create a connection, you select a connector and add configuration information. For example, to connect to an AMQ broker in an integration, you create a connection by selecting the AMQ connector, and then following prompts to identify the broker to connect to and the account to use for the connection.

A connection is one specific instance of the connector that it is created from. You can create any number of connections from one connector. For example, you can use the AMQ connector to create three AMQ connections where each connection accesses a different broker.

To create an integration, you select a connection to start the integration, a connection to end the integration, and optionally one or more connections for accessing additional applications. Any number of integrations can use the same connection. A particular integration can use the same connection more than once.

For details, see Chapter 4, *Connecting to applications that you want to integrate* .

1.4.4. About actions performed by connections

In an integration, each connection performs one action. As you create an integration, you choose a connection to add to the integration and then you choose the action that the connection performs. For example, when you create an integration that uses a Salesforce connection, you choose from a set of actions that includes, but is not limited to, creating a Salesforce account, updating a Salesforce account, and searching Salesforce.

Some actions require additional configuration and Fuse Online prompts you for this information.

1.4.5. About integration steps

An integration is a set of ordered steps. Each step operates on data. Some steps operate on data while connected to an application or service outside Fuse Online. These steps are connections. Between connections, there can be other steps that operate on data in Fuse Online. Typically, an integration includes a step that maps data fields used in a connection to data fields used in the next connection. Except for the start connection, each step operates on data it receives from the previous steps.

To operate on data between connections, Fuse Online provides steps for:

- Mapping data fields in one application to data fields in another application.
- Filtering data so that the integration continues only when it meets criteria that you define.
- Logging information in addition to the default logging that Fuse Online automatically provides.

See also Section 2.2, “General workflow for creating an integration” and Section 5.3, “Adding steps between connections”.

To operate on data between connections in a way that is not built into Fuse Online, you can upload an extension that provides a custom step. See Section 7.4, “Adding and managing extensions” .

CHAPTER 2. HOW TO GET READY TO CREATE INTEGRATIONS

Some planning and an understanding of the workflow for creating an integration can help you create integrations that meet your needs. The following topics provide information for getting ready to create integrations.

2.1. PLANNING INTEGRATIONS

Consider the following questions before you create an integration.

To start the integration:

- Which application should the integration obtain data from?
- In that application, what triggers the action that obtains the data? For example, an integration that starts by obtaining data from Twitter might trigger on a Twitter mention.
- What are the data fields of interest?
- What credentials does Fuse Online use to access this application?

To finish the integration:

- Which application receives the data?
- In that application, what action does the integration perform?
- What are the data fields of interest?
- What credentials does Fuse Online use to access this application?

Between the start and finish applications:

- Do you need to access any other applications? For any other applications the integration accesses:
  - Which application does the integration need to connect to?
  - What action should the integration perform?
  - What are the data fields of interest?
  - What credentials should the integration use to connect to this application?
- Does the integration need to operate on the data between connections? For example:
  - Should the integration filter the data it operates on?
  - Do field names differ between source and target applications? If they do then data mapping is required.

- Does the integration need to operate on the data in some customized way?

2.2. GENERAL WORKFLOW FOR CREATING AN INTEGRATION

After you log in to the Fuse Online console, the general steps for integrating applications are:

1. For each application that you want to integrate and that uses the OAuth protocol, register Fuse Online as a client of that application.
2. For each application that you want to integrate, create a connection.
3. Create the integration:
   a. Select the start connection. This connection starts the integration by accessing the application you want to obtain data from.
   b. Select the action that you want the start connection to perform.
   c. Optionally, depending on the connection, enter some configuration information, for example, you might indicate whether to operate on a Salesforce contact or a Salesforce lead.
   d. Select the finish connection. This connection completes the integration by accessing the application that uses the data from the start connection.
   e. Select the action you want the finish connection to perform.
   f. Optionally, depending on the connection, enter some configuration details.
   g. Optionally, between the start connection and the finish connection, add one or more connections to other applications.
   h. Add a data mapping step between connections that have different field names for the same data. Typical integrations require some data mapping.
   i. Optionally, between connections, add additional steps, such as filtering data or logging that is in addition to the automatically-provided logging.
4. Click **Publish** to start running your integration.

2.3. EXAMPLE WORKFLOW FOR CREATING A SALESFORCE TO DATABASE INTEGRATION

The best way to understand the workflow for using Fuse Online is to create the sample integrations by following the instructions in the sample integration tutorials. If you have already done that, you can skip this section.

The following abbreviated description of one of the samples provides an example workflow for using Fuse Online. These steps omit details, so you should not try to follow them.

1. Register your installation of Fuse Online as an application that can access Salesforce. You need to do this only once to be able to create any number of integrations that connect to Salesforce.
2. Create a Salesforce connection. To configure this connection, Fuse Online prompts you to log in to the Salesforce account you used to register Fuse Online. You can use the same Salesforce connection in any number of integrations.

3. Choose your Salesforce connection as the connection that starts the integration.

4. Choose the action that you want the Salesforce connection to perform. In the sample integration, you choose the **On create** action for the **Lead** object. This means that after connecting to Salesforce, the integration watches for notifications that a Salesforce lead was created. When the integration finds such a notification, it passes the new lead’s data to the next step in the integration. However, before you can add the next step, you must choose the integration’s finish connection.

5. Choose the **PostgresDB** connection as the connection that completes the integration.

6. Choose the action that you want the **PostgresDB** connection to perform. In the sample integration, you choose **add_lead** as the procedure you want to invoke. This is a provided stored procedure that runs in the sample database. This procedure determines the requirements for mapping Salesforce data fields to database fields.

7. Add a step between the Salesforce connection and the database connection. This step maps Salesforce data fields to database fields.

8. Give the integration a name. Optionally, enter a description of what the integration does.

9. Click **Publish** to start running the integration.

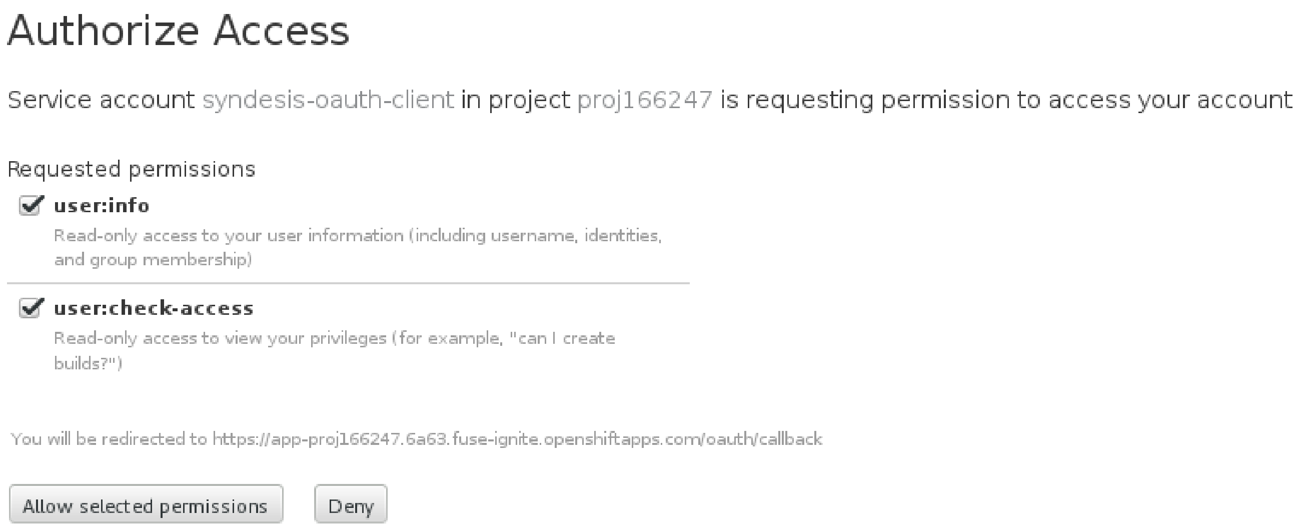
10. On the Fuse Online dashboard, confirm that the Salesforce to database integration is designated as **Running**, which means that it is running.

11. Confirm that the integration is working as expected by creating a new lead in Salesforce.

12. For this sample integration, in a browser, insert **todo-** in front of the URL for your Fuse Online installation. This displays the notification that a new lead was created in the database.

CHAPTER 3. WHAT TO EXPECT WHEN USING FUSE ONLINE FOR THE FIRST TIME

To use Fuse Online on OpenShift Online, Red Hat provides a link. Clicking this link displays the **Red Hat OpenShift Online Log In** page, which prompts you to log in by using your Red Hat account. Logging in prompts you to authorize Fuse Online to access to your account:



Authorize Access

Service account syndesis-oauth-client in project proj166247 is requesting permission to access your account

Requested permissions

☑ **user:info**
Read-only access to your user information (including username, identities, and group membership)

☑ **user:check-access**
Read-only access to view your privileges (for example, "can I create builds?")

You will be redirected to https://app-proj166247.6a63.fuse-ignite.openshiftapps.com/oauth/callback

Click **Allow selected permissions**. You need to do this only once. The next time you click the Fuse Online access link, Fuse Online immediately appears.

To use Fuse Online on OpenShift Container Platform, follow the installation instructions, which are at the end of Integrating Applications with Fuse Online.

Red Hat supports using Fuse Online in the following browsers:

- Chrome
- Firefox
- Microsoft Edge

CHAPTER 4. CONNECTING TO APPLICATIONS THAT YOU WANT TO INTEGRATE

To integrate applications:

1. Create a connection to each application or service that you want to integrate.
2. Create an integration.
3. Add connections to the integration.

The procedure for creating a connection varies for each application or service. The details for creating each kind of connection are in Connecting Fuse Online to Applications and Services .

The following topics provide an overview of the workflow:

- Section 4.1, “About creating connections from Fuse Online to applications”
- Section 4.2, “Obtaining authorization to access applications”
- Section 4.3, “About adding connections to integrations”
- Section 4.4, “Viewing and editing connection information”
- Section 4.5, “Creating connections from custom connectors”
- Section 4.6, “Specifying connection action input or output types”

4.1. ABOUT CREATING CONNECTIONS FROM FUSE ONLINE TO APPLICATIONS

To create a connection, you select the connector for the application that you want to connect to and then enter values in input fields to configure a connection to that application. The configuration details that you need to provide vary for each application. After configuring the connection, you give it a name that helps you distinguish it from any other connections to the same application. Optionally, you can specify a description of the connection.

You can use the same connector to create any number of connections to that application. For example, you might use the AMQ connector to create three different connections. Each AMQ connection could specify a different broker.

For examples, see:

- Creating AMQ connections
- Creating HTTP and HTTPS connections
- Creating Slack connections

4.2. OBTAINING AUTHORIZATION TO ACCESS APPLICATIONS

In an integration, you might want to connect to an application that uses the OAuth protocol to authenticate access requests. To do this, you must register your installation of Fuse Online for access to that application. Registration authorizes all connections from your Fuse Online installation to a given application. For example, if you register your Fuse Online installation with Salesforce, all connections from your Fuse Online installation to Salesforce use the same client ID and the same client secret.

In each Fuse Online environment, for each application that uses OAuth, only one registration of Fuse Online as a client is required. This registration lets you create multiple connections and each connection can use different user credentials.

For details, see the following topics:

- Section 4.2.1, "General procedure for obtaining authorization"
- Section 4.2.2, "About connection validation"

For information about using custom connectors that let you access applications that use the OAuth protocol, see Section 4.5, "Creating connections from custom connectors" .

4.2.1. General procedure for obtaining authorization

To integrate applications that use OAuth, you must register with that application before you can create a connection to the application. For example, after you register your installation of Fuse Online as an application that can access Salesforce, then you can create a Salesforce connection.

While the specific steps vary for each OAuth application that you want to connect to, registration always provides your installation of Fuse Online with a client ID and a client secret. Some applications use other labels for the client ID and client secret. For example, Salesforce generates a consumer key and a consumer secret.

For some OAuth applications, Fuse Online provides an entry in its **Settings** page that makes it easy to register with the application. To see which applications this applies to, in the left panel of Fuse Online, click **Settings**.

For an application that has an entry in the Fuse Online **Settings** page, to register Fuse Online with that application, the main steps are:

1. In the Fuse Online **OAuth Application Management** page, in the entry for the application with which you want to register Fuse Online, click **Register** to display the **Client ID** and **Client Secret** fields.
2. Near the top of the **OAuth Application Management** page, where you see **During registration, enter this callback URL:**, copy that URL to the clipboard.
3. In another browser tab, go to the web site for the application that you want to register with and perform the steps required to obtain a client ID and secret. One of these steps requires you to enter the callback URL for your installation of Fuse Online. Paste the URL that you copied to the clipboard in the second step.
4. On your Fuse Online installation **Settings** page, paste the client ID and client secret and save the settings.

For examples, see

- Registering Fuse Online as a Salesforce client
- Registering Fuse Online as a Twitter client

For an example of registering with an application that does not have an entry in the Fuse Online **Settings** page, see: Registering Fuse Online as a Dropbox client .

4.2.2. About connection validation

After obtaining authorization for Fuse Online to access an application that uses OAuth, you can create one or more connections to that application. When you create a connection to an OAuth application, Fuse Online validates it to confirm that authorization is in place. At any time, you can validate the connection again to ensure that authorization is still in place.

Some OAuth applications grant access tokens that have an expiration data. If the access token expires, you can reconnect to the application to obtain a new access token.

To validate a connection that uses OAuth or to obtain a new access token for an OAuth application:

1. In the left panel, click **Connections**.
2. Click the connection you want to validate or for which you want to obtain a new access token.
3. In the connection's details page, click **Validate** or click **Reconnect**.

If validation or reconnection fails, then check with the application/service provider to determine if the application's OAuth keys, IDs, tokens, or secrets are still valid. It is possible that an item has expired or been revoked.

If you find that an OAuth item is invalid, has expired, or been revoked, obtain new values and paste them into the Fuse Online settings for the application. See the instructions in Connecting Fuse Online to Applications and Services for registering the application whose connection did not validate. With the updated settings in place, follow the instructions above to try to validate the updated connection. If validation is successful, and there is a running integration that is using this connection, restart the integration. To restart an integration, stop it and then start it.

If validation fails and reconnection fails but everything appears to be valid at the service provider, then try reregistering your Fuse Online environment with the application and then recreate the connection. Fuse Online validates the connection when you recreate it. If you recreate the connection, and there is an integration that is using the connection, then you must edit the integration to delete the old connection and add the new connection. If the integration is running, then you must stop it and restart it.

4.3. ABOUT ADDING CONNECTIONS TO INTEGRATIONS

When you add a connection to an integration, Fuse Online displays a list of the actions that the connection can perform when it connects to the application. You must select exactly one action. In a running integration, each connection performs only the action you choose. For example, when you add a Twitter connection as an integration's start connection, you might choose the **Mention** action, which monitors Twitter for tweets that mention your Twitter handle.

Selection of some actions prompts you to specify one or more parameters. For example, if you add a Salesforce connection to an integration and choose the **On create** action then you must indicate the type of object whose creation you are interested in, such as a lead or a contact.

4.4. VIEWING AND EDITING CONNECTION INFORMATION

After you create a connection, Fuse Online assigns an internal identifier to the connection. This identifier does not change. You can change the connection's name, description, or configuration values and Fuse Online recognizes it as the same connection.

There are two ways to view and edit information about a connection:

- In the left panel, click **Connections** and then click any connection to view its details.

- In the left panel, click **Integrations** and then click any integration to view its details. In the **Integration Summary** page, in the flow diagram of the integration, click a connection icon to view that connection's details.

On the **Connection Details** page, for the connection you want to edit, click ✎ next to a field to edit that field. Or, for some connections, below the configuration fields, click **Edit** to change configuration values. If you change any values, be sure to click **Save**.

If you update a connection that is used in an integration that is running, you must republish the integration by stopping it and starting it again.

For connections to applications that use the OAuth protocol to authorize access, you cannot change the login credentials that the connection uses. To connect to the application and use different login credentials, you must create a new connection.

4.5. CREATING CONNECTIONS FROM CUSTOM CONNECTORS

After you upload an extension that defines a custom connector, the custom connector is available for use. You use custom connectors to create connections in the same way that you use Fuse Online-provided connectors to create connections.

A custom connector might be for an application that uses the OAuth protocol. Before you create a connection from this kind of connector, you must register your installation of Fuse Online for access to the application that the connector is for. You do this in the interface for the application that the connector is for. The details for how to register your installation of Fuse Online vary for each application.

For example, suppose the custom connector is for creating connections to Yammer. You would need to register your installation of Fuse Online by creating a new application within Yammer. Registration provides a Yammer client ID for Fuse Online and a Yammer client secret value for Fuse Online A connection from your Fuse Online installation to Yammer must provide these two values.

Note that an application might use different names for these values, such as consumer ID or consumer secret.

After you register your installation of Fuse Online, you can create a connection to the application. When you configure the connection, there should be parameters for entering the client ID and the client secret. If these parameters are not available, you need to talk with the extension developer and ask for an updated extension that lets you specify the client ID and client secret.

For more information, see Chapter 4, *Connecting to applications that you want to integrate* .

4.6. SPECIFYING CONNECTION ACTION INPUT OR OUTPUT TYPES

To process data from the start connection through the finish connection, sometimes you need to specify input/output types when you configure a connection's action. Type specifications let Fuse Online alert you when a data mapping step is required. A data mapping step ensures that the next integration step can process the data it receives.

After you configure an Amazon S3, AMQ, AMQP, Dropbox, FTP/SFTP, or HTTP/HTTPS connection, Fuse Online prompts you to specify input and/or output data types as follows:

1. In the **Select Type** field, if the data type does not need to be known, accept **Type specification not required** and then, at the bottom, click **Done**. You do not need to follow the rest of these instructions.
   Otherwise, select one of the following as the schema type:

- **JSON schema** is a document that describes the structure of JSON data. The document's media type is **application/schema+json**.
- **JSON instance** is a document that contains JSON data. The document's media type is **application/json**.
- **XML schema** is a document that describes the structure of XML data. The document's file extension is **.xsd**.
- **XML instance** is a document that contains XML data. The document's file extension is **.xml**.

2. In the **Definition** input box, paste a definition that conforms to the schema type you selected. For example, if you select **JSON schema** then you would paste the content of a JSON schema file, which has a media type of **application/schema+json**.

3. In the **Data Type Name** field, enter a name that you choose for the data type. For example, suppose you are specifying a JSON schema for vendors. You can specify **Vendor** as the data type name.
   You will see this data type name when you are creating or editing an integration that uses the connection for which you are specifying this type. Fuse Online displays the type name in the integration visualization panel and in the data mapper.

4. In the **Data Type Description** field, provide information that helps you distinguish this type. This description appears in the data mapper when you hover over the step that processes this type.

5. Click **Done**.

The connection appears in the integration flow in the location where you added it.

CHAPTER 5. CREATING INTEGRATIONS

After some planning and preparation, you are ready to create an integration. In the Fuse Online web interface, when you click **Create Integration**, Fuse Online guides you through the procedure to create an integration.

It is assumed that you are familiar with the information in these topics:

- Section 2.1, “Planning integrations”
- Section 2.2, “General workflow for creating an integration”

The following topics provide information and instructions for creating an integration:

- Section 5.1, “Preparing to create an integration”
- Section 5.2, “Procedure for creating an integration”
- Section 5.3, “Adding steps between connections”

5.1. PREPARING TO CREATE AN INTEGRATION

Preparation for creating an integration starts with answers to the questions listed in Section 2.1, “Planning integrations”. After you have a plan for the integration, you need to do the following before you can create the integration:

1. Determine whether an application that you want to connect to uses the OAuth protocol. For each application that uses OAuth, register Fuse Online as a client that is authorized to access that application. See Section 4.2, “Obtaining authorization to access applications” .

2. Determine whether an application that you want to connect to uses HTTP basic authentication. For each application that does, identify the user name and password for accessing that application. You need to provide this information when you create the integration.

3. For each application that you want to integrate, create a connection. See Section 4.1, “About creating connections from Fuse Online to applications”.

5.2. PROCEDURE FOR CREATING AN INTEGRATION

To create an integration:

1. In the left panel in Fuse Online, click **Integrations**.

2. In the upper right, click **Create Integration**.

3. Choose and configure the start connection:

   a. On the **Choose a Start Connection** page, click the connection you want to use to start the integration. When this integration is running, Fuse Online will connect to this application and obtain data that you want the integration to operate on.

   b. On the **Choose an Action** page, click the action you want this connection to perform. The available actions vary for each connection.

   c. On the page for configuring the action, enter values in the fields.

   d. Optionally, if the connection requires data type specification, Fuse Online prompts you to click **Next** to specify the input and/or output type of the action.

   e. Click **Done** to add the start connection.

4. Choose and configure the finish connection:

   a. On the **Choose a Finish Connection** page, click the connection you want to use to complete the integration. When this integration is running, Fuse Online will connect to this application with the data that the integration has been operating on.

   b. On the **Choose an Action** page, click the action you want this connection to perform. The available actions vary for each connection.

   c. On the page for configuring the action, enter values in the fields.

   d. Optionally, if the connection requires data type specification, Fuse Online prompts you to click **Next** to specify the input and/or output type of the action.

   e. Click **Done** to add the finish connection.

5. Optionally, add one or more connections between the start connection and the finish connections. For each connection, choose its action and enter any required configuration details.

6. Optionally, add one or more steps that operate on integration data between connections. See Section 5.3, "Adding steps between connections" .

7. When the integration contains all needed steps, click **Save as Draft** or **Publish** according to whether you want to start running the integration.

8. In the **Integration Name** field, enter a name that distinguishes this integration from any other integrations.

9. In the **Description** field, enter a description, for example, you can indicate what this integration does.

10. If you are ready to start running the integration, in the upper right, click **Publish**. Otherwise, click **Save as Draft**

In the Fuse Online **Integrations** page, you can see your new integration in the list of integrations. If you published the integration, then you can see that Fuse Online is in the process of publishing it. It may take a few moments for the status of the integration to become **Running**. If you saved the integration as a draft, then **Draft** appears on the integration's entry.

Click the integration's entry to see a summary of the integration, including its version history, logs for each execution, and aggregate execution metrics.

5.3. ADDING STEPS BETWEEN CONNECTIONS

After you add connections to an integration, you can optionally add steps between connections. Each step operates on data obtained from the previous connection(s) and any other previous steps and makes the data available to the next step in the integration.

Often, you must map data fields that are received from a connection to data fields that the next connection in the integration can operate on. After you add all connections to your integration, check the integration visualization panel on the left. For each connection that requires data mapping before it can operate on the input data, Fuse Online displays ⚠. Click this icon to see **Data Type Mismatch: Add a data mapping step before this connection to resolve the difference.** Click the link in the message to display the **Configure Mapper** page in which you specify the data mapping step.

For details about adding additional steps, see the following topics:

- Section 5.3.1, “Adding a data mapping step”
- Section 5.3.2, “Adding a basic filter step”
- Section 5.3.3, “Adding an advanced filter step”
- Section 5.3.4, “Adding a log step”
- Section 5.3.5, “Adding a custom step”

5.3.1. Adding a data mapping step

You can add a step to an integration to map data from the previous connection(s) and any other steps to the next connection. For example, suppose the integration data contains a **Name** field and the next connection in the integration has a **CustomerName** field. You need to map the source **Name** field to the target **CustomerName** field.

When you add a data mapper step you might be creating a new integration or editing an integration. The flow of the integration appears in the left panel.

To add a data mapper step:

1. In the left panel, where you want to add a data mapper step, hover over the ✚.
2. In the popup that appears, click **Add a Step**.
3. On the **Choose a Step** page, click **Data Mapper** to display source and target fields.

For details about configuring the data mapping step, see Chapter 6, *Mapping integration data to fields for the next connection*.

5.3.2. Adding a basic filter step

You can add a step to an integration to filter the data that the integration operates on. In a filter step, Fuse Online inspects the data and continues the integration only if the content meets criteria that you define. For example, in an integration that obtains data from Twitter, you can specify that you want to continue the integration by operating only on tweets that contain "Red Hat".

Add all connections to your integration before you add additional steps. When you add a step, Fuse Online operates on the data it receives from the previous connection(s) and any other previous step(s).

If you cannot define the filter you need in a basic filter step, see Section 5.3.3, “Adding an advanced filter step”.

You can add a step when you are creating an integration or editing an integration. The flow of the integration appears in the left panel. To add a filter step:

1. In the left panel, where you want to add a filter step to the integration, hover over the ➕ and in the popup that appears, click **Add a Step**.

2. On the **Add a Step** page, click **Basic Filter**.

3. On the **Configure Basic Filter Step** page, in the **Continue only if incoming data match**field, select one of the following options:

   - Accept the default that all defined rules must be satisfied.

   - Indicate that only one rule must be satisfied by selecting **ANY of the following**.

4. Define the filter rule:

   a. **For this field**: In the field on the left, enter the name of the field that contains the content you want the filter to evaluate. For example, suppose the data coming in to the step consists of tweets that mention your Twitter handle. You want to continue the integration only when the tweet contains certain content. The tweet is in a field named **text** so you enter or select **text** as the value in the first field.
   You can define the field value in the following ways:

      - Start typing. The data name field has a typeahead feature that provides a list of possible completions for you in a pop-up box. Select the correct one from the box.

      - Click in the **text** field. A dropdown box appears with a list of available fields. Select the field of interest from the list.

   b. **This condition must be satisfied** In the middle field, select a condition from the dropdown box. The setting defaults to **Contains**. For the integration to continue, the condition that you select in this field must be true for the value that you enter in the third field.

   c. **For this value**: In the third field, enter a value to filter on. For example, if you want to operate on mentions of a certain product from the Twitter feed, you would enter the product name here.

5. Optionally, click **+ Add another rule** and define another rule.
   You can delete a rule by clicking the trash can icon next to the entry.

6. When the filter step is complete, click **Done** to add it to the integration.

5.3.3. Adding an advanced filter step

In a filter step, Fuse Online inspects the data and continues the integration only if the content meets criteria that you define. If the basic filter step does not let you define the exact filter you need, then add an advanced filter step.

Add all connections to your integration before you add additional steps. When you add a step, Fuse Online operates on the data it receives from the previous connections and any additional step(s).

When you add a step you might be creating a new integration or editing an integration. The flow of the integration appears in the left panel.

To add an advanced filter step:

1. In the left panel, where you want to add an advanced filter step to the integration, hover over the ➕ and in the popup that appears, click **Add a Step**.
2. Select **Advanced Filter**.
3. In the edit box, use the Camel Simple Expression language to specify a filter expression. For example, the following expression evaluates to true when the message header's **type** field is set to **widget**:

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

   In the following example, the expression evaluates to true when the body of the message contains a **title** field:

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

4. Click **Done** to add the advanced filter step to the integration.

5.3.4. Adding a log step

Fuse Online provides log information for each integration version that it executes. To learn what information is automatically logged, see Section 8.4.2, "Viewing integration log information" .

To log additional information between any two steps, add a log step to the integration. For each message that it receives, a log step can provide one or more of the following:

- The message's header, which provides metadata about the message.
- The message's body, which provides the content of the message.
- Text that you specify either explicitly or through evaluation of an Apache Camel Simple language expression.

To add a log step, you must be creating a new integration or editing an integration. The integration must already have its start and finish connections.

To add a log step:

1. In the integration visualization panel on the left, hover over the ➕ at the location where you want to add a log step.
2. In the popup, click **Add a Step**.
3. On the **Choose a Step** page, click **Log**.
4. On the **Configure Log Step** page, select the content that you want to log. If you select **Custom text**, then in the text input field, enter one of the following:

   - The text that you want to log
   - A Camel Simple language expression

   If you enter an expression, Fuse Online resolves the expression and logs the resulting text.

5. When log step configuration is complete, click **Done** to add the step to the integration.

6. When the integration is complete, publish it to start seeing output from the new log step.

After an integration that has a log step has been executed, output from its log step appears in the integration's **Activity** tab. See Section 8.4.2, "Viewing integration log information" .

5.3.5. Adding a custom step

If Fuse Online does not provide a step that you need in an integration, a developer can define one or more custom steps in an extension. A custom step operates on integration data between connections. See Section 7.4.1, "Making custom features available" .

You add a custom step to an integration in the same way that you add a built-in step. Create an integration, choose the start and finish connections, add other connections as needed and then add additional steps. When you add a step, Fuse Online operates on the data it receives from the previous step(s).

When you click **Add a Step**, the list of available steps includes any custom steps that are defined in extensions that were uploaded to your installation of Fuse Online.

At the top of the list of steps, in the **Name** field, you can select **Custom Steps** to display only custom steps.

Click the custom step that you want to add to the integration. Fuse Online prompts you for any information required to perform the step. This information varies for each custom step.

CHAPTER 6. MAPPING INTEGRATION DATA TO FIELDS FOR THE NEXT CONNECTION

In most integrations, you need to map data fields that have already been obtained or processed to data fields that the next connection can process. Fuse Online provides a data mapper to help you do this. In an integration, at each point where you need to map data fields, add a data mapper step. Details for mapping data fields are in the following topics:

- Section 6.1, “Identifying where data mapping is needed”
- Section 6.2, “Helpful information for mapping data fields”
- Section 6.3, “Mapping one source field to one target field”
- Section 6.4, “Combining multiple source fields into one target field”
- Section 6.5, “Separating one source field into multiple target fields”
- Section 6.6, “Transforming source or target data”
- Section 6.7, “Descriptions of available transformations”
- Section 6.8, “Viewing the mappings in a step”

6.1. IDENTIFYING WHERE DATA MAPPING IS NEEDED

To identify where data mapping is needed:

1. When you are creating or editing an integration, add all connections to the integration.
2. In the integration visualization panel on the left, look for any ⚠ icons.
3. Click the icon to see the message. A **Data Type Mismatch** notification indicates that you need to add a data mapper step before that connection.

To see the input type or output type for a particular connection:

1. In the Fuse Online left panel, click **Integrations**.
2. In the list of integrations, identify the integration that has the connection whose input type or output type you want to know.
3. At the right of that integration’s entry click ⋮ .
4. Select **Edit**.
5. In the integration’s visualization panel, to the right of a connection, click ℹ to display information about that connection, including its input and/or output type.

6.2. HELPFUL INFORMATION FOR MAPPING DATA FIELDS

In a relatively simple integration, mapping data fields is easy and intuitive. In more complex integrations or integrations that handle large sets of data fields, mapping from source to target is easier when you have some background about how to use the data mapper.

- Section 6.2.1, “Finding the data field that you want to map”
- Section 6.2.2, “Example of missing or unwanted data when combining or separating fields”

6.2.1. Finding the data field that you want to map

In an integration that has a few steps or that operates on a small set of data, it is probably easy to find the data field that you want to map. But in an integration that has many steps, or that has a step that accesses a large set of data, the list of data fields might be very long. The data mapper displays two columns of data fields:

- **Sources** is a list of the data fields that are obtained or processed in all previous steps in the integration.
- **Target** is a list of the data fields that the next connection expects and can process.

To quickly find the data field that you want to map, you can do any of the following:

- Search for it.
  The **Sources** panel and the **Target** panel each have a search field at the top. If the search field is not visible, click 🔍 at the top right of the **Sources** or **Target** panel.
- Enter the names of the fields that you want to map.
  To do this, in the upper right of the **Configure Mapper** page, click the plus sign to display the **Mapping Details** panel. In the **Sources** section, enter the name of the source field. In the **Action** section, accept the default **Map**, which maps one field to one other field. Or, select **Combine** or **Separate**. In the **Target** section, enter the name of the field that you want to map to.
- Expand and collapse folders to limit the visible fields.
  To view the data fields available in a particular step, expand the folder for that step.

  As you add steps to an integration, Fuse Online numbers and renumbers them to indicate the order in which the integration processes the steps. When you are creating or editing an integration, these numbers are visible in the integration visualization panel on the left. When you add a data mapping step, the step numbers appear in the folder labels in the **Sources** panel and in the **Target** panel.

  The folder label also displays the name of the data type that is output by that step. Connections to applications such as Twitter, Salesforce, and SQL define their own data types. For connecting to applications such as Amazon S3, AMQ, AMQP, Dropbox, and FTP/SFTP, you define the connection’s input and/or output type when you add the connection to an integration and select the action that the connection performs. When you specify the data type, you also give the type a name. The type name you specify appears as the name of a folder in the data mapper. If you specified a description when you declared the data type, then the type description appears when you hover over the step folder in the mapper.

6.2.2. Example of missing or unwanted data when combining or separating fields

In a data mapping, you might need to identify missing or unwanted data when a source or target field contains compound data. For example, consider a **long_address** field that has this format:

number street apartment city state zip zip+4 country

Suppose that you want to separate the **long_address** field into discrete fields for **number**, **street**, **city**, **state**, and **zip**. To do this, you select **long_address** as the source field and then select the target fields. You then add padding fields at the locations for the parts of the source field that you do not want. In this example, the unwanted parts are *apartment*, *zip+4*, and *country*.

To identify the unwanted parts, you need to know the order of the parts. The order indicates an index for each part of the content in the compound field. For example, the **long_address** field has 8 ordered parts. Starting at 1, the index of each part is:

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

In the data mapper, to identify *apartment*, *zip+4*, and *country* as missing, you add padding fields at indexes 3, 7, and 8. See Section 6.4, “Combining multiple source fields into one target field” .

Now suppose that you want to combine source fields for **number**, **street**, **city**, **state**, and **zip** into a **long_address** target field. Further suppose that there are no source fields to provide content for *apartment*, *zip+4*, and *country*. In the data mapper, you need to identify these fields as missing. Again, you add padding fields at indexes 3, 7, and 8. See Section 6.5, “Separating one source field into multiple target fields”.

6.3. MAPPING ONE SOURCE FIELD TO ONE TARGET FIELD

The default mapping behavior maps one source field to one target field. For example, map the **Name** field to the **CustomerName** field.

Procedure

1. In the **Sources** panel, click the data field you want to map from.
   You might need to expand an integration step to see the data fields that it provides.

   When there are many source fields, you can search for the field of interest by clicking the 🔍 and entering the name of the data field in the search field.

2. In the **Target** panel, click the data field you want to map to.

The data mapper displays a line that connects the two fields that you just selected.

To confirm that the mapping is defined, in the upper right, click ▦ to display the defined mappings. Click ▦ again to display the data field panels.

Alternative procedure

Here is another way to map a single source field to a single target field:

1. In the **Configure Mapper** page, in the upper right, click the plus sign to display the **Mapping Details** panel.

2. In the **Sources** section, enter the name of the source field.

3. In the **Action** section, accept the default **Map** action.

4. In the **Target** section, enter the name of the field that you want to map to and click **Enter**.

6.4. COMBINING MULTIPLE SOURCE FIELDS INTO ONE TARGET FIELD

In a data mapper step, you can combine multiple source fields into one compound target field. For example, you can map the **FirstName** and **LastName** fields to the **CustomerName** field.

Prerequisite

For the target field, you must know what type of content is in each part of this compound field, the order and index of each part of the content, and the separator between parts, such as a space or comma. See Section 6.2.2, “Example of missing or unwanted data when combining or separating fields” .

Procedure

1. In the **Target** panel, click the field into which you want to map more than one source field.

2. In the **Sources** panel, click the first field that you want to combine into the target field.

3. In the **Sources** panel, for the each of the other fields that you want to combine into the target field, hover over that field, and press **CTRL-Mouse1** (**CMD-Mouse1** on MacOS).
   The data mapper automatically changes the field action from **Map** to **Combine**.

   When you are done you should see a line from each of the source fields to the target field.

4. In the **Mapping Details** panel, in the **Separator** field, accept or select the character that the data mapper inserts in the target field between the content from different source fields. The default is a space.

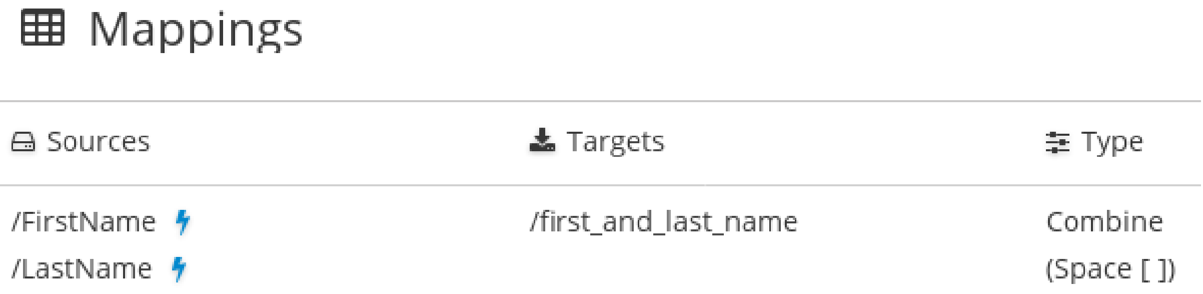
5. In the **Mapping Details** panel, under **Sources**, ensure that the source fields are in the same order as the corresponding content in the compound target field.
   If necessary, drag and drop source fields to achieve the same order. The data mapper automatically updates the index numbers to reflect the new order.

6. If you mapped a source field to each part of the compound target field, then skip to the last step.
   If the target field expects data that is not available to be mapped, then in the **Mapping Details**

panel, edit the index of each source field so that it is the same as the index of the corresponding data in the compound target field. The data mapper automatically adds padding fields as needed to indicate missing data.

If you accidentally create too many padding fields, click the trash-can icon on each extra padding field to delete it.

7. To confirm that the mapping is correctly defined, in the upper right, click ▦ to display the mappings defined in this step. A mapping that combines the values of more than one source field into one target field looks like this:



Mappings

Sources	Targets	Type
/FirstName /LastName	/first_and_last_name	Combine (Space [ ])

.

6.5. SEPARATING ONE SOURCE FIELD INTO MULTIPLE TARGET FIELDS

In a data mapper step, you can separate a compound source field into multiple target fields. For example, map the **Name** field to the **FirstName** and **LastName** fields.

Prerequisite

For the source field, you must know what type of content is in each part of this compound field, the order and index of each part of the content, and the separator between parts, such as a space or comma. See Section 6.2.2, “Example of missing or unwanted data when combining or separating fields” .

Procedure

1. In the **Sources** panel, click the field whose content you want to separate.

2. In the **Target** panel, click the first field that you want to separate the source field data into.

3. In the **Target** panel, for each additional target field that you want to contain some of the data from the source field, hover over the field and press **CTRL-Mouse1** (**CMD-Mouse1** on MacOS) to select it.
The data mapper automatically changes the field action to **Separate**.

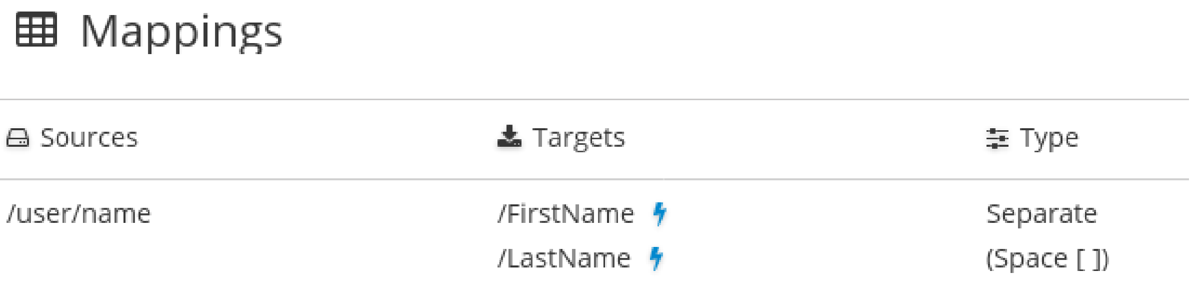
   When you are done selecting target fields, you should see lines from the source field to each of the target fields.

4. In the **Mapping Details** panel, in the **Separator** field, accept or select the character in the source field that indicates where to separate the source field values. The default is a space.

5. In the **Mapping Details** panel, under **Targets**, ensure that the target fields are in the same order as the corresponding content in the compound source field.
If necessary, drag and drop target fields to achieve the same order. The data mapper automatically updates the index numbers to reflect the new order.

6. If you mapped each part of the compound source field to a target field, then skip to the last step.
   If the source field contains data that you do not need, then in the **Mapping Details** panel, edit the index of each target field so that it is the same as the index of the corresponding data in the compound source field. The data mapper automatically adds padding fields as needed to indicate unwanted data.

7. To confirm that the mapping is correctly defined, click ▦ to display the mappings defined in this step. A mapping that separates the value of a source field into multiple target fields looks like this:



▦ Mappings

Sources	Targets	Type
/user/name	/FirstName /LastName	Separate (Space [ ])

.

6.6. TRANSFORMING SOURCE OR TARGET DATA

In the data mapper, after you define a mapping, you can transform any field in the mapping. Transforming a data field defines how you want to store the data. For example, you could specify the **Capitalize** transformation to ensure that the first letter of a data value is uppercase.

To transform a field:

1. Map the fields. This can be a one-to-one mapping, a combination mapping, or a separation mapping.

2. In the **Mapping Details** panel, under **Sources** or under **Targets**, in the box for the field that you want to transform, click the arrow that points to the trash can. This displays a field where you can select the transformation that you want the data mapper to perform.

3. Click in this field to display the list of transformations.

4. Click the transformation that you want to perform.

5. If the transformation requires any input parameters, specify them in the appropriate input fields.

6. To add another transformation, click the arrow that points to the trash can again.

See Section 6.7, “Descriptions of available transformations” .

6.7. DESCRIPTIONS OF AVAILABLE TRANSFORMATIONS

The following table describes the available transformations. The date and number types refer generically to any of the various forms of these concepts. That is, number includes, for example, **integer**, **long**, **double**. Date includes, for example, **date**, **Time**, **ZonedDateTime**.

Transformation	Input Type	Output Type	Parameter (* = required)	Description
AbsoluteValue	number	number	None	Return the absolute value of a number.
AddDays	date	date	**days**	Add days to a date. The default is 0 days.
AddSeconds	date	date	**seconds**	Add seconds to a date. The default is 0 seconds.
Append	string	string	string	Append a string to the end of a string. The default is to append nothing.
Camelize	string	string	None	Convert a phrase to a camelized string by removing whitespace, making the first word lowercase, and capitalizing the first letter of each subsequent word.
Capitalize	string	string	None	Capitalize the first character in a string.
Ceiling	number	number	None	Return the whole number ceiling of a number.
Contains	any	Boolean	**value**	Return true if a field contains the specified value.

Transformation	Input Type	Output Type	Parameter (* = required)	Description
ConvertAreaUnit	number	number	**fromUnit*** **toUnit** *	Convert a number that represents an area to another unit. For the **fromUnit** and **toUnit** parameters, select the appropriate unit from the **From Unit** and **To Unit** menus. The choices are: **Square Foot**, **Square Meter**, or **Square Mile**.
ConvertDistanceUnit	number	number	**fromUnit** * **toUnit** *	Convert a number that represents a distance to another unit. For the **fromUnit** and **toUnit** parameters, select the appropriate unit from the **From Unit** and **To Unit** menus. The choices are: **Foot**, **Inch**, **Meter**, **Mile**, or **Yard**.
ConvertMassUnit	number	number	**fromUnit** * **toUnit** *	Convert a number that represents mass to another unit. For the **fromUnit** and **toUnit** parameters, select the appropriate unit from the **From Unit** and **To Unit** menus. The choices are: **Kilogram** or **Pound**.

Transformation	Input Type	Output Type	Parameter (* = required)	Description
ConvertVolumeUnit	number	number	**fromUnit** * **toUnit** *	Convert a number that represents volume to another unit. For the **fromUnit** and **toUnit** parameters, select the appropriate unit from the **From Unit** and **To Unit** menus. The choices are: **Cubic Foot**, **Cubic Meter**, **Gallon US Fluid**, or **Liter**.
DayOfWeek	date	number	None	Return the day of the week (1 through 7) that corresponds to the date.
DayOfYear	date	number	None	Return the day of the year (1 through 366) that corresponds to the date.
EndsWith	string	Boolean	**string**	Return true if a string ends with the specified **string**, including case.
Equals	any	Boolean	**value**	Return true if a field is equal to the specified **value**, including case.
FileExtension	string	string	None	From a string that represents a file name, return the file extension without the dot.

Transformation	Input Type	Output Type	Parameter (* = required)	Description
Floor	number	number	None	Return the whole number floor of a number.
Format	any	string	**template** *	In **template**, replace each placeholder (such as **%s**) with the value of the input field and return a string that contains the result. This is similar to mechanisms that are available in programming languages such as Java and C.
IndexOf	string	number	**string**	In a string, starting at 0, return the first index of the specified **string**. Return **-1** if it is not found.
IsNull	any	Boolean	None	Return true if a field is null.
LastIndexOf	string	number	**string**	In a string, starting at 0, return the last index of the specified **string**. Return **-1** if it is not found.
Length	any	number	None	Return the length of the field, or **-1** if the field is null.
Lowercase	string	string	None	Convert a string to lowercase.

Transformation	Input Type	Output Type	Parameter (* = required)	Description
Normalize	string	string	None	Replace consecutive whitespace characters with a single space and trim leading and trailing whitespace from a string.
PadStringLeft	string	string	**padCharacter** * **padCount** *	Insert the character supplied in **padCharacter** at the beginning of a string. Do this the number of times specified in **padCount**.
PadStringRight	string	string	**padCharacter** * **padCount** *	Insert the character supplied in **padCharacter** at the end of a string. Do this the number of times specified in **padCount**.
Prepend	string	string	**string**	Prefix **string** to the beginning of a string. the default is to prepend nothing.
ReplaceAll	string	string	**match** * **newString**	In a string, replace all occurrences of the supplied matching string with the supplied **newString**. The default **newString** is an empty string.

Transformation	Input Type	Output Type	Parameter (* = required)	Description
ReplaceFirst	string	string	**match** * **newString** *	In a string, replace the first occurrence of the specified **match** string with the specified **newString**. The default **newString** is an empty string.
Round	number	number	None	Return the rounded whole number of a number.
SeparateByDash	string	string	None	Replace each occurrence of whitespace, colon (:), underscore (_), plus (+), and equals (=) with a hyphen (-).
SeparateByUnderscore	string	string	None	Replace each occurrence of whitespace, colon (:), hyphen (-), plus (+), and equals (=) with an underscore (_).
StartsWith	string	Boolean	**string**	Return true if a string starts with the specified string (including case).

Transformation	Input Type	Output Type	Parameter (* = required)	Description
Substring	string	string	**startIndex** * **endIndex**	Retrieve a segment of a string from the specified inclusive **startIndex** to the specified exclusive **endIndex**. Both indexes start at zero. **startIndex** is inclusive. **endIndex** is exclusive. The default value of **endIndex** is the length of the string.
SubstringAfter	string	string	**startIndex** * **endIndex** **match** *	Retrieve the segment of a string after the specified **match** string from the specified inclusive **startIndex** to the specified exclusive **endIndex**. Both indexes start at zero. The default value of **endIndex** is the length of the string after the supplied **match** string.

Transformation	Input Type	Output Type	Parameter (* = required)	Description
SubstringBefore	string	string	**startIndex** * **endIndex** **match** *	Retrieve a segment of a string before the supplied **match** string from the supplied inclusive **startIndex** to the supplied exclusive **endIndex**. Both indexes start at zero. The default value of **endIndex** is the length of the string before the supplied **match** string.
Trim	string	string	None	Trim leading and trailing whitespace from a string.
TrimLeft	string	string	None	Trim leading whitespace from a string.
TrimRight	string	string	None	Trim trailing whitespace from a string.
Uppercase	string	string	None	Convert a string to uppercase.

6.8. VIEWING THE MAPPINGS IN A STEP

While you are adding or editing a data mapper step, you can view the mappings already defined in this step. This lets you check whether the correct mappings are in place.

To view mappings when you are already in the data mapper, in the upper right, click ▦ .

To dismiss the list of mappings and redisplay the source and target fields, click ▦ again.

To view mappings when you are editing an integration but you are not adding or editing a data mapper step:

1. In the integration visualization panel on the left, click the data mapper step for which you want view the defined mappings.

2. In the data mapper, in the upper right, click ▦ .

To view mappings when you are not editing a data mapper step:

1. In the left panel, click **Integrations**.
2. In the entry for the integration whose data mappings you want to view, on the right, click the ⋮ .
3. In the popup menu, click **Edit**.
4. In the integration visualization panel on the left, click the data mapper step for which you want view the defined mappings.
5. In the data mapper, in the upper right, click ▦ .

CHAPTER 7. CUSTOMIZING FUSE ONLINE

Fuse Online provides many connectors that you can use to connect to common applications and services. There are also a number of built-in steps for processing data in common ways. However, if Fuse Online does not provide a feature that you need, talk with a developer about your requirements. An experienced developer can help you customize integrations by providing any of the following:

- An OpenAPI specification that Fuse Online can use to create a connector for a REST API client. You upload this specification to Fuse Online and Fuse Online creates a connector according to the specification. You then use the connector to create a connection that you can add to an integration. For example, many retail web sites provide a REST API client interface that a developer can capture in an OpenAPI specification.

- A **JAR** file that implements a Fuse Online extension. An extension can be any one of the following:

    - One or more steps that operate on integration data between connections

    - A connector for an application or service

    - A library resource such as a JDBC driver for a proprietary SQL database
      You upload this **JAR** file to Fuse Online and Fuse Online makes the custom feature provided by the extension available.

See the following topics for details:

- Section 7.1, “Developing REST API client connectors”

- Section 7.2, “Adding and managing REST API client connectors”

- Section 7.3, “Developing Fuse Online extensions”

- Section 7.4, “Adding and managing extensions”

7.1. DEVELOPING REST API CLIENT CONNECTORS

Fuse Online can create connectors for Representational State Transfer Application Programming Interfaces (REST APIs) that support Hypertext Transfer Protocol (HTTP). To do this, Fuse Online requires a valid OpenAPI 2.0 specification that describes a REST API that you want to connect to. If the API service provider does not make an OpenAPI specification available then an experienced developer must create the OpenAPI specification.

The following topics provide information and instructions for developing REST API connectors:

- Section 7.1.1, “Requirements for REST API client connectors”

- Section 7.1.2, “Guidelines for OpenAPI specifications”

- Section 7.1.3, “Providing client credentials in parameters”

- Section 7.1.4, “Automatically refreshing access tokens”

7.1.1. Requirements for REST API client connectors

After you upload an OpenAPI specification to Fuse Online, a connector to the REST API becomes available. You can select the connector to create a REST API client connection. You can then create a new integration and add the REST API client connection, or you can edit an existing integration to add the REST API client connection.

Fuse Online connectors support OAuth 2.0 and HTTP Basic Authorization. If access to the REST API requires Transport Layer Security (TLS) then the API needs to use a valid certificate that is issued by a recognized Certificate Authority (CA).

A REST API that uses OAuth must have an authorization URL that takes a client callback URL as input. After Fuse Online creates the connector and before you use the connector to create a connection, you must visit that URL to register your Fuse Online environment as a client of the REST API. This authorizes your Fuse Online environment to access the REST API. As part of registration, you provide the Fuse Online callback URL. The details for doing this are described in Connecting Fuse Online to Applications and Services, Register with REST APIs.

Fuse Online cannot create connectors for REST APIs that support the HTTP 2.0 protocol.

7.1.2. Guidelines for OpenAPI specifications

When Fuse Online creates a REST API client connector, it maps each resource operation in the OpenAPI specification to a connection action. The action name and action description come from documentation in the OpenAPI specification.

The more detail that the OpenAPI specification provides, the more support Fuse Online can offer when connecting to the API. For example, the API definition is not required to declare data types for requests and responses. Without type declarations, Fuse Online defines the corresponding connection action as typeless. However, in an integration, you cannot add a data mapping step immediately before or immediately after an API connection that performs a typeless action.

One remedy for this is to add more information to the OpenAPI specification. Identify the OpenAPI resource operations that will map to the actions you want the API connection to perform. In the OpenAPI specification, ensure that there is a JSON schema that specifies each operation’s request and response types.

After you upload the specification, Fuse Online gives you an opportunity to review and edit it in Apicurio Studio, which is a visual editor for designing APIs based on the OpenAPI specification. You can add more detail, save your updates, and Fuse Online creates an API client connector hat incorporates your updates.

If the OpenAPI specification for the API declares support for **application/json** content type and also **application/xml** content type then the connector uses the JSON format. If the OpenAPI specification specifies **consumes** or **produces** parameters that define both **application/json** and **application/xml**, then the connector uses the JSON format.

7.1.3. Providing client credentials in parameters

When Fuse Online tries to obtain authorization to access an OAuth2 application, it uses HTTP basic authentication to provide client credentials. If you need to, you can change this default behavior so that Fuse Online passes client credentials to the provider as parameters instead of using HTTP basic authentication. This affects the use of the **tokenUrl** endpoint that is used to obtain an OAuth access token.

IMPORTANT

This is a Technology Preview feature.

To specify that Fuse Online should pass client credentials as parameters, in the **securityDefinitions** section of the OpenAPI specification, add the **x-authorize-using-parameters** vendor extension with a setting of **true**. In the example below, the last line specifies **x-authorize-using-parameters**:

```
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
```

The setting of the **x-authorize-using-parameters** vendor extension is **true** or **false**:

- **true** indicates that client credentials are in parameters.
- **false**, the default, indicates that Fuse Online uses HTTP basic authentication to provide client credentials.

7.1.4. Automatically refreshing access tokens

If an access token has an expiration date, then Fuse Online integrations that use that token to connect to an application stop running successfully when the taken expires. To obtain a new access token, you must either reconnect to the application or re-register with the application.

If you need to, you can change this default behavior so that Fuse Online automatically requests a new access token in the following situations:

- When the expiration date has been reached.
- When HTTP response status codes that you specify are received.

IMPORTANT

This is a Technology Preview feature.

To specify that Fuse Online should automatically try to obtain a new access token in the situations described, in the **securityDefinitions** section of the OpenAPI specification, add the **x-refresh-token-retry-statuses** vendor extension. The setting of this extension is a comma separated list that specifies HTTP response status codes. When an access token's expiration date is reached or when Fuse Online receives a message from an OAuth2 provider and the message has one of these response status codes, then Fuse Online automatically tries to obtain a new access token. In the example below, the last line specifies **x-refresh-token-retry-statuses**:

```
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
```

NOTE

Sometimes, an API operation fails and a side effect of that failure is that the access token is refreshed. In this situation, even when obtaining a new access token is successful, the API operation still fails. In other words, Fuse Online does not retry the failed API operation after it receives the new access token.

7.2. ADDING AND MANAGING REST API CLIENT CONNECTORS

Fuse Online can create a REST API client connector from an OpenAPI specification. For information about the content of the OpenAPI specification, see Section 7.1, "Developing REST API client connectors".

The following topics provide information and instructions for adding and managing REST API client connectors:

- Section 7.2.1, "Creating REST API client connectors"
- Section 7.2.2, "Updating API client connectors"
- Section 7.2.3, "Deleting API client connectors"

After you create a REST API client connector, for details about using that connector, see Connectng Fuse Online to Applications and Services, Connecting to REST APIs.

7.2.1. Creating REST API client connectors

To create an API client connector:

1. In the Fuse Online navigation panel, click **Customizations** to display the **API Client Connectors** tab. Any API client connectors that are already available are listed here.

2. Click **Create API Connector**.

3. On the **Create API Client Connector** page, do one of the following:

    - Click **Browse** and select the OpenAPI file that you want to upload.

    - Select **Use a URL** and paste the URL for your OpenAPI specification in the input field.

4. Click **Next**. If there is invalid or missing content, Fuse Online displays information about what needs to be corrected. Select a different OpenAPI file to upload or click **Cancel**, revise the OpenAPI file, and upload the updated file.
   If the specification is valid, Fuse Online displays a summary of the actions that the connector provides. This might include errors and warnings about the action specifications.

5. If you are satisfied with the action summary, click **Next**.
   Or, to revise the OpenAPI specification, in the lower right, click **Review/Edit** to open Apicurio Studio. Update the specification as needed. In the upper right, click **Save** to incorporate your updates into the new API client connector. Then click **Next** to continue creating the API client connector.

   For details about using Apicurio Studio, see https://www.apicur.io/. Sometimes, if you provide a URL for the OpenAPI specification, Fuse Online can upload it but cannot open it for editing. Typically, this is caused by settings on the file's host. To open the specification for editing, Fuse

Online requires that the file host has:

- An **https** URL. An **http** URL does not work.
- Enabled CORS.

6. Indicate the API's security requirements by selecting one of the following:
    a. **No Security**
    b. **HTTP Basic Authorization** – Enter the user name and password you want to use to access the API.
    c. **OAuth** – Fuse Online prompts you to enter:
        i. **Authorization URL** is the location for registering Fuse Online as a client of the API. Registration authorizes Fuse Online to access the API. See Connecting Fuse Online to Applications and Services, Registering Fuse Online as a REST API client. The OpenAPI specification or other documentation for the API should specify this URL. If it does not then you must contact the service provider to obtain this URL.
        ii. **Access Token URL** is required for OAuth authorization. Again, the OpenAPI specification or other documentation for the API should provide this URL. If it does not then you must contact the service provider.
7. Click **Next**. Fuse Online displays the connector's name, description, host, and base URL as indicated by the OpenAPI specification. For connections that you create from this connector,
    - Fuse Online concatenates the host and base URL values to define the endpoint for the connection. For example, if the host is **https://example.com** and the base URL is **/api/v1** then the connection endpoint is **https://example.com/api/v1**.
    - Fuse Online applies the schema specified in the OpenAPI specification to data mapping steps. If the OpenAPI specification supports more than one schema then Fuse Online uses the TLS (HTTPS) schema.
8. Review the connector details and optionally upload an icon for the connector. If you do not upload an icon, Fuse Online generates one. You can upload an icon at a later time. When Fuse Online displays the flow of an integration, it displays a connector's icon to represent connections that are created from that connector.
   To override a value obtained from the OpenAPI file, edit the field value that you want to change. After Fuse Online creates a connector, you cannot change it. To effect a change, you need to upload an updated OpenAPI specification so that Fuse Online can create a new connector or you can upload the same specification and then edit it in Apicurio Studio. You then continue the process for creating a new API client connector.
9. When you are satisfied with the connector details, click **Create API Connector**. Fuse Online displays the new connector with the other connectors.

For details about using your new API connector, see Connecting Fuse Online to Applications and Services, Connecting to REST APIs.

7.2.2. Updating API client connectors

You cannot update an API client connector. If there is an update to the API's OpenAPI specification, then you must do one of the following:

- Upload the updated OpenAPI specification and create a new API client connector.
- Upload the out-of-date specification again, update it in Apicurio Studio, and create a new API client connector.

To update integrations to use connections that are based on the updated OpenAPI specification:

1. Create a new API client connector based on the updated OpenAPI specification. To easily distinguish between the old connector and the new connector, you might want to specify a version number in the connector name or the connector description.
2. Create a new connection from the new connector. Again, you want to be able to easily distinguish between connections created from the old connector and connections created from the new connector. A version number in the connection name or connection description is helpful.
3. Edit each integration that uses a connection that was created from the old connector by removing the old connection and adding the new connection.
4. Publish each updated integration.
5. Recommended, but not required: delete the old connector and the old connections.

7.2.3. Deleting API client connectors

You cannot delete a connector when there is a connection that was created from that connector and this connection is being used in an integration. After you delete an API client connector, you cannot use a connection that was created from that connector.

To delete an API client connector:

1. In the left panel, click **Customizations**.
2. In the **API Client Connectors** tab, to the right of the name of the connector that you want to delete, click **Delete**.
3. Read the confirmation popup to be sure that you want to click **Delete**.

7.3. DEVELOPING FUSE ONLINE EXTENSIONS

If Fuse Online does not provide a feature that is needed to create an integration, then an expert developer can code an extension that provides the needed behavior. The Syndesis extensions repository contains examples of extensions.

A business integrator shares requirements with a developer who codes the extension. The developer provides a **.jar** file that contains the extension. The business integrator uploads the **.jar** file in Fuse Online to make the custom connector, custom step(s), or library resource available for use in Fuse Online.

The Fuse Tooling plugin to Red Hat Developer Studio provides a wizard that helps you develop a step extension or a connector extension. It is a matter of personal preference whether you choose to develop a step extension or a connector extension in Developer Studio or in some other IDE. For information about using the Developer Studio plugin, see Developing extensions for Fuse Online integrations .

In this document, the following topics outline the procedure, describe the requirements, and provide additional examples for developing extensions in an IDE that you choose.

7.3.1. General procedure for developing extensions

Before you start to develop an extension, become familiar with the tasks that you will need to accomplish.

Prerequisites

- Familiarity with Maven
- Familiarity with Camel if you are developing an extension that provides a connector or that provides an integration step that operates on data between connections
- Experience programming

Procedure

1. Obtain an understanding of what the extended feature must do. Talk to your business colleague to understand the feature requirements.
2. Determine whether you need to develop a step extension, a connector extension, or a library extension.
3. Set up the Maven project in which to develop the extension.
4. If you are developing a step extension:
   a. Decide whether to implement it as a Camel route or implement it by using the Syndesis **Step** API. Information for the Syndesis API is at http://javadoc.io/doc/io.syndesis.extension/extension-api.
   b. If you choose to implement the extension as a Camel route, decide whether to implement XML fragments, a **RouteBuilder** class, or a bean.
   c. In your Maven project, specify the required metadata, such as the **schemaVersion**, extension **name**, **extensionId**, and so on.
5. Code the classes that implement the feature.
6. Add dependencies to the project’s **pom.xml** file.

7. For connector and library extensions, and for step extensions that you implement in XML, create the JSON file that defines the extension.
   For step extensions that you implement in Java, Maven can generate the JSON extension definition file for you when you specify corresponding data structure values in your Maven project.

8. Run Maven to build the extension and create the extension's JAR file.

9. Test the extension by uploading the JAR file to a Fuse Online development environment.

10. Provide the JAR file that packages the extension to your business colleague, who uploads it to a Fuse Online production environment. When you provide the JAR file, let your business colleague know about any configuration settings that require information beyond what appears in the Fuse Online web interface.

7.3.2. Description of the kinds of extensions

An extension defines one of the following:

- One or more custom steps that operate on integration data between connections. Each custom step performs one action. This is a step extension.

- A library resource that an integration runtime uses. For example, a library extension can provide a JDBC driver for connecting to a proprietary SQL database, such as Oracle.

- A single custom connector for creating connections to a particular application or service that you want to integrate. This is a connector extension.

> NOTE
>
> Fuse Online can use an OpenAPI specification to create a connector for a REST API client. See Section 7.1, "Developing REST API client connectors".

A business integrator shares requirements with a developer who codes the extension. The developer provides a **.jar** file that contains the extension. The business integrator uploads the **.jar** file in Fuse Online to make the custom connector, custom step(s), or library resource available for use within Fuse Online.

An extension **.jar** file that you upload to Fuse Online always contains exactly one extension.

For an example of uploading and using an extension that provides a step that operates on data between connections, see the AMQ to REST API sample integration tutorial .

7.3.3. Overview of extension content and structure

An extension is a collection of classes, dependencies, and resources that are packaged in a **.jar** file.

Fuse Online uses Spring Boot to load an extension. Consequently, you must package an extension according to Spring Boot's executable JAR format. For example, ensure that you use the **ZipEntry.STORED()** method to save a nested JAR file.

The structure of a **.jar** file that packages an extension is as follows:

```
extension.jar
|
+- META-INF
| |
| +- syndesis
|     |
|     +- syndesis-extension-definition.json ❶
|
+- mycompany
| |
| +-project
|    |
|    +-YourClasses.class ❷
|
+- lib ❸
  |
  +-dependency1.jar
  |
  +-dependency2.jar
```

❶ A JSON schema file that specifies the data structures that define the extension. This is referred to as the extension definition JSON file.

❷ The Java classes that implement the behavior that the extension provides.

❸ Additional dependencies that are required to build and execute the custom feature.

7.3.4. Requirements in an extension definition JSON file

Each extension must have a **.json** file that defines the extension by specifying values for data structures such as name, description, supported actions, and dependencies. For each extension type, the following table indicates whether Maven can generate the extension definition JSON file and which data structures are required.

Extension Type	Maven Can Generate Extension Definition	Required Data Structures
Step extension in Java	Yes	**schemaVersion** **name** **description** **version** **extensionId** **extensionType** **actions** **dependencies** *

Extension Type	Maven Can Generate Extension Definition	Required Data Structures
Step extension in XML	No	**schemaVersion** **name** **description** **version** **extensionId** **extensionType** **actions** **dependencies** *
Connector extension	No	**schemaVersion** **name** **description** **version** **extensionId** **extensionTypeproperties** **actions** **dependencies** * **componentScheme** **connectorCustomizers** **connectorFactory**
Library extension	No	**schemaVersion** **name** **description** **version** **extensionId** **extensionType** **dependencies** *

*While specification of **dependencies** is not strictly required, in practice, there are almost always dependencies that you need to specify.

Typically, an extension definition file has the following layout:

```
{
  "schemaVersion": "v1",
  "name": "",
  "description": "",
  "version": "",
  "extensionId": "",
  "extensionType": "",
  "properties": {
  },
  "actions": [
  ],
  "dependencies": [
  ],
}
```

- **schemaVersion** defines the version of the extension definition schema. Internally, Syndesis uses **schemaVersion** to determine how to map the extension definition to the internal model. This allows extensions that were developed against an old version of Syndesis to be deployed on newer versions of Syndesis.

- **name** is the name of the extension. When you upload an extension to Fuse Online, this name appears.

- **description** is any useful information that you want to specify. Fuse Online does not operate on this value.

- **version** is for your convenience to help you distinguish updates to an extension. Fuse Online does not operate on this value.

- **extensionId** defines a unique ID for the extension. This should be unique at least across a Syndesis environment.

- **extensionType** indicates to Syndesis what the extension provides. As of Syndesis version 1.3, the following extension types are supported:

  - **Steps**
  - **Connectors**
  - **Libraries**

- **properties** defines the global configuration options that are supported by the extension. Only connector extensions use properties that you specify. For example:

```
"propertyName": {
  "deprecated": true|false,
  "description": "",
  "displayName": "",
  "group": "",
  "kind": "",
  "label": "",
  "required": true|false,
  "secret": true|false,
  "javaType": "",
  "type": "",
  "defaultValue": "",
  "enum": {
  }
}
```

- **actions** defines the operations that a connector can perform or the operation that a step between connections can perform. Only connector and step extensions use actions that you specify. The format for an action specification looks like this:

```
{
    "id": "",
    "name": "",
    "description": "",
    "actionType": "step|connector",
      "descriptor": {
      }
  }
```

  - **id** is a unique ID for the action. This should be unique at least within a Syndesis environment.
  - **name** is the action name that appears in Fuse Online. An integrator sees this value as the name of a connection action or as the name of a step that operates on integration data between connections.
  - **description** is the action description that appears in Fuse Online. Use this field to help the integrator understand what the action does.
  - **actionType** indicates whether the action is performed by a connection or a step that is between connections.
  - **descriptor** specifies nested attributes such as **kind**, **entrypoint**, **inputDataType**, **outputDatatype** and more.

- **dependencies** defines the resources that this extension requires Fuse Online to provide. Define a dependency as follows:

```
{
  "type": "MAVEN",
  "id"   : "org.apache.camel:camel-telegram:jar:2.21.0"
}
```

  - **type** indicates the type of the dependency. Specify **MAVEN**. (It is expected that other types will be supported in the future.)
  - **id** is the ID of the Maven dependency, which is a Maven GAV.

7.3.5. Description of Maven plugin that supports extensions

The **extension-maven-plugin** supports extension development by packaging the extension as a valid Spring Boot module. For step extensions that you implement in Java, this plugin can generate the extension definition JSON file.

In your Maven project's`pom.xml` file, add the following plugin declaration:

```
<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>
```

The **extension-maven-plugin** defines the following goals:

- **generate-metadata** generates the JSON extension definition file that will be in the generated JAR file as follows:

    a. Maven starts with the data structure specifications that are in the **META-INF/syndesis/syndesis-extension-definition.json** file, if there is one.
    If you are coding in XML, then you must define the extension definition JSON file yourself and it must specify all required data structures.

    If you are developing a connector or library extension, then you must define the extension definition JSON file yourself and it must specify all required data structures.

    If you are developing a step extension in Java, you can:

    - Create the extension definition JSON file yourself.
    - In your Java code, specify annotations that define all required data structures. You do not create an extension definition JSON file.
    - Create an extension definition JSON file and specify some but not all data structures.

    b. For step extensions that you develop in Java, Maven obtains missing specifications from code annotations

    c. Maven adds the dependencies list, which specifies dependencies that are provided with a scope of **provided** and that are managed through the **extension-bom**.

- **repackage-extension** packages the extension.

    - Dependencies and related transitive dependencies that are not managed through the **extension-bom** are in the **lib** folder of the generated JAR.
    - For library extensions, dependencies whose scope is **system** are in the **lib** folder of the generated JAR.

For example, suppose your Maven project has the following **pom.xml** file:

```
<?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>
```

Based on this **pom.xml** file, the generated extension definition JSON file looks like this:

```
{
  "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"
}
```

The generated archive has this structure and content:

```
my-extension-1.0.0.jar
|
+- lib
|  |
|  + jfiglet-0.0.8.jar
|
+- META-INF
  |
  +- MANIFEST.MF
     |
     +- syndesis
        |
        +- syndesis-extension-definition.json
```

7.3.6. Alternatives for developing step extensions

A step extension implements one or more custom steps. Each custom step implements one action for processing integration data between connections. The following examples demonstrate the alternatives for developing step extensions:

- Section 7.3.6.1, “Example of developing a Camel route with XML fragments”
- Section 7.3.6.2, “Example of developing a Camel route with RouteBuilder”
- Section 7.3.6.3, “Example of using a Camel bean”
- Section 7.3.6.4, “Example of using the Syndesis Step API”

Syndesis provides custom Java annotations that you can use in conjunction with the **syndesis-extension-plugin**. When you implement a step extension or a connector extension in Java, you can specify annotations that enable Maven to add action definitions to the extension definition JSON file. To enable annotation processing, add the following dependency to your Maven project:

```
<dependency>
  <groupId>io.syndesis.extension</groupId>
  <artifactId>extension-annotation-processor</artifactId>
  <optional>true</optional>
</dependency>
```

7.3.6.1. Example of developing a Camel route with XML fragments

To develop a custom step, you can implement the action as an XML fragment that is a Camel route that has an input such as **direct**. The Syndesis runtime invokes this route in the same way that it invokes any other Camel route.

For example, suppose that you want to create a step that logs the body of a message with an optional prefix. The following XML defines a Camel route that does this.

```
<?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>
```

When you develop an extension in XML, you must create the extension definition JSON file yourself. For this XML fragment, the **src/main/resources/META-INF/syndesis/syndesis-extension-definition.json** file could define the action as follows:

```
{
  "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. The type of the action is set to **ENDPOINT**. The runtime invokes a Camel endpoint to execute the action provided by this custom step.
2. The Camel endpoint to invoke is **direct:log**. This is the **from** specification in the route.
3. This is the location of the XML fragment.
4. These are the properties that the action defined in this custom step exposes to the integrator who will be adding this step to an integration. In Fuse Online, each value that the integrator specifies in the user interface gets mapped to a message header that has the same name as the property. In this example, the integrator will see one input field, with the **Log Prefix** display name.

> **WARNING**
>
> Syndesis does not support full Camel XML configuration. Syndesis supports only the <routes> tag.

7.3.6.2. Example of developing a Camel route with RouteBuilder

You can implement a custom step by developing an action as a Camel route with the support of the **RouteBuilder** class. Such a route has an input such as **direct**. Syndesis invokes this route in the same way that it invokes any other Camel route.

To implement the example that creates a step that logs the body of a message with an optional prefix, you can write something like this:

```
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 The **@Action** annotation indicates the action definition.

2 The **@ConfigurationProperty** annotation indicates the property definition.

3 This is the action implementation.

This Java code uses Syndesis annotations, which means that the **extension-maven-plugin** can automatically generate the action definition. In the extension definition JSON file, the action definition will look like this:

```
{
  "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. The type of action is **ENDPOINT**. The runtime invokes a Camel endpoint to execute the action that this step implements.

2. This is the Camel endpoint to invoke. It is the **from** specification in the route.

3. This is the class that implements **RoutesBuilder**.

4. These are the properties that the action defined in this custom step exposes to the integrator who will be adding this step to an integration. In Fuse Online, each value that the integrator specifies in the user interface gets mapped to a message header that has the same name as the property. In this example, the integrator will see one input field, with the **Log Prefix** display name.

7.3.6.3. Example of using a Camel bean

You can implement a custom step by developing an action as a Camel bean processor. To implement the example that creates a step that logs the body of a message with an optional prefix, you can write something like this:

```
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 This is the function that implements the action.

This Java code uses Syndesis annotations, which means that the **extension-maven-plugin** can automatically generate the action definition. In the extension definition JSON file, the action definition will look like this:

```
{
  "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 The type of the action is **BEAN**. The runtime invokes a Camel bean processor to execute the action in this custom step.

② This is the Camel bean to invoke.

③ These are the properties that the action defined in this custom step exposes to the integrator who will be adding this step to an integration. In Fuse Online, each value that the integrator specifies in the user interface gets mapped to a message header that has the same name as the property. In this example, the integrator will see one input field, with the **Log Prefix** display name.

When you use beans, you might find it convenient to inject user properties into the bean instead of retrieving them from the exchange header. To do this, implement getter and setter methods for the properties that you want to get injected. The action implementation would look like this:

```
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 This is the property setter method.

2 This is the property getter method.

7.3.6.4. Example of using the Syndesis Step API

You can implement a custom step by using the Syndesis **Step** API. This provides a way to interact with runtime route creation. You can use any method provided by the **ProcessorDefinition** class and you can create more complex routes. Information for the Syndesis API is at http://javadoc.io/doc/io.syndesis.extension/extension-api.

Here is an example of a step extension that uses the Syndesis **Step** API to implement a split action:

```
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 This is the implementation of the action that the custom step performs.

This Java code uses Syndesis annotations, which means that the **extension-maven-plugin** can automatically generate the action definition. In the extension definition JSON file, the action definition will look like this:

```
{
  "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. The type of the action is **STEP**.
2. This is the class that is implementing the **Step** interface.

7.3.7. Example of developing a connector extension

If Fuse Online does not provide a connector for the application or service that you want to connect to in an integration, an experienced developer can code an extension that contributes a new connector to Fuse Online.

IMPORTANT

For connector extensions, it is not yet possible to automatically generate the extension definition JSON file from Java code.

A connector is essentially a proxy for a Camel component. A connector configures the underlying component and creates endpoints according to options that are defined in the extension definition and in user-supplied options that the Fuse Online web interface collects.

The connector extension definition extends the extension definition that is required for step extensions with the following additional data structures:

- **componentScheme**
  Defines the Camel component that the connector uses. For example, **telegram**.

- **connectorCustomizers**
  Specifies a list of classes that implement the ComponentProxyCustomizer class. Each class customizes the behavior of a connector. For example, a class might manipulate properties before they are applied to the underlying component/endpoint, or a class might add pre/post endpoint logic. For each class, specify the full class name of the implementation, for example, **com.mycomponent.MyCustomizer**.

- **connectorFactory**
  Defines the class that implements the ComponentProxyFactory class, which creates and/or configures the underlying component/endpoint. Specify the full class name of the implementation.

You can set the above data structures at the action level or at the global level. Settings at the action level have precedence over the same items defined at global level.

Customizer example

The following customizer example sets up a **DataSource** from individual options:

```
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");
            }
        }
    }
}
```

①② By implementing **CamelContextAware**, Syndesis injects the Camel context and then invoking the customize method.

③④⑤ Processes options and then removes them from the options map.

Example of injecting properties

If the customizer respects Java bean conventions, you can also inject the properties, as shown in this revision of the previous example:

```
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 By implementing **CamelContextAware**, Syndesis injects the Camel context and then invokes the customize method. This sample code overrides the **setCamelContext()** and **getCamelContext()** methods, and sets the user name.

4 5 6 7 8 Starting here, the sample code processes the injected options and automatically removes them from the options map.

Using a customizer to configure before/after logic

You can use a customizer to configure before/after logic as shown in this example:

```
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);
       }
    }
}
```

Customizing behavior of **ComponentProxyComponent**

To customize the behavior of the ComponentProxyComponent class, you can override any of the following methods:

- **createDelegateComponent()**
  Syndesis invokes this method when the proxy starts and it is used to eventually create a dedicated instance of the component with the scheme defined by the **componentScheme** option.

  The default behavior of this method is to determine if any of the connector/action options applies at the component level. If the same option cannot be applied at the endpoint, and only in this case, the method creates a custom component instance and configures it according to the applicable options.

- **configureDelegateComponent()`**
  Syndesis invokes this method only if a custom component instance has been created to configure additional behavior of the delegated component instance.

- **createDelegateEndpoint()**
  Syndesis invokes this method when the proxy creates the endpoint and by default creates the endpoint by using Camel catalog facilities.

- **configureDelegateEndpoint()**
  After the delegated endpoint has been created, Syndesis invokes this method to configure additional behavior of the delegated endpoint instance, for example:

```
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(","))
                    );
                }
            }
        };
    }
}
```

7.3.8. How to develop library extensions

A library extension provides a resource that an integration requires at runtime. A library extension does not contribute steps or connectors to Fuse Online.

Support for library extensions is evolving. In this release, in the Fuse Online web interface:

- When you create an integration, you cannot select which library extension(s) an integration should include.

- When you add a database connection to an integration, Fuse Online adds all extensions that have the **jdbc-driver** tag to the integration runtime.

See the following topics for information about creating library extensions:

- Section 7.3.8.1, “Description of library extension definition and structure”
- Section 7.3.8.2, “Creating JDBC driver library extensions”

7.3.8.1. Description of library extension definition and structure

A library extension does not define any actions. Here is a sample definition for a library extension:

```
{
  "schemaVersion" : "v1",
  "name" : "Example JDBC Driver Library",
  "description" : "Syndesis Extension for adding a custom JDBC Driver",
  "extensionId" : "io.syndesis.extensions:syndesis-library-jdbc-driver",
  "version" : "1.0.0",
  "tags" : [ "jdbc-driver" ],
  "extensionType" : "Libraries"
}
```

Other than the lack of actions, the structure of a library extension is the same as the structure of a step or connector extension. See Section 7.3.3, “Overview of extension content and structure” and Section 7.3.4, “Requirements in an extension definition JSON file” .

In a Maven project that creates a library extension, to add dependencies that are not available from a Maven repository, specify a **system** dependency, for example:

```
<dependency>
    <groupId>com.company</groupId>
    <artifactId>jdbc-driver</artifactId>
    <version>1.0</version>
    <scope>system</scope>
    <systemPath>${project.basedir}/lib/jdbc-driver-1.0.jar</systemPath>
</dependency>
```

7.3.8.2. Creating JDBC driver library extensions

To connect to a SQL database other than Apache Derby, MySQL, and PostgreSQL, you can create a library extension that wraps a JDBC driver for the database you want to connect to. After uploading this extension to Fuse Online, the Fuse Online-provided **Database** connector can access the driver to validate and create connections to the proprietary database. You do not create a new connector for your particular database.

The Syndesis open source community provides a project for creating an extension that wraps a JDBC driver.

Package one driver only in an extension. This makes it easier to manage the extension as part of managing your particular database. However, it is possible to create a library extension that wraps more than one driver.

Prerequisites

To use the Syndesis project, you must have a GitHub account.

Procedure

1. Ensure access to the JDBC driver for the database you want to connect to by doing one of the following:
    a. Confirm that the driver is in a Maven repository.

    b. Download the driver.

2. In a browser tab, go to https://github.com/syndesisio/syndesis-extensions

3. Fork the **syndesis-extensions** repository to your GitHub account.

4. Create a local clone from your fork.

5. In your **syndesis-extensions** clone:
    a. If the driver is not in a Maven repository, copy the driver into the **syndesis-library-jdbc-driver/lib** folder.

    b. Edit the **syndesis-library-jdbc-driver/pom.xml** file:
        i. Update the value of the **Name** element to be a name that you choose for this extension.

        ii. Update the value of the **Description** element to provide helpful information about this extension.

        iii. If the driver is in a Maven repository, ensure that a reference to that Maven repository is in the **pom.xml** file.

        iv. Examine the rest of the content of the **pom.xml** file and change any relevant metadata as needed.

    c. In the **syndesis-library-jdbc-driver** folder, execute **mvn clean package** to build the extension.

The generated **.jar** file is in the **syndesis-library-jdbc-driver/target** folder. Provide this **.jar** file for uploading to Fuse Online.

NOTE

Fuse Online does not yet offer a way to select which library extension(s) an integration should include. When you add a database connection to an integration, Fuse Online adds all extensions that have the **jdbc-driver** tag to integration runtime. This is expected to improve in a future release.

7.4. ADDING AND MANAGING EXTENSIONS

Extensions let you add customizations to Fuse Online so you can integrate applications the way you want to.

The following topics provide details:

- Section 7.4.1, “Making custom features available”
- Section 7.4.2, “Managing extensions”

7.4.1. Making custom features available

To make a custom feature available for use in an integration, you upload the extension to Fuse Online as follows:

1. In the left Fuse Online panel, click **Customizations**.
2. At the top, click **Extensions**.
3. Click **Import Extension**.
4. Drag and drop, or choose, the **.jar** file that contains the extension that you want to upload. A developer needs to make this file available to you. Fuse Online immediately tries to validate that the file contains an extension. If there is a problem, Fuse Online displays a message about the error. You must coordinate with the extension developer to obtain an updated **.jar** file, which you can then try to upload.
5. Review the extension details. After Fuse Online validates the file, it extracts and displays the extension's name, ID, description, and type. The type indicates whether the extension defines a custom connector, or one or more custom steps for operating on data between connections, or a JDBC driver for a proprietary database. An extension that provides a JDBC driver is referred to as a library extension.

   For a connector extension, Fuse Online displays the actions that are available to a connection that is created from this custom connector. In the extension, the developer might have provided an icon that Fuse Online can use to represent the application connections created from this connector. While you do not see this icon in the extension details page, it appears when you create connections from the custom connector. If the extension developer did not provide an icon in the extension, then Fuse Online generates an icon.

   For a step extension, Fuse Online displays the name of each custom step that the extension defines.

   For a library extension, if this extension contains a newer version of a JDBC driver that you previously uploaded, then you must remove the older version from your classpath. Ensure that your classpath has only one version of this driver and that the reference is to the newer driver you are uploading. Integrations that are running and that use connections based on the older driver are not affected. New connections that you create will use the new driver. If you start an integration that has a connection that was created with the older driver, Fuse Online automatically uses the new driver instead.
6. Click **Import Extension**. Fuse Online makes the custom connector or custom step(s) available and displays the extension's details page.

See also:

- Section 4.5, "Creating connections from custom connectors"
- Section 5.3.5, "Adding a custom step"
- Connecting to proprietary databases

7.4.2. Managing extensions

After you start using customizations provided in extensions, you can identify the integrations that use those customizations. This is helpful to do before you update or delete an extension.

For details, see:

- Section 7.4.2.1, “Identifying integrations that use extensions”
- Section 7.4.2.2, “Updating extensions”
- Section 7.4.2.3, “Deleting extensions”

7.4.2.1. Identifying integrations that use extensions

Before you update or delete an extension, you should identify the integrations that use customizations provided by that extension. To do this:

1. In the left Fuse Online panel, click **Customizations**.
2. At the top, click **Extensions**.
3. In the list of extensions, find the entry for the extension that you want to update or delete and click it.

Fuse Online displays details about the extension including a list of any integrations that use a customization provided by the extension.

7.4.2.2. Updating extensions

To update an extension:

1. Obtain an updated **.jar** file for the extension from the developer.
2. In Fuse Online, in the left panel, click **Customizations**.
3. Click the **Extensions** tab.
4. At the right of the entry for the extension that you want to update, click **Update**.
5. Click **Browse**, select the updated **.jar** file, and click **Open**.
6. Confirm that the extension details are correct and click **Update**.
7. In the details page for the updated extension, determine which integrations use the connector or custom step(s) defined in the extension.

It is up to you to know exactly what is required to update each integration that uses a custom connector or a custom step from the updated extension. At the very least, you must stop and restart (click **Start**) each integration that uses a customization defined in the updated extension. See Section 8.3.2, “Stopping integrations”.

In some cases, you might need to edit the integration to change or add configuration details for a customization. You must communicate with the extension developer to understand how to update integrations.

7.4.2.3. Deleting extensions

You can delete an extension even if a running integration uses a step that is provided by that extension or uses a connection that was created from a connector that was provided by that extension. After you delete an extension, you cannot start an integration that uses a customization that was provided by that extension.

To delete an extension:

1. In the left Fuse Online panel, click **Customizations**.
2. At the top, click **Extensions**.
3. In the list of extensions, find the entry for the extension that you want to delete and click **Delete**, which appears at the right of the entry.

There might be stopped or draft integrations that use a customization provided by an extension that you delete. To run one of these integrations, you will need to edit the integration to remove the customization. See Section 7.4.2.1, “Identifying integrations that use extensions” and Section 8.6, “Updating integrations”.

CHAPTER 8. MANAGING INTEGRATIONS

The following topics provide information to help you manage your integrations:

8.1. OVERVIEW OF MANAGING INTEGRATIONS

Each Fuse Online environment is hosted on OpenShift Online or OpenShift Container Platform (OCP). A common set up is to have a Fuse Online development environment, a Fuse Online test environment, and a Fuse Online deployment environment.

To facilitate this, Fuse Online provides the ability to export an integration from one Fuse Online environment and then import that integration into another Fuse Online environment.

The information and procedures for managing integrations are the same in each kind of Fuse Online environment, unless specifically noted.

8.2. ABOUT INTEGRATION LIFECYCLE HANDLING

After you create and publish an integration, you might want to update what the integration does. You can edit a draft of the published integration and then replace the running version with the updated version. To facilitate this, for each integration, Fuse Online maintains multiple versions as well as each version’s state. The following topics provide an understanding of the behavior to help you manage your integrations.

8.2.1. Description of integration versions

In each Fuse Online environment, each integration can have multiple versions. Support for multiple integration versions has several benefits:

- If you publish a version that does not work correctly, then you can return to running a correct version of the integration. To do that, you stop the incorrect version and start a version that runs the way you want it to.
- As requirements or tools change, you can incrementally update an integration. You do not need to create a new integration.

Fuse Online assigns a new version number each time it starts running a new version of an integration. For example, suppose you publish the Twitter to Salesforce sample integration. After it has been running, you update the integration to use a different account to connect to Twitter. You then publish the updated integration. Fuse Online stops the running version of integration, and publishes the updated version of the integration with an incremented version number.

The initial integration that was running is version 1. The updated integration that is now running is version 2. If you edit version 2, for example to use a different account to connect to Salesforce, and you publish that version then it becomes version 3 of the integration.

See also: Section 8.3.1, “Publishing integrations” and Section 8.3.4, “Restarting older integration versions”.

8.2.2. Description of integration states

Fuse Online can maintain multiple versions of the same integration. The benefit of this is that you can edit an integration to change its behavior. You do not need to create a new integration when something changes.

There can be exactly one draft version of an integration. Fuse Online has a definition for the draft version of an integration but it has never run this version of the integration. When you edit an integration, you are updating the draft version of the integration.

In Fuse Online, you can see a list of the versions of an integration in the integration’s summary page. To view this page, in the left navigation panel, click **Integrations** and then click the entry for the integration whose versions you want to see. In the list of versions, each entry indicates the state of that version, which is one of the following:

State	Description
Running	A **Running** version is executing; it is in service. Only one version of an integration can be running. That is, only one version at a time can be in the **Running** state.
Stopped	A **Stopped** version is not running. The draft version of an integration is in the **Stopped** state. Each integration that was running at one time and then stopped is in the **Stopped** state. If no version of this integration is in the **Running** state, then you can start a version that is stopped.
Pending	A **pending** version is in transition. Fuse Online is in the process of either starting this version of the integration or stopping this version of the integration, but the integration is not yet running or stopped.

Error	An integration version that is in the **Error** state encountered an OpenShift error while being started or while running. The error suspended start-up or execution. If this happens, try starting an earlier integration version that ran correctly. Alternatively, contact technical support for help. To do that, in any Fuse Online page, in the upper right, click the ⓘ icon and select **Support**.

8.3. PUTTING INTEGRATIONS INTO AND OUT OF SERVICE

After you create an integration, you can save it as a draft or start running it. The first time that you run an integration, Fuse Online assembles the needed resources, builds the integration runtime, deploys the OpenShift pod that will run the integration, and then starts executing the integration.

At any time, you can click a button to stop running the integration. When you want to start the integration again, Fuse Online already has what it needs so starting it takes less time than when you ran it for the first time.

The process of starting an integration for the first time is referred to as publishing the integration. The following topics provide details:

- Section 8.3.1, “Publishing integrations”
- Section 8.3.2, “Stopping integrations”
- Section 8.3.3, “Starting integrations”
- Section 8.3.4, “Restarting older integration versions”
- Section 8.3.5, “Troubleshooting integration execution”
- Section 8.3.6, “Copying integrations to other environments”

8.3.1. Publishing integrations

The first time that you run an integration you publish it. Publishing an integration builds and deploys the integration runtime. The integration starts running. After publishing an integration, you can stop it and restart it. Exactly one version of an integration can be running at one time.

To run an integration for the first time, publish it by doing one of the following:

- At the end of the procedure in which you create or edit the integration, in the upper right, click **Publish**.
- Publish the draft version of an integration:
  1. In the left Fuse Online panel, click **Integrations**.
  2. In the list of integrations, click the entry for the integration that you want to publish.
  3. On the integration’s summary page, in the **Details** tab, to the right of the draft entry, click ⋮ and select **Publish**.

Fuse Online displays the progress of the publication process, which has several stages:

1. **Assembling** creates pod resources needed to build the integration.

2. **Building** gets the integration ready to deploy.

3. **Deploying** waits for deployment of the pod that will run the integration.

4. **Starting** waits for the pod to start running the integration.

5. **Deployed** indicates that the integration is running.

During start-up, you can click **View Logs** to display OpenShift logs that provide start-up information.

When publishing the integration is complete, the **Running** status appears next to the integration name. The pod is running the integration.

8.3.2. Stopping integrations

Each integration can have exactly one version that is running. A running version is in the **Running** state. To stop running an integration:

1. In the left Fuse Online panel, click **Integrations**.

2. In the list of integrations, identify the entry for the integration that you want to stop running. The entry shows that this integration is **Running**.

3. At the far right of this integration's entry, click ⋮ and select **Stop**.

Fuse Online stops running the integration. **Stopping** and then **Stopped** appears in the integration's entry in the list of integrations.

8.3.3. Starting integrations

The first time that you start an integration, the process is referred to as publishing the integration because Fuse Online has to build the integration runtime before it can run the integration. See Section 8.3.1, "Publishing integrations".

If you stop running an integration and then you want to start it again, the steps are as follows:

1. In the left navigation panel, click **Integrations**.

2. In the list of integrations, to the right of the entry for the integration that you want to start, click ⋮ and select **Start**.

Fuse Online displays **Starting** as the status of that integration version, and then **Running** when the integration is running again.

8.3.4. Restarting older integration versions

You might publish an integration that does not work the way you want it to. In this situation, you can stop the incorrect version and replace it with a version that you published previously and that runs correctly.

To republish an older integration version:

1. In the left panel, click **Integrations** to display a list of the the integrations in this environment.
2. Click the entry for the integration for which you want to publish an older version. Fuse Online displays a list of the versions of the integration.
3. In the entry for the version that is running, at the far right, click ⋮ and select **Stop**.
4. Click **OK** to confirm that you want to stop running this version.
5. Wait for **Stopped** to appear to the right of the integration name near the top of the page.
6. Optionally, before you publish the older version, you can update it:
   a. In the entry for the integration version that you want to update, at the far right, click ⋮ and select **Replace Draft**.
   b. Update the integration as needed. For details, see Section 8.6, "Updating integrations" .
   c. When updates are complete, in the upper right, click **Publish**, and then click **Publish** to confirm. This takes the place of the next two steps.
7. To publish the older version as is, in the entry for the integration version that you want to start running again, at the far right, click ⋮ and select **Start**.
8. Click **Start** to confirm that you want to start this version of the integration.

Fuse Online starts the integration, which takes a few minutes. When the integration is running, then **Running version** ***n*** appears to the right of the integration's name.

8.3.5. Troubleshooting integration execution

If an integration stops working, check its logs and activity details. See Section 8.4.2, "Viewing integration log information" and Section 8.4.1, "Viewing integration history" .

For a connection to an application that uses OAuth, you might see an error message that indicates that the access token for the application has expired. Sometimes, you might get a less explicit **403 - Access denied** message. The information in the message depends on the application that the integration is connecting to. In this situation, try reconnecting to the application and then republishing the integration:

1. In the left panel, click **Integrations**.
2. In the list of integrations, click the entry for the integration that stopped running.
3. In the integration's summary page, in the visual integration flow, click the icon for the application that you want to reconnect to.
4. In the connection's details page, click **Reconnect**.
5. Respond to that application's OAuth workflow prompts.
   Fuse Online displays a message to indicate that its access to that application has been authorized. For some applications, this takes a few seconds but it can take longer for other applications.
6. After reconnecting to the application, start the integration.

If reconnection is not successful, try this:

1. Re-register Fuse Online as a client of the application. See Section 4.2, “Obtaining authorization to access applications”.
2. Create a new connection.
3. Edit each integration that was using the old connection:
   a. Remove the old connection.
   b. Replace it with the new connection.
4. Publish each updated integration.

8.3.6. Copying integrations to other environments

To run integrations across development, staging and production environments, you can export and import integrations. The environments can all be on a single OpenShift cluster, or they can be spread out across multiple OpenShift clusters. See the following topics:

- Section 8.3.6.1, “About copying integrations”
- Section 8.3.6.2, “Exporting integrations”
- Section 8.3.6.3, “Importing integrations”

8.3.6.1. About copying integrations

Each Fuse Online installation is an environment from which you can export an integration. Exporting an integration downloads a zip file that contains the information needed to recreate the integration in a different Fuse Online environment.

In an environment, each integration can have only one **Draft** version.

The result of importing an integration depends on:

- Whether the integration was previously imported
- Whether a connection that the integration uses was previously imported

Fuse Online uses an internal identifier for each integration and each connection to determine whether it already exists in the environment that it is being imported into. If you change the name of an integration or connection Fuse Online recognizes it as the same integration or connection, which just has a different name.

The following table describes the possible results of importing an integration:

In the importing environment:	What the import operation does:
The integration has not been previously imported.	Creates the integration. The integration is in the **Draft** state.

In the importing environment:	What the import operation does:
The integration has been previously imported.	Fuse Online updates the integration. The updated integration is in the **Draft** state. If there was a**Draft** version of this integration, it is lost.
The imported integration uses a connection that did not exist in the environment before the import operation.	Fuse Online creates a connection that has the same settings except for secrets. You must review each new connection. If a connection is not completely configured for its new environment then you must add the missing settings. For example, you might need to obtain secret settings by registering this installation of Fuse Online as a client of the application that this connection accesses.

8.3.6.2. Exporting integrations

When Fuse Online exports an integration it downloads a zip file to your local **Downloads** folder. This zip file contains the information needed to recreate the integration in a different Fuse Online environment.

To export an integration:

1. In the left panel of Fuse Online, click **Integrations**.

2. In the list of integrations, identify the entry for the integration that you want to export.

3. At the right of the entry, click ⋮ and select **Export**.

To import the integration into another Fuse Online environment, open that environment and import the exported zip file.

Exporting an integration is also a way to have a backup of the integration. However, Fuse Online maintains the versions of an integration so exporting an integration is not required for having a backup copy.

8.3.6.3. Importing integrations

In a Fuse Online environment, to import an integration from another Fuse Online environment, you must have used that other Fuse Online environment to export the integraion. Exporting an integration downloads the zip file that you upload to import the integration.

To import an integration:

1. Open the Fuse Online environment that you want to import the integration into.

2. In the left panel, click **Integrations**.

3. In the upper right, click **Import**.

4. Drag and drop one or more exported integration zip files, or navigate to a zip file that contains an exported integration and select it.

5. After Fuse Online imports the file(s), click **Done**. Fuse Online displays information about imported integrations.

6. In the left panel, click **Connections**.
   If an imported integration uses a connection that requires configuration, then there is a **Configuration Required** message at the bottom of the connection's card.

7. For each connection that requires configuration:

   a. Click it to display its details.

   b. Enter or change connection details as needed. It is possible that every field on this page is correct and that only security configuration is required.

   c. If you updated any fields, click **Save**.

   d. In the left panel, click **Settings**.
   The **Settings** page displays entries for applications that use the OAuth protocol.

8. For each connection that requires configuration and that accesses an application that uses the OAuth protocol, register this installation of Fuse Online with the application. The steps vary for each application. See the appropriate topic:

   - Register with Dropbox
   - Register with Gmail
   - Register with a REST API
   - Register with Salesforce
   - Register with Twitter

9. In the left panel, click **Connections** and confirm that there are no longer any connections that require configuration.

10. In the left panel, click **Integrations**. In the list of integrations, imported integrations have a green triangle in the upper left corner of their entries.

11. In the list of integrations, at the right of the entry for the integration that you imported, click ⋮ and select **Edit**.

12. In the upper right, click **Save as Draft** or, if you want to start running the imported integration, click **Publish**. Regardless of whether you save the integration as a draft or you publish the integration, Fuse Online updates the integration to use the updated connections.

8.4. MONITORING INTEGRATIONS

Fuse Online provides various ways for you to monitor the execution of your integrations. See:

- Section 8.4.1, "Viewing integration history"
- Section 8.4.2, "Viewing integration log information"
- Section 8.4.3, "Viewing metrics for a particular integration"
- Section 8.4.4, "Viewing metrics for a Fuse Online environment"

8.4.1. Viewing integration history

To view a list of the versions of an integration:

1. In the left panel, click **Integrations** to display a list of the integrations in your environment.

2. Click the entry for the integration whose versions you want to see.

In the page that appears, the **History** section lists the versions of the integration. The ✓ icon identifies the current version, which is the most recently, successfully running version. For each version, you can also see the date on which it was last started.

To edit, start, or stop a particular version, click the ⋮ to the right of the version’s entry. Select the operation you want to perform.

8.4.2. Viewing integration log information

Fuse Online provides log information for each integration version that it executes. To see this information:

1. In the left panel, click **Integrations**.

2. Click the entry for the integration for which you want to view log information.

3. In the integration’s summary page, click the **Activity** tab.

4. Optionally, enter date and/or keyword filters to limit the versions listed.

5. Click the integration version for which you want to view log information.

For each integration step, Fuse Online provides:

- The date and time that the step was executed
- How long it took to execute the step
- Whether execution was successful
- The error message if execution was not successful

To log additional information between any two steps, you can add a log step to the integration. A log step provides information about each message it receives and can provide custom text that you specify. If you add a log step, then it appears as one of the integration’s steps when you expand the integration version that you want to view log information for. You view Fuse Online information for a log step in the same way that you view Fuse Online information for any other step.

To add a log step, see Section 5.3.4, “Adding a log step” .

8.4.3. Viewing metrics for a particular integration

To view integration metrics:

1. In the left panel, click **Integrations**.

2. Click the entry for the integration for which you want to view metrics.

3. In the integration’s summary page, click **Metrics**.

Fuse Online provides the following metrics:

- **Total Errors** indicates the number of runtime errors that all executions of this integration encountered during the past 30 days.
- **Last Processed** displays the most recent date and time that this integration processed a message. The message might have been successfully processed or there might have been an error.
- **Total Messages** is the number of messages that all executions of this integration processed in the last 30 days. This includes message failures.
- **Uptime** indicates when this integration started running and how long it has been running without an error.

8.4.4. Viewing metrics for a Fuse Online environment

Metrics for your Fuse Online environment appear on the Fuse Online home page. To see them, click **Home** in the left panel. Fuse Online updates the following metrics every 5 seconds:

- The number of integrations that are defined in this environment as well as the number of integrations that are running, the number of integrations that are stopped, and the number of integrations that are pending. Fuse Online is either stopping or starting pending integrations. A red cross indicates any integrations that were running but that encountered an error that suspended execution.
- The number of connections that are defined in this environment.
- Total number of messages that have been processed by integrations in this environment in the last 30 days. This includes messages that were processed by integrations that might no longer be running or that might have been deleted from this environment.
- Uptime indicates how long there has been at least one integration that is running. The date and time when uptime started appears as well.

8.5. TESTING INTEGRATIONS

After you create an integration and it is running correctly in a development environment, you might want to run it in a different environment to test it.

To test an integration in a different Fuse Online environment:

1. See Section 8.3.6.1, “About copying integrations”.
2. Export the integration from the development environment. See Section 8.3.6.2, “Exporting integrations”.
3. Import the integration into the test environment. See Section 8.3.6.3, “Importing integrations”.

8.6. UPDATING INTEGRATIONS

After you create an integration, you might need to update it to add, edit or remove a step. To update an integration:

1. In the left Fuse Online panel, click **Integrations**.

2. In the list of integrations, click the entry for the integration that you want to update.

3. On the integration's summary page, in the upper right corner, click **Edit Integration**.

In the left panel, you can see that each step in the integration is represented by an icon that indicates whether it is a connection or a data operation between connections. Update the integration as needed:

- To add a step, in the left panel, hover over the plus sign that is in the location where you want to add it. Click **Add a Connection** or **Add a Step**.

- To delete a step, in the left panel, click 🗑 to the right of the step that you want to delete.

- To change the configuration of a step, in the left panel, click the step that you want to update. In the configuration page, update the parameter settings as needed.

See also: Section 8.3.1, "Publishing integrations".

8.7. DELETING INTEGRATIONS

To delete an integration:

1. In the left Fuse Online panel, click **Integrations**.

2. In the list of integrations, at the right of the entry for the integration that you want to delete, click ⋮ and select **Delete**.

3. In the popup, click **OK** to confirm that you want to delete the integration.

After you delete an integration, Fuse Online still has the history of that integration. If you import a version of the deleted integration, then Fuse Online associates the history of the deleted integration with the imported integration.

CHAPTER 9. INSTALLING FUSE ONLINE ON OPENSHIFT CONTAINER PLATFORM

You can install Fuse Online on OpenShift Container Platform (OCP) on premise. See the following topics for details:

- Section 9.1, “Overview of steps for installing Fuse Online on OCP”
- Section 9.2, “Register a custom resource definition”
- Section 9.3, “Procedure for installing or upgrading Fuse Online on OCP”

9.1. OVERVIEW OF STEPS FOR INSTALLING FUSE ONLINE ON OCP

To install Fuse Online on OCP on premise, the main steps are:

1. A user with cluster administration permissions:
   a. Registers a custom resource definition (CRD) at the cluster level.
   b. Grants permission for a user to install Fuse Online in their projects.
2. A user who is granted permission to install Fuse Online:
   a. Ensures that all prerequisites are met.
   b. Decides how to install Fuse Online with regard to the OpenShift project to install into, the OpenShift route for Fuse Online, and the level of accessibility of OpenShift logs.
   c. Downloads the installation script.
   d. Invokes the installation script with the command that implements decisions.
   e. Confirms that Fuse Online is running.

9.2. REGISTER A CUSTOM RESOURCE DEFINITION

To enable installation of Fuse Online, a cluster administrator registers a custom resource definition. The administrator needs to do this only once for the OpenShift cluster. The administrator also grants permission for installing Fuse Online to the appropriate users. Each user can then install Fuse Online in their projects.

Prerequisites

- You must have cluster administration permissions.
- You must be connected to the OCP cluster into which Fuse Online will be installed.

Procedure

1. Download the Fuse Online installation script to the current local directory by invoking the following command:
   $ wget https://raw.githubusercontent.com/syndesisio/fuse-online-install/1.4/install_ocp.sh

2. To verify that you are properly connected and can list custom resource definitions, invoke the following command:

   ```
   $ oc get crd
   ```

3. To register the custom resource definition at the cluster level, invoke the following command:

   ```
   $ bash install_ocp.sh --setup
   ```

4. Grant installation permission to each user who needs to install Fuse Online. For example, suppose you want to grant permission to an account with the user name **developer**. The following command grants permission to **developer** to install Fuse Online into any project that the **developer** account can access on the currently connected cluster:

   ```
   $ bash install_ocp.sh --grant developer --cluster
   ```

   Repeat this command for each user account that needs permission to install Fuse Online.

9.3. PROCEDURE FOR INSTALLING OR UPGRADING FUSE ONLINE ON OCP

To install Fuse Online on OCP on premise, you need to make some decisions about how you want to install Fuse Online. You then download the installation script, run it, and confirm that Fuse Online is installed.

Prerequisites

- You must be running OCP on premise.
- You must be connected to the OCP cluster in which you want to install Fuse Online.
- A user with cluster administration permissions must have given you permission to install Fuse Online in any project that you have permission to access in the cluster.

Decisions

To correctly specify the installation command, you need to decide on the answers to these questions:

- Do you want to install Fuse Online into the current OpenShift project or do you want to specify the project into which you want to install Fuse Online in the command that you invoke to do the installation?
  The default is that the installation script installs Fuse Online into the current project.

  You can install Fuse Online into a project that you specify. If this project does not yet exist, then the script creates it. If this project exists then the installation script prompts you to confirm that it is okay to delete the project’s content. To continue, you must confirm. The installation script then deletes the project and re-creates it.

- Do you want the installation script to calculate the OpenShift route by which Fuse Online can be reached or do you want to specify the OpenShift route in the installation command?
  The default is that the installation script calculates the route.

- Do you want to be able to access OpenShift log information for Fuse Online integrations by clicking a link in the Fuse Online user interface?
  While you can always access OpenShift logs manually, you might want to enable direct access by means of a link in the Fuse Online user interface. The default is that the installation script does not enable this link. To enable this link, you specify the URL for your OpenShift console when you invoke the installation script.

Download

Download the Fuse Online installation script to the current local directory by invoking the following command:

$ wget https://raw.githubusercontent.com/syndesisio/fuse-online-install/1.4/install_ocp.sh

Installation

To install Fuse Online into the current project, under the OpenShift route chosen by the installation script, without enabling the link from Fuse Online to OpenShift logs, invoke the following commands:

1. Log in with an account that has permission to install Fuse Online. For example:
   $ oc login developer

2. Ensure that the current project is the project into which you want to install Fuse Online:
   $ oc project

3. In the directory in which you downloaded the installation script, invoke the installation script as follows:
   $ bash install_ocp.sh

To install Fuse Online into a project that you specify, under an OpenShift route that you specify, and also enable the link from Fuse Online to OpenShift logs, invoke a command in which you:

- Specify the **--project** option. The argument is the name of the project in which you want to install Fuse Online.

- Specify the **--route** option. The argument is the URL that your Fuse Online environment will have. This URL will be where you access your installation of Fuse Online. This URL specifies the project in which you are installing Fuse Online followed by the domain that is specific to your OpenShift cluster.

- Replace the **--console** argument with the URL for your OpenShift console.

For example:

```
$ bash install_ocp.sh \
   --project my-project \
   --route my-project.6a63.fuse-online.openshiftapps.com \
   --console https://console.fuse-online.openshift.com/console
```

According to how you want to install Fuse Online, you can specify

- All three options

- Any two options

- Any one of the options

- No options

The installation script uses the default for an option that you do not specify.

Upgrading

If Fuse Online is installed on your OpenShift cluster, you can upgrade it to the new version by using the same procedure as for installing Fuse Online. During installation, the Fuse Online installer automatically scans the namespace for Fuse Online installations. If the installer finds an older version then it triggers an infrastructure upgrade.

Any existing integrations continue to run both during and after the upgrade. The existing integrations continue to run with the *older* versions of Fuse Online libraries and dependencies. After the infrastructure upgrade, you can upgrade an integration by republishing it. In Fuse Online, select the integration that you want to upgrade, select **Edit**, and publish it again. This step forces a rebuild that uses the latest Fuse Online dependencies.

Upgrading Fuse Online images

From time to time, fresh docker-formatted images are released for Fuse Online, in order to incorporate patches and security fixes. You will be notified of these updates through Red Hat's errata update channel. To upgrade the Fuse Online images, download the latest **install_ocp.sh** script, as follows:

```
$ wget https://raw.githubusercontent.com/syndesisio/fuse-online-install/1.4/install_ocp.sh
```

And follow the instructions for upgrading.

NOTE

The **install_ocp.sh** script on the **1.4** branch is updated at the time of the images release, so when you run the latest **install_ocp.sh** script, it updates the Fuse Online images to use the latest available images.

Confirm installation

To confirm that installation was successful:

1. Display the OpenShift OAuth proxy log-in page at **https://openshift-route**
   If you specified the **--route** option when you ran the installation script, replace **openshift-route** with the route name that you specified. If you chose to let the installation script calculate the OpenShift route, then the the script displays the calculated route near the end of its execution. Replace **openshift-route** with the value that the script provided.

2. If you are not already logged in to the OpenShift console, its log-in page appears. Enter your OpenShift user name and password to log in.

The Fuse Online home page appears either immediately or after you log in to the OpenShift console.

Deleting your Fuse Online project

If you want to delete your Fuse Online project, invoke the **delete project** command. For example, to delete a project whose name is **fuseonline**, enter the following command:

```
$ oc delete project fuseonline
```

Uninstalling Fuse Online

To uninstall Fuse Online without deleting its project nor anything else in that project, invoke the following command:

```
$ oc delete syndesis app
```

This command deletes the Fuse Online infrastructure but it does not delete running integrations. Integrations that are running continue to run and execute as implemented.