



Solace PubSub+ Connector for SFTP

User Guide

Solace Corporation

Version 1.1.9



Table of Contents

Preface	1
Getting Started	2
Prerequisites	2
Quick Start common steps	2
Quick Start: Running the connector via command line	2
Quick Start: Running the connector via <code>start.sh</code> script	3
Quick Start: Running the connector as a Container	6
Enabling Workflows	7
Configuring Connection Details	8
Solace PubSub+ Connection Details	8
Preventing Message Loss when Publishing to Topic-to-Queue Mappings	8
Connecting to Multiple Systems	8
User-configured Header Transforms	9
User-configured Payload Transforms	11
Registered Functions	11
Message Headers	13
Solace Headers	13
Reserved Message Headers	13
Dynamic Producer Destinations	14
Asynchronous Publishing	15
Management and Monitoring Connector	16
Monitoring Connector's States	16
Exposed HTTP/HTTPS Endpoints	16
Health	18
Workflow Health	18
Solace Binder Health	19
Leader Election	20
Leader Election Modes: Standalone	20
Leader Election Mode: Active-Standby	20
Leader Election Management Endpoint	21
Workflow Management	22
Workflow Management Endpoint	22
Workflow States	23
Metrics	24
Connector Meters	24
Add a Monitoring System	25
Security	27
Securing Endpoints	27

Exposed Management Web Endpoints.....	27
Authentication & Authorization.....	27
TLS.....	28
Consuming Object Messages.....	29
Adding External Libraries.....	30
Configuration.....	31
Providing Configuration.....	31
Converting Canonical Spring Property Names to Environment Variables.....	31
Spring Profiles.....	31
Configure Locations to Find Spring Property Files.....	31
Obtaining Build Information.....	32
Spring Configuration Options.....	32
Connector Configuration Options.....	33
Workflow Configuration Options.....	34
SFTP Checkpoint Store Configuration options.....	36
SFTP Source Configuration Options.....	37
File Type Specific Properties.....	46
SFTP Source Headers.....	49
SFTP Target Configuration Options.....	50
SFTP Target Headers.....	57
Logging.....	58
Configuring Logback.....	58
License.....	59
Support.....	59

Preface

Solace PubSub+ Connector for SFTP bridges data between the Solace PubSub+ Event Broker and SFTP servers, enabling seamless and efficient integration of file-based data with your Solace-backed, event-driven architecture and Event Mesh. The connector supports bidirectional data flow, allowing you to read files from SFTP servers, split and process their contents, and publish data to Solace, as well as write Solace events to SFTP servers. The connector can be deployed in standalone or active-standby modes. Each instance supports up to 20 workflows (file-to-message or message-to-file pipelines), reducing the number of connector instances required. The use of various Spring Framework technologies allows for easy configuration of the connector, advanced logging capabilities, and export of live metrics data to external monitoring solutions.

Getting Started

Presuming you're using the default `application.yml` provided with the download, follow one of the below quick start to connect a PubSub+ event broker with the SFTP Server. The quick starts use default credentials as examples to get started with two workflows enabled, workflow 0 and workflow 1.

Where:

- Workflow 0 consumes messages from the Solace PubSub+ queue, `Solace/Queue/0`, and publishes them to the SFTP server, `sftp://localhost:22/startDirectory`.
- Workflow 1 consumes messages from the SFTP server, `sftp://localhost:22/startDirectory`, and publishes them to the Solace PubSub+ topic, `Solace/Topic/1`.

A workflow is the configuration of a flow of messages from a source to a target. The connector supports up to 20 concurrent workflows per instance.



The connector does not provision queues that do not exist.

Prerequisites

- [Solace PubSub+ Event Broker](#)
- SFTP server with password or private key authentication

Quick Start common steps

These are the steps that are required to run all quick-start examples:

1. Update the provided `samples/config/application.yml` with the values for your deployment.

Quick Start: Running the connector via command line

Run:

```
java -jar pubsubplus-connector-sftp-1.1.9.jar --spring.config.additional-location
=file:samples/config/
```



By default, this command detects any Spring Boot configuration files as per the [Spring Boot's default locations](#).

For more information, see [Configure Locations to Find Spring Property Files](#).

Quick Start: Running the connector via **start.sh** script

For convenience, you can start the connector through the shell script using the following command:

```
chmod 744 ./bin/start.sh
./bin/start.sh [-n NAME] [-l FOLDER] [-p PROFILE] [-c FOLDER] [-ch HOST] [-cp PORT] [-j FILE] [-cm] [-cmh HOST] [-cmp PORT] [-mh HOST] [-mp PORT] [-o OPTIONS] [-b]
```

The script shows you all errors at the same time:

```
./bin/start.sh -l dummy_folder -c dummy_folder -j dummy_file.jar
```

The script shows you all errors at the same time:

Solace PubSub+ Connector for SFTP

Connector startup failed:

```
Following folder doesn't exists on your filesystem: 'dummy_folder'
Following folder doesn't exists on your filesystem: 'dummy_folder'
Following file doesn't exists on your filesystem: 'dummy_file.jar'
```

In situations where you don't provide a parameter, the script runs with the predefined values as follows:

Parameter	Default Value	Description
-n, --name	application	The name of the connector instance, that is configured in [spring.application.name]. This name impacts on grouping connectors only.
-l, --libs	./libs	The directory that contains the required and optional dependency JAR files, such as Micrometer metrics export dependencies (if configured). If this option is not specified, it will use the current ./libs/ directory.

Parameter	Default Value	Description
<code>-p, --profile</code>	empty, no profile is used	The profile to be used with the connector's configuration. The configuration file named 'application-<profile>.yaml' is used. If this option is not specified, no profile is used.
<code>-c, --config</code>	<code>./</code> or current folder	The path to the folder containing the configuration files to be applied when the connector starts up the chosen profile. If not specified, the current directory is used.
<code>-H, --host</code>	127.0.0.1	Specifies the host where the connector runs.
<code>-P, --port</code>	8090	Specifies the port where connector runs.
<code>-mp, --mgmt_port</code>	9009	Specifies the management port for back calls of current connector from PubSub+ Connector Manager. This parameter is ignored if the <code>-cm</code> parameter is not provided.
<code>-j, --jar</code>	pubsubplus-connector-sftp-1.1.9.jar	The path to the specified JAR file to start the connector. If the option is not specified, the default JAR file is used from the current directory.

Parameter	Default Value	Description
<code>-cm, --manager</code>	<code>application</code>	Specifies PubSub+ Connector Manager to use the configuration storage and allows you to enable the cloud configuration for the connector. When this parameter is enabled, you can specify the <code>-mp</code> or <code>--mgmt_port</code> , <code>-H</code> or <code>--host</code> , and <code>-cmh</code> with the <code>-cmp</code> parameters, unless you want to use default values for those parameters. Be aware, this option disable listed parameters to be read from configuration file. In this case, the operator must explicitly specify the parameters for the script, otherwise defaultdefault values are used.
<code>-cmh, --cm_host</code>	<code>127.0.0.1</code>	Specifies the host where Connector Manager is running. This parameter is ignored if the <code>-cm</code> parameter is not provided.
<code>-cmp, --cm_port</code>	<code>9500</code>	Specifies the port where Connector Manager is running. This parameter is ignored if <code>-cm</code> parameter is not provided.
<code>-o, --options</code>	<code>no default values</code>	Specifies the JVM options used on when the connector starts. For example, <code>-Xms64M -Xmx1G</code> .
<code>-tls</code>	<code>N/A</code>	Specifies to use HTTPS instead of HTTP. . When this parameter is used, the configuration file must contain an additional section with the preconfigured paths for the key store and trust store files.
<code>-s, --show</code>	<code>N/A</code>	Performs a dry run (does nothing). The output prints the start CLI command and its raw output and exits. This parameter is useful to check your parameters without running the connector.

Parameter	Default Value	Description
<code>-b, --background</code>	N/A	Runs the connector in the background. No logs are shown and the connector continues running in detached mode.
<code>-h, --help</code>	N/A	Prints the help information and exits.

Script also provides that help information from command line using parameter `-h`.

More configuration example of starting Connector together with Connector Manager are provided by the Connector Manager samples.

Quick Start: Running the connector as a Container

The following steps show how to use the sample docker compose file that has been included in the package:

1. Change to the `docker` directory:

```
cd samples/docker
```

This directory contains both the `docker-compose.yml` file as well as an `.env` file that contains environment secrets required for the container's health check.

2. Run the connector:

```
docker-compose up -d
```

This sample docker compose file will:

- Exposes the connector's `8090` web port to `8090` on the host.
- Connects a PubSub+ event broker and SFTP exposed on the host using default ports.
- Mounts the `samples/config` directory.
- Mounts the previously defined `libs` directory.
- Creates a `healthcheck` user with read-only permissions.
 - The default username and password for this user can be found within the `.env` file.
 - This user overrides any users you have defined in your `application.yml`. See [here](#) for more information.
- Uses the connector's management health endpoint as the container's health check.

For more information about how to use and configure this container, see [the connector's container documentation](#).

Enabling Workflows

The provided `application.yml` enables workflow 0 and 1. To enable additional workflows, define the following properties in the `application.yml`, where `<workflow-id>` is a value between `[0-19]`:

```
spring:
  cloud:
    stream:
      bindings: # Workflow bindings
        input-<workflow-id>:
          destination: <input-destination>
          binder: (solace|camel) # Input system
        output-<workflow-id>:
          destination: <output-destination>
          binder: (solace|camel) # Output system

solace:
  connector:
    workflows:
      <workflow-id>:
        enabled: true
```



The connector only supports workflows in the directions of:

- `solace` → `SFTP`
- `SFTP` → `solace`

For more information about Spring Cloud Stream and the Solace PubSub+ binder, see:

- [Spring Cloud Stream Reference Guide](#)
- [Spring Cloud Stream Binder for Solace PubSub+](#)

Configuring Connection Details

Solace PubSub+ Connection Details

The Spring Cloud Stream Binder for PubSub+ uses [Spring Boot Auto-Configuration for the Solace Java API](#) to configure its session.

In the `application.yml`, this typically is configured as follows:

```
solace:
  java:
    host: tcp://localhost:55555
    msg-vpn: default
    client-username: default
    client-password: default
```

For more information and options to configure the PubSub+ session, see [Spring Boot Auto-Configuration for the Solace Java API](#).

Preventing Message Loss when Publishing to Topic-to-Queue Mappings

If the connector is publishing to a topic that is subscribed to by a queue, messages may be lost if they are rejected. For example, if queue ingress is shutdown.

To prevent message loss, configure `reject-msg-to-sender-on-discard` with the `including-when-shutdown` flag.

Connecting to Multiple Systems

The connector does not support multiple binder system configurations. The multiple binder systems configuration may be supported in future releases.



- The connector does not support multiple binder system configurations.
- The multiple binder systems configuration may be supported in future releases.

User-configured Header Transforms

Deprecated

This feature is deprecated and will be removed in a future release. Use [Message Transforms](#) instead.

Generic Migration Rules:

1. Configuration Structure Changes:

- a. Replace `solace.connector.workflows.<n>.transform-headers` with `solace.connector.workflows.<n>.transform` and within it:
 - i. Add `enabled: true` to enable transforms
 - ii. Move `transform-headers.expressions` to the `transform.expressions` list property

2. Expression Syntax Changes:

- Replace direct header references, `headers.<headerName>`, with `source['headers']['<headerName>']` for reading
- Use `target['headers']['<headerName>'] = ...` for writing
- Replace Java methods (`T(String)`, etc.) with built-in functions:
 - `T(String).join()` → `#joinString()`
 - `split()` → `#splitString()`
 - `toUpperCase()` → `#upperCaseString()`
 - `toLowerCase()` → `#lowerCaseString()`
 - etc



Example Migration:

Old Configuration:

```
solace:
  connector:
    workflows:
      0:
        transform-headers:
          expressions:
            route: "T(String).format('%s/%s', headers.region,
headers.status)"
            count: "headers.count.toString()"

```

New Configuration:

```
solace:

```

```
connector:
  workflows:
    0:
      transform:
        enabled: true
        expressions:
          - transform: "target['headers']['route'] = #joinString('/',
source['headers']['region'], source['headers']['status'])"
          - transform: "target['headers']['count'] =
#convertNumberToString(source['headers']['count'])"
```

Generally, the consumed message's headers are propagated through the connector to the output message. If you want to transform the headers, then you can do so as follows:

```
# <workflow-id> : The workflow ID ([0-19])
# <header> : The key for the outbound header
# <expression> : A SpEL expression which has "headers" as parameters
```

```
solace.connector.workflows.<workflow-id>.transform-
headers.expressions.<header>=<expression>
```

Example 1: To create a new header, `new_header`, for workflow `0` that is derived from the headers `foo` & `bar`:

```
solace.connector.workflows.0.transform-headers.expressions.new_header
="T(String).format('%s/abc/%s', headers.foo, headers.bar)"
```

Example 2: To remove the header, `delete_me`, for workflow `0`, set the header transform expression to `null`:

```
solace.connector.workflows.0.transform-headers.expressions.delete_me="null"
```

For more information about Spring Expression Language (SpEL) expressions, see [Spring Expression Language \(SpEL\)](#).

User-configured Payload Transforms



Deprecated

This feature is deprecated and will be removed in a future release. Use [Message Transforms](#) instead.

Message payloads going through a workflow can be transformed using a Spring Expression Language (SpEL) expression as follows:

```
# <workflow-id> : The workflow ID ([0-19])
# <expression> : A SpEL expression

solace.connector.workflows.<workflow-id>.transform-payloads.expressions[0].transform
=<expression>
```

A SpEL expression may reference:

- **payload**: To access the message payload.
- **headers.<header_name>**: To access a message header value.
- Registered functions.



While the syntax uses an array of expressions, only a single transform expression is supported in this release. Multiple transform expressions may be supported in the future.

Registered Functions

Registered functions are built-in and can be called directly from SpEL expressions. To call a registered function, use the **#** character followed by the function name. The following table describes the available registered functions:

Registered Function Signature	Description
<code>boolean isPayloadBytes(Object obj)</code>	<p>Returns whether the object <code>obj</code> is an instance of <code>byte[]</code> or not.</p> <p>Sample usage of this function within a SpEL expression: <code>"#isPayloadBytes(payload) ? true : false"</code></p>

Example 1: To normalize `byte[]` and `String` payloads as upper-cased `String` payloads or leave payloads unchanged when of different types:

```
solace.connector.workflows.0.transform-payloads.expressions[0].transform
="#isPayloadBytes(payload) ? new String(payload).toUpperCase() : payload instanceof
```

```
T(String) ? payload.toUpperCase() : payload"
```

Example 2: To convert `String` payloads to `byte[]` payloads using a `charset` retrieved from a message header or leave payloads unchanged when of different types:

```
solace.connector.workflows.0.transform-payloads.expressions[0].transform="payload  
instanceof T(String) ?  
payload.getBytes(T(java.nio.charset.Charset).forName(headers.charset)) : payload"
```

For more information about Spring Expression Language (SpEL) expressions, see [Spring Expression Language \(SpEL\)](#).

Message Headers

Solace headers can be created or manipulated using the [User-configured Header Transforms](#) feature described above.

Solace Headers

Solace headers exposed to the connector are documented in the [Spring Cloud Stream Binder for Solace PubSub+](#) documentation.

Reserved Message Headers

The following are reserved header spaces:

- `solace_`
- `scst_`
- Any headers defined by the core Spring messaging framework. See [Spring Integration: Message Headers](#) for more info.

Any headers with these prefixes (that are not defined by the connector or any technology used by the connector) may not be backwards compatible in future releases of this connector.

Dynamic Producer Destinations

To route messages to dynamic destinations at runtime, use the [Message Transform](#) feature to set the following headers:

Header Name	Type	Values	Applies To	Description
<code>scst_targetDestination</code>	<code>string</code>	Any valid destination name	Solace, SFTP	Specifies the name of the dynamic destination to publish to. Setting this header overrides the configured destination.
<code>solace_scst_targetDestinationType</code>	<code>string</code>	<code>(queue topic)</code>	Solace	Specifies the destination type of the dynamic destination. When unspecified, the configured or default destination type is used.



Setting the `scst_targetDestination` header under `solace.connector.default.workflow.*` may not be viable if not all workflows follow the same direction.

Asynchronous Publishing

This connector does not support asynchronous publishing. Publish acknowledgments are resolved synchronously for all workflows regardless of the config option:

```
# <workflow-id> : The workflow ID ([0-19])
```

```
solace.connector.workflows.<workflow-id>.acknowledgment.publish-async=(true|false)
```



Enabling **publish-async** enable asynchronous publishing on the connector's core, but the effective publishing mode is still synchronous because there is no support for this feature on either the consumer binding or the producer binding.

Management and Monitoring Connector

Monitoring Connector's States

The connector provides an ability to monitor its internal states through exposed endpoints provided by [Spring Boot Actuator](#).

An Actuator shares information through the endpoints reachable over HTTP/HTTPS. The endpoints that are available are configured in the connector configuration file.

What endpoints are available is configured in the connector configuration file:

```
management:
  simple:
    metrics:
      export:
        enabled: true
    endpoints:
      web:
        exposure:
          include:
            "health,metrics,loggers,logfile,channels,env,workflows,leaderelection,bindings,info"
```

The above sample configuration enables metrics collection through the configuration parameter of `management.simple.metrics.export.enabled` set to `true` and then shares them through the HTTP/HTTPS endpoint together with other sections configured for the current connector.

Exposed HTTP/HTTPS Endpoints

The set of endpoints exposed through the HTTP/HTTPS endpoint.

- Exposed endpoints are available if you query the endpoints using the web interface (for example `https://localhost:8090/actuator/<some_endpoint>`) and also available in PubSub+ Connector Manager.
- The operator may choose to not expose all or some of these endpoints. If so, the Actuator endpoints that are not exposed are not visible if you query the endpoints (for example, `https://localhost:8090/actuator/<some_endpoint>`) nor in PubSub+ Connector Manager.



The simple metrics registry is only to be used for testing. It is not a production-ready means of collecting metrics. In production, use a dedicated monitoring system (for example, Datadog, Prometheus, etc.) to collect metrics.

The Actuator endpoint now contains information about Connector's internal states shared over the following HTTP/HTTPS endpoint:

```
GET: /actuator/
```

The following shows an example of the data shared with the configuration above:

```
{
  "_links": {
    "self": {
      "href": "/actuator",
      "templated": false
    },
    "workflows": {
      "href": "/actuator/workflows",
      "templated": false
    },
    "workflows-workflowId": {
      "href": "/actuator/workflows/{workflowId}",
      "templated": true
    },
    "leaderelection": {
      "href": "/actuator/leaderelection",
      "templated": false
    },
    "health-path": {
      "href": "/actuator/health/{*path}",
      "templated": true
    },
    "health": {
      "href": "/actuator/health",
      "templated": false
    },
    "metrics": {
      "href": "/actuator/metrics",
      "templated": false
    },
    "metrics-requiredMetricName": {
      "href": "/actuator/metrics/{requiredMetricName}",
      "templated": true
    }
  }
}
```

Health

The connector reports its health status using the [Spring Boot Actuator health endpoint](#).

To configure the information returned by the `health` endpoint, configure the following properties:

- `management.endpoint.health.show-details`
- `management.endpoint.health.show-components`

For more information, about health endpoints, see [Spring Boot documentation](#).

Health for the workflow, Solace binder, and camel binder components are exposed when `management.endpoint.health.show-components` is enabled. For example:

```
management:
  endpoint:
    health:
      show-components: always
      show-details: always
```

This configuration would always show the full details of the health check including the workflows and binders. The default value is `never`.

Workflow Health

A `workflows` health indicator is provided to show the health status for each of a connector's workflows. This health indicator has the following form:

```
{
  "status": "(UP|DOWN)",
  "components": {
    "<workflow-id>": {
      "status": "(UP|DOWN)",
      "details": {
        "error": "<error message>"
      }
    }
  }
}
```

Health Status	Description
UP	A status that indicates the workflow is functioning as expected.
DOWN	A status that indicates the workflow is unhealthy. Operator intervention may be required.

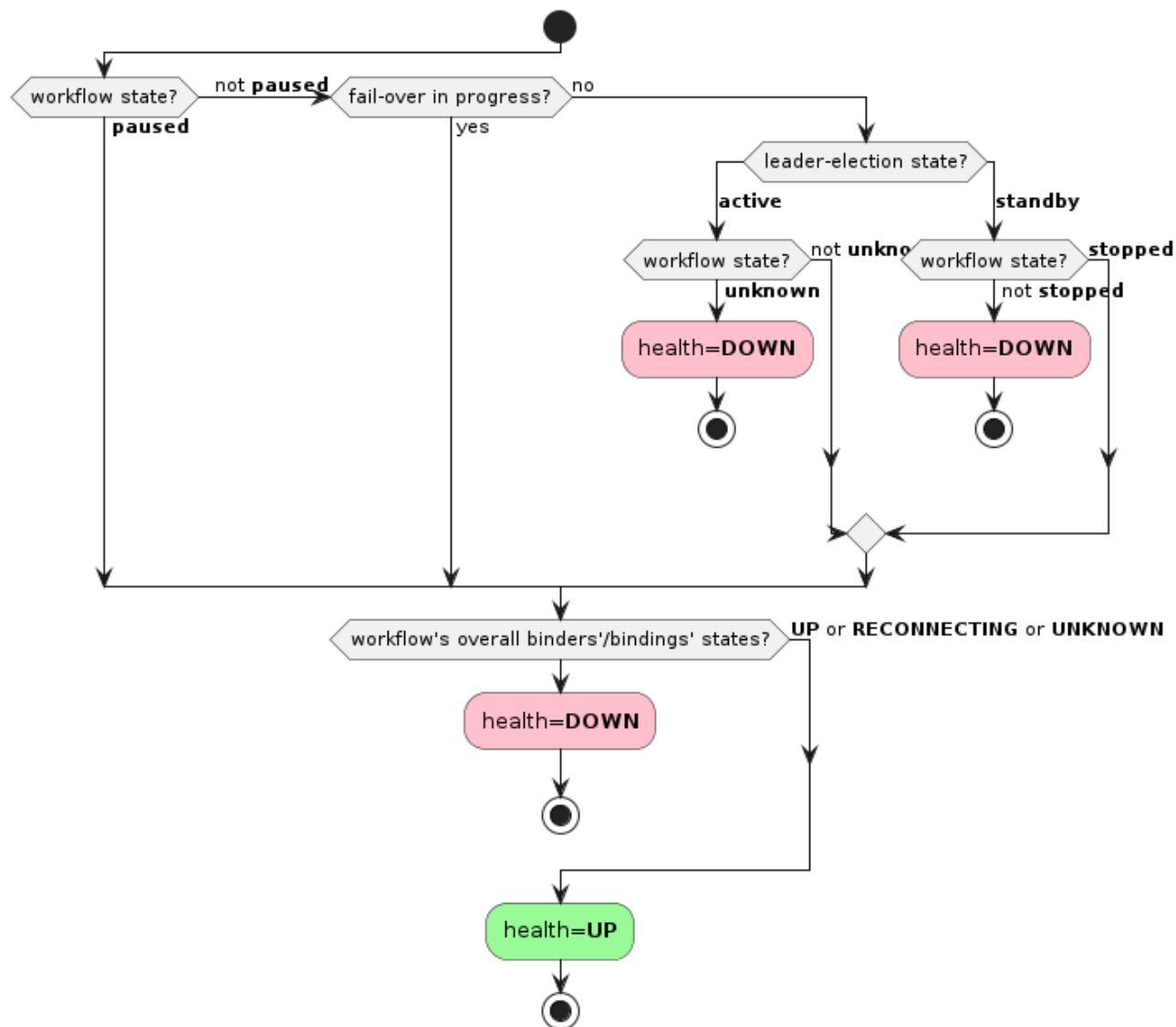


Figure 1. Workflow Health Resolution Diagram

This health indicator is enabled default. To disable it, set the property as follows:

```
management.health.workflows.enabled=false
```

Solace Binder Health

For details, see the [Solace binder](#) documentation.

Leader Election

The connector has two leader election modes for redundancy:

Leader Election Mode	Description
Standalone (Default)	A single instance of a connector without any leader election capabilities.
Active-Standby	A participant in a cluster of connector instances where only one instance is active (i.e. the leader), and the others are standby.

Operators can configure the leader election mode by setting the following configuration:

```
solace.connector.management.leader-election.mode=(standalone|active_standby)
```

Leader Election Modes: Standalone

When the connector starts, all enabled workflows start at the same time. The connector itself is considered as always active.

Leader Election Mode: Active-Standby

If the connector is in active-standby mode, a PubSub+ management session and management queue must be configured as follows:

```
solace.connector.leader-election.mode=active_standby

# Management session
# Exact same interface as solace.java.*
solace.connector.management.session.host=<management-host>
solace.connector.management.session.msgVpn=<management-vpn>
solace.connector.management.session.client-username=<client-username>
solace.connector.management.session.client-password=<client-password>
solace.connector.management.session.<other-property-name>=<value>

# Management queue name accessible by the management session
# Must have exclusive access type
solace.connector.management.queue=<management-queue-name>
```

To determine if the connector is **active** or **standby**, it creates a flow to the management queue. If this flow is active, then the connector's state is **active** and will start its enabled workflows. Otherwise, if this flow is inactive, then the connector's state is **standby** and will stop its enabled workflows.

At a macro level for a cluster of connectors, failover only happens when there are infrastructure failures (for example, the JVM goes down or networking failures to the management queue).

If a workflow fails to start or stop during failover, it will retry up to some maximum defined by the

configuration option, `solace.connector.management.leader-election.fail-over.max-attempts`.

During failover, the connector attempts to start or stop all enabled workflows. After an attempt has been made to start or stop each workflow, the connector transitions to the active/standby mode regardless of the status of the workflows.

Leader Election Management Endpoint

A custom `leaderelection` management endpoint was provided using [Spring Actuator](#).

Operators can navigate to the connector's `leaderelection` management endpoint to view its leader election status.

Endpoint	Operation	Payloads
<code>/leaderelection</code>	Read (HTTP <code>GET</code>)	<p>Request: None.</p> <p>Response:</p> <pre> { "mode": { "type": "(standalone active_active ① active_standby)", "state": "(active standby)", ② "source": { ③ "queue": "<management-queue-name>", "host": "<management-host>", "msgVpn": "<management-vpn>" } } } </pre> <p>① Mandatory parameter in output</p> <p>② Mandatory parameter in output</p> <p>③ Optional section. Appears only when <code>type</code> is set to <code>active_standby</code>.</p>

Workflow Management

Workflow Management Endpoint

A custom `workflows` management endpoint using [Spring Actuator](#) is provided to manage workflows.

To enable the `workflows` management endpoint:

```
management:
  endpoints:
    web:
      exposure:
        include: "workflows"
```

Once the `workflows` management endpoint is enabled, the following operations can be performed:

Endpoint	Operation	Payloads
<code>/workflows</code>	Read (HTTP <code>GET</code>)	Request: None. Response: Same payload as the <code>/workflows/{workflowId}</code> read operation, but as a list of all workflows.
<code>/workflows/{workflowId}</code>	Read (HTTP <code>GET</code>)	Request: None. Response: <pre>{ "id": "<workflowId>", "enabled": (true false), "state": "(running stopped paused unknown)", "inputBindings": ["<input-binding>"], "outputBindings": ["<output-binding>"] }</pre>
<code>/workflows/{workflowId}</code>	Write (HTTP <code>POST</code>)	Request: <pre>{ "state": "STARTED STOPPED PAUSED RESUMED" }</pre> Response: None.



Only workflows with Solace PubSub+ consumers (where the **solace** binder is defined in the **input-#**) support pause/resume.



Some features require for the connector to manage workflow lifecycles. There's no guarantee that workflow states continue to persist when write operations are used to change the workflow states while such features are in use.

For example: When the connector is configured in the active-standby leader election mode, workflows will automatically transition from **running** to **stopped** when the connector fails over from **active** to **standby**. Vice-versa for a failover in the opposite direction.

Workflow States

A workflow's state is defined as the aggregate states of its bindings (see the [bindings management endpoint](#)) as follows:

Workflow State	Condition
running	All bindings have state="running" .
stopped	All bindings have state="stopped" .
paused	All consumer bindings and all pausable producer bindings have state="paused" .
unknown	None of the other states. Represents an inconsistent aggregate binding state.



When the producer or consumer binding is not implementing Spring's Lifecycle interface, Spring always reports the bindings as **state=N/A**. The **state=N/A** is ignored when deciding the overall state of the workflow. For example, if the consumer's binding is **state=running** and producer's binding **state=N/A** (or vice-versa), the workflow state would be **running**.

For more information about binding states, see [Spring Cloud Stream: Binding visualization and control](#).

Metrics

This connector uses [Spring Boot Metrics](#) that leverages Micrometer to manage its metrics.

Connector Meters

In addition to the meters already provided by the Spring framework, this connector introduces the following custom meters:

Name	Type	Tags	Description	Notes
<code>solace.connector.process</code>	Timer	type: channel name: <bindingName> result: (success failure) exception: (none exception simple class name)	The processing time.	This meter is a rename of <code>spring.integration.send</code> whose <code>name</code> tag matches a binding name.
<code>solace.connector.error.process</code>	Timer	type: channel name: <bindingNames> result: (success failure) exception: (none exception simple class name)	The error processing time.	This meter is a rename of <code>spring.integration.send</code> whose <code>name</code> tag matches an input binding's error channel name (<destination>.<group>.errors). Meters might be merged under the same <code>name</code> tag (delimited by) if multiple bindings have the same error channel name (for example, bindings can have a matching <code>destination</code> , <code>group</code> , or both). NOTE: Setting a binding's <code>group</code> is not supported.
<code>solace.connector.message.size.payload</code>	DistributionSummary Base Units: bytes	name: <bindingName>	The message payload size.	

Name	Type	Tags	Description	Notes
<code>solace.connector.message.size.total</code>	DistributionSummary Base Units: bytes	name: <bindingName>	The total message size.	
<code>solace.connector.publish.ack</code>	Counter Base Units: acknowledgments	name: <bindingName> result: (success failure) exception: (none exception simple class name)	The publish acknowledgment count.	
<code>solace.connector.transform.expressions.count</code>	Gauge Base Units: expressions	workflow.id: <workflowId>	Transformation expressions count	
<code>solace.connector.transform.time</code>	Timer	workflow.id: <workflowId> result: (success failure)	Transformation execution time	This is the aggregate time for all expressions used to transform a single message.



The `solace.connector.process` meter with `result=failure` is not a reliable measure of tracking the number of failed messages. It only tells you how many times a step processed a message (or batch of messages), how long it took to process that message, and if that step completed successfully.

Instead, we recommend that you use a combination of `solace.connector.error.process` and `solace.connector.publish.ack` to track failed messages.

Add a Monitoring System

By default, this connector includes the following monitoring systems:

- [Datadog](#)
- [Dynatrace](#)
- [Influx](#)
- [JMX](#)
- [OpenTelemetry \(OTLP\)](#)

- [StatsD](#)

To add additional monitoring systems, add the system's `micrometer-registry-<system>` JAR file and its dependency JAR files to [the connector's classpath](#). The included systems can then be individually enabled/disabled by setting `management.<system>.metrics.export.enabled=true` in the `application.yml`.

Security

Securing Endpoints

Exposed Management Web Endpoints

There are many endpoints that are automatically enabled for this connector. For a comprehensive list, see [Management and Monitoring Connector](#).

The **health** endpoint only returns the root status by default (i.e. no health details).

To enable other management endpoints, see [Spring Actuator Endpoints](#).

Authentication & Authorization

This release of the connector only supports basic HTTP authentication.

By default, no users are created unless the operator configures them in their configuration file. The configuration parameters responsible for security are as follows:

```
solace:
  connector:
    security:
      enabled: true
      users:
        - name: user1
          password: pass
        - name: admin1
          password: admin
      roles:
        - admin
```

In the above example, we have created two users:

- **user1**: Has access to perform GET (Read) requests.
- **admin1**: Has access to perform GET and POST (Read & Write) requests.

To fully disable security and permit anyone to access the connector's web endpoints, operators can configure the `solace.connector.security.enabled` parameter **false**.



While these properties could be defined in an `application.yml` file, we recommend that you use environment variables to set secret values.

The following example shows you how to define users using environment variables:

```
# Create user with no role (i.e. read-only)
SOLACE_CONNECTOR_SECURITY_USERS_0_NAME=user1
```

```
SOLACE_CONNECTOR_SECURITY_USERS_0_PASSWORD=pass

# Create user with admin role
SOLACE_CONNECTOR_SECURITY_USERS_1_NAME=admin1
SOLACE_CONNECTOR_SECURITY_USERS_1_PASSWORD=admin
SOLACE_CONNECTOR_SECURITY_USERS_1_ROLES_0=admin
```

In the above example, we have created two users:

- **user1**: Has access to perform GET (Read) requests.
- **admin1**: Has access to perform GET and POST (Read & Write) requests.



`solace.connector.security.users` is a list. When users are defined in multiple sources (different `application.yml` files, environment variables, and so on), overriding works by replacing the entire list. In other words, you must pick one place to define all your users, whether in a **single** application properties file or as environment variables.

For more information, see [Spring Boot - Merging Complex Types](#).

TLS

TLS is disabled by default.

To configure TLS, see [Spring Boot - Configure SSL](#) and [TLS Setup in Spring](#).

Consuming Object Messages

For the connector to process object messages, it needs access to the classes which define the object payloads.

Assuming that your payload classes are in their own project(s) and are packaged into their own jar(s), place these jar(s) and their dependencies (if any) onto [the connector's classpath](#).



It is recommended that these jars only contain the relevant payload classes to prevent any oddities.

In the jar(s), your class files must be archived in the same directory/classpath as the application that publishes them.



e.g. If the source application is publishing a message with payload type, `MySerializablePayload`, defined under classpath `com.sample.payload`, then when packaging the payload jar for the connector, the `MySerializablePayload` class must still be accessible under the `com.sample.payload` classpath.

Typically, build tools such as Maven or Gradle will handle this when packaging jars.

Adding External Libraries

The connector jar uses the `loader.path` property as the recommended mechanism for adding external libraries to the connector's classpath.

See [Spring Boot - PropertiesLauncher Features](#) for more info.

To add libraries to the connector's container image, see [the connector's container documentation](#).

Configuration

Providing Configuration

For information about about how the connector detects configuration properties, see [Spring Boot: Externalized Configuration](#).

Converting Canonical Spring Property Names to Environment Variables

For information about converting the Spring property names to environment variables, see the [Spring documentation](#).

Spring Profiles

If multiple configuration files exist within the same configuration directory for use in different environments (development, production, etc.), use Spring profiles.

Using Spring profiles allow you to define different application property files under the same directory using the filename format, `application-{profile}.yml`.

For example:

- `application.yml`: The properties in non-specific files that always apply. Its properties are overridden by the properties defined in profile-specific files.
- `application-dev.yml`: Defines properties specific to the development environment.
- `application-prod.yml`: Defines properties specific to the production environment.

Individual profiles can then be enabled by setting the `spring.profiles.active` property.

See [Spring Boot: Profile-Specific Files](#) for more information and an example.

Configure Locations to Find Spring Property Files

By default, the connector detects any Spring property files as described in the [Spring Boot's default locations](#).

- If you want to add additional locations, add `--spring.config.additional-location=file:<custom-config-dir>` (This parameter is similar to the example command in [Quick Start: Running the connector via command line](#)).
- If you want to exclusively use the locations that you've defined and ignore Spring Boot's default locations, add `--spring.config.location=optional:classpath:/,optional:classpath:/config/,file:<custom-config-dir>`.

For more information about configuring locations to find Spring property files, see [Spring Boot documentation](#).



If you want configuration files for multiple, different connectors within the same `config` directory for use in different environments (such as development, production, etc.), we recommend that you use [Spring Boot Profiles](#) instead of child directories. For example:

- Set up your configuration like this:
 - `config/application-prod.yml`
 - `config/application-dev.yml`
- Do not do this:
 - `config/prod/application.yml`
 - `config/dev/application.yml`

Child directories are intended to be used for merging configuration from multiple sources of configuration properties. For more information and an example of when you might want to use multiple child directories to compose your application's configuration, see the [Spring Boot documentation](#).

Obtaining Build Information

Build information, including version, build date, time and description is enabled by default via [Spring Boot Actuator Info Endpoint](#). By default, every connector shares all information related to its `build` only.

Below is the structure of the output data:

```
{
  "build": {
    "version": "<connector version>",
    "artifact": "<connector artifact>",
    "name": "<connector name>",
    "time": "<connector build time>",
    "group": "<connector group>",
    "description": "<connector description>",
    "support": "<support information>"
  }
}
```

If you want to exclude build data from the output of the `info` endpoint, set `management.info.build.enabled` to `false`.

Alternatively, if you want to disable the info endpoint entirely, you can remove 'info' from the list of endpoints specified in `management.endpoints.web.exposure.include`.

Spring Configuration Options

This connector packages many libraries for you to customize functionality. Here are some

references to get started:

- [Spring Cloud Stream](#)
- [Spring Cloud Stream Binder for Solace PubSub+](#)
- [Spring Logging](#)
- [Spring Actuator Endpoints](#)
- [Spring Metrics](#)

Connector Configuration Options

The following table lists the configuration options. The following options in **Config Option** are prefixed with `solace.connector.:`


Config Option	Type	Default Value	Description
<code>management.leader-election.fail-over.max-attempts</code>	int Constraint: > 0	3	The maximum number of attempts to perform a fail-over.
<code>management.leader-election.fail-over.back-off-initial-interval</code>	long Constraint: > 0	1000	The initial interval (milliseconds) to back-off when retrying a fail-over.
<code>management.leader-election.fail-over.back-off-max-interval</code>	long Constraint: > 0	10000	The maximum interval (milliseconds) to back-off when retrying a fail-over.
<code>management.leader-election.fail-over.back-off-multiplier</code>	double Constraint: >= 1.0	2.0	The multiplier to apply to the back-off interval between each retry of a fail-over.
<code>management.leader-election.mode</code>	enum One of: <ul style="list-style-type: none"> • <code>standalone</code> • <code>active_active</code> • <code>active_standby</code> 	<code>standalone</code>	<p>The connector's leader election mode.</p> <p>standalone: A single instance of a connector without any leader election capabilities.</p> <p>active_active: A participant in a cluster of connector instances where all instances are active.</p> <p>active_standby: A participant in a cluster of connector instances where only one instance is active (i.e. the leader), and the others are standby.</p>
<code>management.queue</code>	string	null	The management queue name.


Config Option	Type	Default Value	Description
<code>management.session.*</code>	See Spring Boot Auto-Configuration for the Solace Java API		Defines the management session. This has the same interface as that used by <code>solace.java.*</code> . See Spring Boot Auto-Configuration for the Solace Java API for more info.
<code>security.enabled</code>	boolean	true	If <code>true</code> , security is enabled. Otherwise, anyone has access to the connector's endpoints.
<code>security.users[<index>].name</code>	string	null	The name of the user.
<code>security.users[<index>].password</code>	string	null	The password for the user.
<code>security.users[<index>].roles</code>	list<string> Valid values: • <code>admin</code>	empty list (i.e. read-only)	The list of roles that the specified user has. It has read-only access if no roles are returned.

Workflow Configuration Options

These configuration options are defined under the following prefixes:

- `solace.connector.workflows.<workflow-id>.`: If the options support per-workflow configuration and the default prefixes.
- `solace.connector.default.workflow.`: If the options support default workflow configuration.

Config Option	Type	Default Value	Description
<code>enabled</code>	boolean	false	If <code>true</code> , the workflow is enabled. <div>  Cannot be set at the default workflow level. </div>
<code>transform.enabled</code>	boolean	false	If <code>true</code> , message transformation is enabled for the workflow. Disables the legacy <code>transform-headers.*</code> and <code>transform-payload.*</code> options. See Message Transforms for more info.

Config Option	Type	Default Value	Description
<code>transform.source-payload.content-type</code>	string One of: <ul style="list-style-type: none"> <code>application/vnd.solace.micro-integration.unspecified</code> <code>application/json</code> 	<code>application/vnd.solace.micro-integration.unspecified</code>	The content type to interpret the source payload as. See Content Type Interpretation for more info.
<code>transform.target-payload.content-type</code>	string One of: <ul style="list-style-type: none"> <code>application/vnd.solace.micro-integration.unspecified</code> <code>application/json</code> 	<code>application/vnd.solace.micro-integration.unspecified</code>	The content type to interpret and serialize the target payload as. See Content Type Interpretation for more info.
<code>transform.expressions[<index>].transform</code>	string A SpEL expression		A SpEL expression at some <code><index></code> in the ordered list of expressions to transform the message. See Message Transform for more info.
<code>transform-headers.expressions</code>	Map<string, string> Key: A header name. Value: A SpEL string that accepts <code>headers</code> as parameters.	empty map	<div>  <div> Deprecated. Use Message Transforms (<code>transform.*</code>) instead. </div> </div> A mapping of header names to header value SpEL expressions. The SpEL context contains the <code>headers</code> parameter that can be used to read the input message's headers.
<code>acknowledgment.publish-async</code>	boolean	<code>false</code>	If <code>true</code> , publisher acknowledgment processing is done asynchronously. The workflow's consumer and producer bindings must support this mode, otherwise the publisher acknowledgments are processed synchronously regardless of this setting.

Config Option	Type	Default Value	Description
<code>acknowledgment.back-pressure-threshold</code>	int Constraint: ≥ 1	255	The maximum number of outstanding messages with unresolved acknowledgments. Message consumption is paused when the threshold is reached to allow for producer acknowledgments to catch up.
<code>acknowledgment.publish-timeout</code>	int Constraint: ≥ -1	600000	The maximum amount of time (in millisecond) to wait for asynchronous publisher acknowledgments before considering a message as failed. A value of -1 means to wait indefinitely for publisher acknowledgments.

SFTP Checkpoint Store Configuration options

The SFTP MI stores the information about the current progress of processing files in a checkpoint store backed by Solace LVQ (Last Value Queue).

In case of SFTP Consumer, the checkpoint store is used to store the information about the files that have been processed and the files that are currently being processed.

In case of SFTP Producer, the checkpoint store is used to store the information about the number of events written to the file and the file size, for file rotation purposes.

The following table lists the configuration options for the SFTP Checkpoint Store.

Config Option	Type	Valid Values	Default Value	Description
<code>camel-binder.checkpoint.lvqName</code>	String			<p>Required.</p> <p>The name of the Solace LVQ (spool size 0) to be used for checkpointing. The queue must exist on the same Solace PubSub+ broker/vpn as the target queue.</p> <p>Note: If the LVQ is deleted (administratively) or message from LVQ is deleted or consumed by another consumer, the SFTP MI will not be able to resume from the last checkpoint. Also, LVQ should not be shared by multiple instances of the SFTP MI.</p>
<code>camel-binder.checkpoint.autoProvisionLvq</code>	boolean	true, false	false	<p>Optional.</p> <p>Set to <code>true</code> to auto-provision the LVQ queue if it does not exist.</p>

SFTP Source Configuration Options

The SFTP source reads delimited, JSON, and XML files from the SFTP server and send them to the Solace PubSub+ event broker. The SFTP consumer reads the files, splits file data based on configuration into individual events, and sends them to the Solace PubSub+ event broker.

The configuration properties for the SFTP source are divided into two categories: common properties (SFTP connection or file filtering etc.) and file type specific properties.

The Spring Cloud Stream standard property for the SFTP Source are as follows.

Config Option	Type	Valid Values	Default Value	Description
<code>spring.cloud.stream.bindings.<input-x>.destination</code>	String	Format: <code>sftp://host:port/startDirectoryName/</code> e.g. <code>sfpt://localhost:22/orders-data/</code>		The source SFTP server URI and the directory path. Basically destination must include the start directory name.
<code>spring.cloud.stream.bindings.<input-x>.binder</code>	String	<code>camel</code>		The SFTP consumer is implemented using the Apache Camel SFTP component. This property must be set to <code>camel</code> .

The following table lists the SFTP source specific properties, common for all file types. For more information, see the [Apache Camel SFTP component](#) documentation.

All properties must be prefixed with `spring.cloud.stream.camel.bindings.<input-x>.consumer.endpoint.query-parameters`.

Example configuration:

```
spring:
  cloud:
    stream:
      camel:
        bindings:
          input-0:
            consumer:
              endpoint:
                query-parameters:
                  noop: true
                  delete: false
                  includeExt: csv
                  #other properties
```

Config Option	Type	Valid Values	Default Value	Description
<code>username</code>	String			Username to use for login.

Config Option	Type	Valid Values	Default Value	Description
password	String			Password to use for login, if using the password authentication. if not set, private key file has to be set.
privateKeyFile	String			Set the private key file so that the SFTP endpoint can do private key verification.
privateKeyPassphrase	String			Set the private key file passphrase so that the SFTP endpoint can do private key verification.
passiveMode	boolean		false	Sets passive mode connections. Default is active mode connections.
knownHostsFile	String			<p>Sets the known_hosts file, so that the SFTP endpoint can do host key verification.</p> <p>One can create a known_hosts file using the command:</p> <pre>ssh-keyscan -H <host> -P <port> >> known_hosts.</pre> <p>e.g. <code>ssh-keyscan -H myftpserver -p 22 >> known_hosts</code></p>
strictHostKeyChecking	String	yes,no	no	Sets whether to use strict host key checking.
autoCreateKnownHostsFile	boolean		false	If knownHostFile does not exist, then attempt to auto-create the path and file.

Config Option	Type	Valid Values	Default Value	Description
<code>ciphers</code>	String			Set a comma separated list of ciphers that will be used in order of preference. Possible cipher names are defined by JCraft JSCH. Some examples include: aes128-ctr,aes128-cbc,3des-ctr,3des-cbc,blowfish-cbc,aes192-cbc,aes256-cbc.
<code>keyExchangeProtocols</code>	String			Set a comma separated list of key exchange protocols that will be used in order of preference. Possible cipher names are defined by JCraft JSCH. Some examples include: diffie-hellman-group-exchange-sha1,diffie-hellman-group1-sha1,diffie-hellman-group14-sha1,diffie-hellman-group-exchange-sha256,ecdh-sha2-nistp256,ecdh-sha2-nistp384,ecdh-sha2-nistp521.
<code>publicKeyAcceptedAlgorithms</code>	String			Set a comma separated list of public key accepted algorithms. Some examples include: ssh-dss,ssh-rsa,ecdsa-sha2-nistp256,ecdsa-sha2-nistp384,ecdsa-sha2-nistp521.
<code>serverHostKeys</code>	String			Set a comma separated list of algorithms supported for the server host key. Some examples include: ssh-dss,ssh-rsa,ecdsa-sha2-nistp256,ecdsa-sha2-nistp384,ecdsa-sha2-nistp521.

Config Option	Type	Valid Values	Default Value	Description
connectTimeout	int	> 0	10000	Sets the connect timeout for waiting for a connection to be established.
soTimeout	int	> 0	300000	Sets the so timeout FTP and FTPS Is the SocketOptions.SO_TIMEOUT value in millis. Recommended option is to set this to 300000 so as not have a hanged connection.
maximumReconnectAttempts	int	>= 0		Specifies the maximum reconnect attempts when trying to connect to the remote FTP server. Use 0 to disable this behavior.
reconnectDelay	long	> 0	1000	Delay in millis to wait before performing a reconnect attempt.
serverAliveCountMax	int	> 0	1	Sets the number of keep-alive messages which may be sent without receiving any messages back from the server. If this threshold is reached while keep-alive messages are being sent, the connection will be disconnected.
serverAliveInterval	int	>= 0	0	Sets the interval (millis) to send a keep-alive message. If zero is specified, any keep-alive message must not be sent.
binary	boolean		false	Specifies the file transfer mode, BINARY or ASCII. Default is ASCII (false).
charset	String			Specifies the encoding of the file content.
noop	boolean		false	If true, the file is not moved or deleted in any way.

Config Option	Type	Valid Values	Default Value	Description
delete	boolean		false	If true, the file will be deleted after it is processed successfully.
move	String	<p>To move files into a .done subdirectory: : .done</p> <p>To move files into a .processed subdirectory and rename file: .processed/\${file:name}.done</p> <p>To move files into a .processed subdirectory and rename file with current timestamp: .processed/\${date:now:yyyMMdd}/\${file:name.noext}-\${date:now:yyyyMMddHHmmss}.done</p>		Expression (such as Simple Language) used to dynamically set the filename when moving it after processing.

Config Option	Type	Valid Values	Default Value	Description
<code>moveFailed</code>	String	<p>To move failed files into a <code>.error</code> subdirectory: <code>.error</code></p> <p>To move failed files into a <code>.error</code> subdirectory and rename file: <code>.error/\${file:name}.error</code></p> <p>To move failed files into a <code>.error</code> subdirectory and rename file with current timestamp: <code>.error/\${date:now:yyyyMMdd}/\${file:name.noext}-\${date:now:yyyyMMddHHmmss}.error</code></p>		Expression (such as Simple Language) used to dynamically set the filename when moving it after processing.
<code>sortBy</code>	String	<p>Sort by file name: <code>\${file:name}</code></p> <p>Sort by file modified time: <code>\${file:modified}</code></p>		Built-in sort by using the File Language. Supports nested sorts, so you can have a sort by file name and as a 2nd group sort by modified date.
<code>recursive</code>	boolean		<code>false</code>	If a directory, will look for files in all the subdirectories as well.
<code>maxDepth</code>	int	<code>> 0</code>	<code>2147483647</code>	The maximum depth to traverse when recursively processing a directory.

Config Option	Type	Valid Values	Default Value	Description
<code>minDepth</code>	<code>int</code>	<code>> 0</code>		The minimum depth to start processing when recursively processing a directory. Using <code>minDepth=1</code> means the base directory. Using <code>minDepth=2</code> means the first subdirectory.
<code>antExclude</code>	<code>String</code>			Ant style filter exclusion. If both <code>antInclude</code> and <code>antExclude</code> are used, <code>antExclude</code> takes precedence over <code>antInclude</code> . Multiple exclusions may be specified in comma-delimited format.
<code>antInclude</code>	<code>String</code>			Ant style filter inclusion. Multiple inclusions may be specified in comma-delimited format.
<code>antFilterCaseSensitive</code>	<code>boolean</code>		<code>true</code>	Sets case-sensitive flag on ant filter.
<code>exclude</code>	<code>String</code>			Is used to exclude files, if filename matches the regex pattern (matching is case in-sensitive).
<code>include</code>	<code>String</code>			Is used to include files, if filename matches the regex pattern (matching is case-insensitive).
<code>includeExt</code>	<code>String</code>			Is used to include files matching file extension name (case-insensitive). For example to include csv files, then use <code>includeExt=csv</code> . Multiple extensions can be separated by comma, for example to include csv and dat files, use <code>includeExt=csv,dat</code> .

Config Option	Type	Valid Values	Default Value	Description
<code>excludeExt</code>	String			Is used to exclude files matching file extension name (case-insensitive). For example to exclude bak files, then use <code>excludeExt=bak</code> . Multiple extensions can be separated by comma, for example to exclude bak and dat files, use <code>excludeExt=bak,dat</code> .
<code>filterFile, filterDirectory</code>	String			The source SFTP server URI and the directory path.
<code>delay</code>	long		5000	Milliseconds before the next poll.
<code>initialDelay</code>	long		1000	Milliseconds before the first poll starts.
<code>timeUnit</code>	String	Enum values: <code>NANOSECONDS, MICROSECONDS, MILLISECONDS, SECONDS, MINUTES, HOURS, DAYS</code>	MILLISECONDS	Time unit for <code>initialDelay</code> and <code>delay</code> options.
<code>stopOnException</code>	boolean	true, false	false	On an exception while processing a file, - If set to false, will log exception and continue processing next event in current file. - If set to true, stop processing the current file and continue to next file if any. if <code>moveFailed</code> is set, the file will be moved to the error directory.

Config Option	Type	Valid Values	Default Value	Description
<code>serverMessageLoggingLevel</code>	String	Enum values: TRACE, DEBUG, INFO, WARN, ERROR, OFF	DEBUG	The logging level used for various human intended log messages from the FTP server. This can be used during troubleshooting to raise the logging level and inspect the logs received from the FTP server.
<code>maxFileSize</code>	Long	≥ 0	2097152 (2MB in bytes)	The maximum size of the files to be consumed bytes. It helps in avoiding OOM (Out Of Memory) errors on small memory machines.

File Type Specific Properties

The following sections describe the properties specific to each file type. All properties are prefixed with `spring.cloud.stream.camel.bindings.<input-x>.consumer.endpoint.query-parameters`.

Config Option	Type	Valid Values	Default Value	Description
<code>fileType</code>	String	Enum values: delimited, json, xml	delimited	Required. The file processor to be used the files. Default is <code>delimited</code> .

Delimited File Config Properties

Note: The `spring.cloud.stream.camel.bindings.<input-x>.consumer.endpoint.query-parameters.fileType` should be set to `delimited`.

Config Option	Type	Valid Values	Default Value	Description
<code>eventDelimiter</code>	String		<code>\n</code> (newline)	The events in the files are separated by this character.

Config Option	Type	Valid Values	Default Value	Description
<code>ignoreHeaderLine</code>	boolean		false	<p>If the first line of the file should be ignored as a header.</p> <ul style="list-style-type: none"> - When set to <code>false</code>, the first line of the file is processed as a data line. - When set to <code>true</code>, the first line of the file is not processed as a data line. Also, if set to <code>true</code> and <code>paramNames</code> is not set, the first line of the file is used as the <code>paramNames</code> to convert the data to <code>json</code>.
<code>paramDelimiter</code>	String		, (comma)	The delimiter to use to separate the parameters in the header line or <code>paramNames</code> config property.
<code>paramNames</code>	String			<p>The list of parameter names separated by the <code>paramDelimiter</code>. The order of the names should match the order of the parameters in the file.</p> <p>If not set, and <code>ignoreHeaderLine</code> is true, first line of the file is used as the <code>paramNames</code>.</p>
<code>payloadFormat</code>	String	Enum Values: <code>json</code> , <code>text</code>	<code>json</code>	<p>The format of the payload.</p> <p>If <code>json</code>, the delimited data is converted to a JSON.</p> <p>If <code>text</code>, the delimited data is kept as is.</p>



if the specified `eventDelimiter` is not found within the file, the connector will process the entire file contents as one single event. For example, the configuration below uses `eventDelimiter: foofoofoofoo` (an unlikely delimiter) to intentionally keep the file intact without splitting. Setting `payloadFormat: text` preserves the raw content without JSON conversion, while `ignoreHeaderLine: false` ensures the first

line is treated as data rather than a header.

```
eventDelimiter: foofoofoofoo
ignoreHeaderLine: false
paramDelimiter: bar
payloadFormat: text
```

The max payload size is 30MB for Solace PubSub+ Enterprise and 10MB for Solace PubSub+ Standard. If the file size is larger than 30MB, the connector will throw an exception.

JSON File Config Properties

Note: The `spring.cloud.stream.camel.bindings.<input-x>.consumer.endpoint.query-parameters.fileType` should be set to `json`.

Config Option	Type	Valid Values	Default Value	Description
<code>jsonPath</code>	String	a valid JSON path expression	<code>\$</code> (root node)	<p>The JSON path expression to extract/split/filter the data from the JSON file.</p> <p>The default value <code>\$</code> means the entire JSON file content will be processed as a single event. For JSON file larger than 30MB, it would result in exception as max Solace message size is 30MB. It is recommended to split the JSON file content into multiple events by configuring the JSON path expression.</p>

XML File Config Properties

Note: The `spring.cloud.stream.camel.bindings.<input-x>.consumer.endpoint.query-parameters.fileType` should be set to `xml`.

Config Option	Type	Valid Values	Default Value	Description
xPath	String	a valid XPath path expression	/ (root node)	<p>The XPath expression to extract/split/filter the data from the XML file.</p> <p>The default value / means the entire XML file content will be processed as a single event. For XML file larger than 30MB, it would result in exception as max Solace message size is 30MB. It is recommended to split the XML file content into multiple events by configuring the XPath expression.</p> <p>e.g. For <code>employees.xml</code> file with following content: ` <pre><employees> <employee> <name>John</name> <age>30</age> </employee> <employee> <name>Smith</name> <age>40</age> </employee> </employees>`</pre> </p> <p>To extract employee nodes, the XPath expression would be <code>//employee</code>.</p>

SFTP Source Headers

The following files related headers are injected in each message. The example value of headers for the `sub1/sub2/employee.csv` file from the SFTP source from "sftp://host:port/rootDirectory/" are given in the table below.

Header Name	Type	Description
scst_file_name	String	<p>Only the file name (the name with no leading paths).</p> <p>e.g. <code>employee.csv</code></p>

Header Name	Type	Description
<code>scst_file_absolute_path</code>	String	<p>The absolute path to the file.</p> <p>e.g. <code>/tmp/.solace/input-0/sub1/sub2/employee.csv</code></p> <p>The file is downloaded to the local working directory <code>/tmp/.solace/input-0/</code>, so the absolute path will be <code>/tmp/.solace/input-0/sub1/sub2/employee.csv</code>.</p>
<code>scst_file_relative_path</code>	String	<p>The relative path of the file relative to the root directory.</p> <p>e.g. <code>sub1/sub2/employee.csv</code></p>
<code>scst_file_parent</code>	String	<p>The parent directory of the file.</p> <p>e.g. <code>sub1/sub2/</code></p>
<code>scst_file_length</code>	String	<p>A long (converted to String) value containing the file size.</p> <p>e.g. <code>2538657</code></p>
<code>scst_file_last_modified</code>	String	<p>A long (converted to String) value, representing the last modified time in milliseconds.</p> <p>e.g. <code>1728345600000</code> (i.e. 2024-10-08, 12:00 a.m.)</p>
<code>scst_event_offset</code>	String	<p>The offset of the event in the file.</p> <p>e.g. <code>5</code>, for the 5th event in the file and so on. For the file with the headers the offset starts at 1 for the first data line.</p>

SFTP Target Configuration Options

The SFTP target/publisher consumes events from the Solace PubSub+ event broker and writes them to files on the SFTP Server. The SFTP target only writes the payload of the event to the file. If one wants to write the headers to the file, one can use the transformation feature to transform the event to include the headers in the payload.

The SFTP publisher uses the Apache Camel SFTP component to write the events to the SFTP server.

The configuration properties for the SFTP target are documented in the following sections.

The Spring Cloud Stream standard properties for the SFTP target are as follows.

Config Option	Type	Valid Values	Default Value	Description
<code>spring.cloud.stream.bindings.<input-x>.destination</code>	String	Format: <code>sftp://host:port/startDirectoryName/</code> e.g. <code>sfpt://localhost:22/orders-data/</code>		The target SFTP server URI and the directory path. Basically destination must include the start directory name.
<code>spring.cloud.stream.bindings.<input-x>.binder</code>	String	<code>camel</code>		The SFTP publisher is implemented using the Apache Camel SFTP component. This property must be set to <code>camel</code> .

The following table lists the SFTP target specific properties. For more information, see the [Apache Camel SFTP component](#) documentation.

All properties must be prefixed with `spring.cloud.stream.camel.bindings.<output-x>.producer.endpoint.query-parameters`.

Example configuration:

```
spring:
  cloud:
    stream:
      camel:
        bindings:
          output-0:
            producer:
              endpoint:
                query-parameters:
                  username: test
                  password: test
                  fileName: output.csv
                  #other properties
```

Config Option	Type	Valid Values	Default Value	Description
<code>username</code>	String			Username to use for login.

Config Option	Type	Valid Values	Default Value	Description
password	String			Password to use for login, if using the password authentication. if not set, private key file has to be set.
privateKeyFile	String			Set the private key file so that the SFTP endpoint can do private key verification.
privateKeyPassphrase	String			Set the private key file passphrase so that the SFTP endpoint can do private key verification.
passiveMode	boolean		false	Sets passive mode connections. Default is active mode connections.
knownHostsFile	String			<p>Sets the known_hosts file, so that the SFTP endpoint can do host key verification.</p> <p>One can create a known_hosts file using the command:</p> <pre>ssh-keyscan -H <host> -P <port> >> known_hosts.</pre> <p>e.g. <code>ssh-keyscan -H myftpsrvr -p 22 >> known_hosts</code></p>
strictHostKeyChecking	String	yes,no	no	Sets whether to use strict host key checking.
autoCreateKnownHostsFile	boolean		false	If knownHostFile does not exist, then attempt to auto-create the path and file.

Config Option	Type	Valid Values	Default Value	Description
<code>ciphers</code>	String			Set a comma separated list of ciphers that will be used in order of preference. Possible cipher names are defined by JCraft JSCH. Some examples include: aes128-ctr,aes128-cbc,3des-ctr,3des-cbc,blowfish-cbc,aes192-cbc,aes256-cbc.
<code>keyExchangeProtocols</code>	String			Set a comma separated list of key exchange protocols that will be used in order of preference. Possible cipher names are defined by JCraft JSCH. Some examples include: diffie-hellman-group-exchange-sha1,diffie-hellman-group1-sha1,diffie-hellman-group14-sha1,diffie-hellman-group-exchange-sha256,ecdh-sha2-nistp256,ecdh-sha2-nistp384,ecdh-sha2-nistp521.
<code>publicKeyAcceptedAlgorithms</code>	String			Set a comma separated list of public key accepted algorithms. Some examples include: ssh-dss,ssh-rsa,ecdsa-sha2-nistp256,ecdsa-sha2-nistp384,ecdsa-sha2-nistp521.
<code>serverHostKeys</code>	String			Set a comma separated list of algorithms supported for the server host key. Some examples include: ssh-dss,ssh-rsa,ecdsa-sha2-nistp256,ecdsa-sha2-nistp384,ecdsa-sha2-nistp521.

Config Option	Type	Valid Values	Default Value	Description
<code>connectTimeout</code>	<code>int</code>	<code>> 0</code>	<code>10000</code>	Sets the connect timeout for waiting for a connection to be established.
<code>soTimeout</code>	<code>int</code>	<code>> 0</code>	<code>300000</code>	Sets the so timeout FTP and FTPS Is the <code>SocketOptions.SO_TIMEOUT</code> value in millis. Recommended option is to set this to 300000 so as not have a hanged connection.
<code>maximumReconnectAttempts</code>	<code>int</code>	<code>>= 0</code>		Specifies the maximum reconnect attempts when trying to connect to the remote FTP server. Use 0 to disable this behavior.
<code>reconnectDelay</code>	<code>long</code>	<code>> 0</code>	<code>1000</code>	Delay in millis to wait before performing a reconnect attempt.
<code>serverAliveCountMax</code>	<code>int</code>	<code>> 0</code>	<code>1</code>	Sets the number of keep-alive messages which may be sent without receiving any messages back from the server. If this threshold is reached while keep-alive messages are being sent, the connection will be disconnected.
<code>serverAliveInterval</code>	<code>int</code>	<code>>= 0</code>	<code>0</code>	Sets the interval (millis) to send a keep-alive message. If zero is specified, any keep-alive message must not be sent.
<code>binary</code>	<code>boolean</code>		<code>false</code>	Specifies the file transfer mode, BINARY or ASCII. Default is ASCII (false).
<code>charset</code>	<code>String</code>			Specifies the encoding of the file content.

Config Option	Type	Valid Values	Default Value	Description
<code>fileExist</code>	String	Override, Append, Fail	Override	<p>What to do if a file already exists with the same name.</p> <p>Override: replaces the existing file.</p> <p>Append: adds content to the existing file.</p> <p>Fail: throws an Exception, indicating that there is already an existing file.</p>
<code>fileName</code>	String	Any valid file name		<p>The filename to write to. Can include the leading file path. e.g. <code>sub1/sub2/employee.csv</code></p> <p>If not set, randomly generated UUID is used for the filename, in which case each event will be written to a unique file.</p> <p>The fileName could be Camel Simple Language expression. e.g. <code>fileName: "backup/\${date:now:yyyyMMdd}/test.txt"</code></p>
<code>chmod</code>	String	Unix chmod value. e.g. 740		Allows you to set chmod on the file.
<code>chmodDirectory</code>	String	Unix chmod value. e.g. 740		Allows you to set chmod during directory/path creation.
<code>preventDuplicates</code>	boolean	true, false	false	If set to true duplicate events will be ignored. The duplicate events are identified based on the value of the header configured as <code>uniqueIdentifierHeader</code> .

Config Option	Type	Valid Values	Default Value	Description
<code>preventDuplicatesOnlyForRedeliveredEvents</code>	boolean	true, false	true	Used only when <code>preventDuplicates</code> set to true. If set to true duplicate key will be checked for Redelivered events only. If false, will be checked for all events.
<code>uniqueIdentifierHeader</code>	String	Any valid header name	<code>eventId</code>	The user property or header name to be used for uniquely identifying the events. Used only when <code>preventDuplicates</code> set to true.
<code>cacheSize</code>	int	> 0	260	The size of the in-memory cache used to store the unique identifiers of the events. Used only when <code>preventDuplicates</code> set to true.
<code>maxEventsPerFile</code>	Long	>= 0	0 (unlimited)	The maximum number of events to be written to a file. If the value is 0, then there is no limit.
<code>maxFileSize</code>	Long	>= 0	0 (unlimited)	The maximum size of the file in bytes. If the value is 0, then there is no limit.
<code>prependToEvent</code>	String	Any valid string	Empty string	The string to be prepended to the event.
<code>appendToEvent</code>	String	Any valid string	\n (newline)	The string to be appended to the event.
<code>prependToFile</code>	String	Any valid string	Empty string	The string to be prepended to the file. This is done only once when the file is created. e.g. header line.
<code>prependToFirstEvent</code>	String	Any valid string	Empty string	The string to be prepended to the first event.

Config Option	Type	Valid Values	Default Value	Description
<code>serverMessageLoggingLevel</code>	String	Enum values: TRACE, DEBUG, INFO, WARN, ERROR, OFF	DEBUG	The logging level used for various human intended log messages from the FTP server. This can be used during troubleshooting to raise the logging level and inspect the logs received from the FTP server.

SFTP Target Headers

User can set the following headers through transformation feature or on the source Solace message.

Header Name	Type	Description
<code>scst_targetDestination</code>	String	<p>Specifies the path/name of the file. This header helps dynamically override the static file path/name configured through <code>fileName</code> property.</p> <p>e.g. <code>scst_targetDestination: dynamically/decided/path/to/employee-123.csv</code></p> <p>The <code>scst_targetDestination</code> could be Camel Simple Language expression. e.g. <code>scst_targetDestination: "backup/\${date:now:yyyyMMdd}/dynamic.txt"</code></p>

Logging

Configuring Logback

If you require Logback configuration beyond what is already available through Spring config properties, then you can decide to run your connector with a `logback-spring.xml` file. For information about using the logback, see:

- [logging.config](#) spring property.
- [Logging](#)
- [Configure Logback for Logging](#) sections in the Spring documentation.

The main difference compared to [what Spring Boot provides](#), is that this connector provides its own alternative logback configuration files that can be [included](#) into your `logback-spring.xml` file.

These new files can be included and are found at `com/solace/connector/core/logging/logback/`:

`defaults.xml`

Provides conversion rules, pattern properties and common logger configurations.



Always include `defaults.xml`

We recommend that you always include this file in your `logback-spring.xml` file as it includes a `%sanitize` `<conversionRule>` that's applied to the default `CONSOLE_LOG_PATTERN` and `FILE_LOG_PATTERN`.

This conversion rule does some filtering to obfuscate potentially sensitive data from the logs.

`console-appender.xml`

Adds a `ConsoleAppender` using the `CONSOLE_LOG_PATTERN`.

`file-appender.xml`

Adds a `RollingFileAppender` using the `FILE_LOG_PATTERN` and `ROLLING_FILE_NAME_PATTERN` with appropriate settings.

Aside from these new include-able files, this connector still supports all the same logging options that regular Spring Boot does.

For more information and examples, see the [Configure Logback for Logging](#) section in the Spring documentation.

License

This project is licensed under the Solace Community License, Version 1.0. - See the [LICENSE](#) file for details.

Support

Support is offered best effort via our [Solace Developer Community](#).

Premium support options are available, please [Contact Solace](#).