Solace PubSub+ Connector for Salesforce
User Guide

Solace Corporation

Version 3.0.0

solace.

Table of Contents

Preface

The Solace Salesforce connector provides the ability to integrate Salesforce platform events to event brokers and the Solace Event Mesh using the Salesforce's Pub Sub API.

Functionality

Following are some of the high level features of Salesforce Connector. Detailed configuration and descriptions can be found in User Guide document.

1. Provides a single interface for publishing and subscribing to platform events.

2. Publish and subscribe to custom platform events to send and receive real-time notifications in another system for business process integration. Some examples of business process integrations are sending real-time notifications for maintenance service or product orders.

3. Connector can subscribe to Salesforce's real-time event monitoring events, change data capture events or custom platform events

solace.

Getting Started

Assuming you're using the default `application.yml` within this package, following one of the below quick start guides will result in a connector that will connect to the PubSub+ broker and SFDC using default credentials, with 2 workflows enabled, workflow 0 and workflow 1. Where:

- Workflow 0 is consuming messages from the Solace PubSub+ queue, `Solace/Queue/0`, and publishing them to the SFDC producer destination, `producer-destination`.

- Workflow 1 is consuming messages from the SFDC consumer destination, `consumer-destination`, and publishing 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 will not provision queues which do not exist.

Prerequisites

- Solace PubSub+ Event Broker
- SFDC

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-salesforce-3.0.0.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 Salesforce

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.
-p, --profile	empty, no profile is used	The profile to be used with the connector's configuration. The configuration file named 'application-<profile>.yml' is used. If this option is not specified, no profile is used.

solace.

Parameter	Default Value	Description
`-c, --config`	`./ 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.
`-H, --host`	`127.0.0.1`	Specifies the host where the connector runs.
`-P, --port`	`8090`	Specifies the port where connector runs.
`-mp, --mgmt_port`	`9009`	Specifies the management port for back calls of current connector from PubSub+ Connector Manager. This parameter is ignored if the `-cm` parameter is not provided.
`-j, --jar`	`pubsubplus-connector-salesforce-3.0.0.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.
`-cm, --manager`	`application`	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 `-mp` or `--mgmt_port`, `-H` or `--host`, and `-cmh` with the `-cmp` 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.
`-cmh, --cm_host`	`127.0.0.1`	Specifies the host where Connector Manager is running. This parameter is ignored if the `-cm` parameter is not provided.

solace.

Parameter	Default Value	Description
`-cmp, --cm_port`	`9500`	Specifies the port where Connector Manager is running. This parameter is ignored if `-cm` parameter is not provided.
`-o, --options`	`no default values`	Specifies the JVM options used on when the connector starts. For example, `-Xms64M -Xmx1G`.
`-tls`	`N/A`	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.
`-s, --show`	`N/A`	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.
`-b, --background`	`N/A`	Runs the connector in the background. No logs are shown and the connector continues running in detached mode.
`-h, --help`	`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 SFDC 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.

solace.

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]`:

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

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

> ℹ️ The connector only supports workflows in the directions of:
>
> - `solace → SFDC`
> - `SFDC → 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+

solace.

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:

```yaml
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

To connect to multiple systems of a same type, use the multiple binder syntax.

For example:

```yaml
spring:
  cloud:
    stream:
      binders:

        # 1st solace binder in this example
        solace1:
          type: solace
          environment:
            solace:
              java:
                host: tcp://localhost:55555

        # 2nd solace binder in this example
```

solace.

```yaml
      solace2:
        type: solace
        environment:
          solace:
            java:
              host: tcp://other-host:55555

      # The only salesforce binder
      salesforce1:
        type: salesforce
        # Add `environment` property map here if you need to customize this binder.
        # But for this example, we'll assume that defaults are used.

      # Required for internal use
      undefined:
        type: undefined
    bindings:
      input-0:
        destination: <input-destination>
        binder: salesforce1
      output-0:
        destination: <output-destination>
        binder: solace1 # Reference 1st solace binder
      input-1:
        destination: <input-destination>
        binder: salesforce1
      output-1:
        destination: <output-destination>
        binder: solace2 # Reference 2nd solace binder
```

The configuration above defines two binders of type `solace` and one binder of type `salesforce`, which are then referenced within bindings.

Each binder above is configured independently under `spring.cloud.stream.binders.<binder-name>.environment`.

> ❗ • When connecting to multiple systems, all binder configuration must be specified using the multiple binder syntax for all binders. For example, under the `spring.cloud.stream.binders.<binder-name>.environment`.
>
> • Do not use single-binder configuration (for example, `solace.java.*` at the root of your `application.yml`) while using the multiple binder syntax.

Salesforce Connection Details

The Spring Cloud Stream Binder for AWS Kinesis uses the following configuration to configure its session

Authentication Flows

1. client_credentials - Oauth2 using Client credentials (clientId and clientSecret)

2. password - Oaith2 using Password (clientId, clientSecret, Username, password) to uniquely identify user in oauth2

3. login_using_soap - Soap Login using Username and Password

4. access_token - Session login using tenantId and accessToken (mostly used for testing)

5. jwt – requires JWT Assertion

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

```
<<<<<<< Updated upstream
salesforce:
  pubsubHost: api.pubsub.salesforce.com
  pubsubPort: 7443
  loginUrl: https://SALESFORCEHOST
  username: XXXXXXX
  password: XXXXXXX
  tenantId: null
  accessToken: null
```

In case of Docker deployment the json file can be placed in config folder and the config path relevant to docker container is used to point the file location.

spring.cloud.stream.salesforce.<binding>.(consumer/producer): pubsubHost: api.pubsub.salesforce.com pubsubPort: 7443 loginUrl: https://SALESFORCEHOST username: Username to login to Salesforce password: Password+accessToken combination tenantId: TenantId incase of Oauth accessToken: accessToken incase of Oauth authentication: loginUrl: https://<identifier>.salesforce.com authFlow: (client_credentials, login_using_soap, jwt, password, access_token) clientId: Client Id of the connecting app clientSecret: Client Secret of the connecting app username: username of the connecting user password: password of the user securityToken: security token used in combination with password jwtAssertion: jwt assertion if authFlow set to jwt tenantId: Salesforce organisation id accessToken: access token used with tenantId for direct authentication. Usually configured for testing environments

```
In case of Docker deployment the json file can be placed in config folder and the
config path relevant to docker container is used to point the file location.

=== Consumer specific Properties

[source,yaml]
```

spring.cloud.stream.salesforce.<binding>.consumer: lvq: Solace LVQ name. To fetch the last successful ReplayId from the last published message outputFormatJSON: Boolean - true/false. Output format will be json text if set to true, else binary avro. Default value true batchSize:

solace.

Integer. Default 5. Number of messages in a batch until next fetch request to the grpc server is done

```
replayPreset: String. Replay Preset Value (Values  - EARLIEST, CUSTOM, LATEST).
For more details, Refer to link
https://developer.salesforce.com/docs/platform/pub-sub-
api/references/methods/subscribe-rpc.html#replaying-an-event-stream
replayId: String. Custom Replay Id if replayPreset is set to CUSTOM. For example
replayId shoulbe be in byte array format as string in the configuration
"[0,0,0,0,5,-118,-36,-61,0,0]"
```

```
=== Producer specific Properties


[source,yaml]
```

spring.cloud.stream.salesforce.<binding>.producer: inputFormatJSON: Boolean - true/false. Input format will be json text if set to true, else binary avro. Default value false asyncPublish: true, false batchPublish: enabled: true, false batchSize: Integer. Batch Size in case enabled. Default value 10 batchTimeout: Time in milliseconds to wait for a new message to be added to the current batch. or Idle time to publish batch to Salesforce. Default 2000

```
If both batchPublish and asyncPublish are disabled, the messages will be published
synchronously.

In case both batchPublish and asyncPublish are enabled, Batch Publish is
prioritised.


:leveloffset!:

:leveloffset: +1




= User-configured Header Transforms

.Deprecated
[WARNING]
====
This feature is deprecated and will be removed in a future release. Use
<<_message_transform, Message Transforms>> instead.
```

Generic Migration Rules:

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

. 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:
[source,yaml]

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

New Configuration:
[source,yaml]

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:

[source,properties]

<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-

solace.

headers.expressions.<header>=<expression>

> **Example 1:** To create a new header, `new_header`, for workflow `0` that is
> derived from the headers `foo` & `bar`:
>
> [source,properties]

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`:
>
> [source,properties]

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

> For more information about Spring Expression Language (SpEL) expressions, see
> https://docs.spring.io/spring-
> framework/docs/current/reference/html/core.html#expressions[Spring Expression
> Language (SpEL)].
>
> :leveloffset: 2
>
> :leveloffset: +1
>
>
>
>
> = User-configured Payload Transforms
>
> .Deprecated
> [WARNING]
> ====
> This feature is deprecated and will be removed in a future release. Use
> <<_message_transform, Message Transforms>> instead.
> ====
>
> Message payloads going through a workflow can be transformed using a Spring
> Expression Language (SpEL) expression as follows:
>
> [source,properties]

solace.

<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.

[NOTE]
====
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

https://docs.spring.io/spring-
framework/docs/current/reference/html/core.html#expressions-ref-
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

|`boolean isPayloadBytes(Object obj)`
|Returns whether the object `obj` is an instance of `byte[]` or not.

Sample usage of this function within a SpEL expression: `"#isPayloadBytes(payload)
? true : false"`
|===

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

[source,properties]
```

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:
```

```
[source,properties]
```

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
https://docs.spring.io/spring-
framework/docs/current/reference/html/core.html#expressions[Spring Expression
Language (SpEL)].

:leveloffset: 2

== Message Headers

Solace and {vendor-binder-id} headers can be created or manipulated using the
<<User-configured Header Transforms>> feature described above.

:leveloffset: +2




= Solace Headers

Solace headers exposed to the connector are documented in the
https://github.com/SolaceDev/solace-spring-cloud/tree/master/solace-spring-cloud-
starters/solace-spring-cloud-stream-starter#solace-message-headers[Spring Cloud
Stream Binder for Solace PubSub+] documentation.

:leveloffset: 2
:leveloffset: +2




= Reserved Message Headers

The following are reserved header spaces:

* `solace_`
```

```
* `scst_`
* Any headers defined by the core Spring messaging framework. See
https://docs.spring.io/spring-integration/reference/html/message.html#message-
headers[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.

:leveloffset: 2

== Management and Monitoring Connector

:leveloffset: +2




= Monitoring Connector's States

The connector provides an ability to monitor its internal states through exposed
endpoints provided by https://docs.spring.io/spring-
boot/docs/current/reference/htmlsingle/#actuator[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:
[source,yaml]
```

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.

[NOTE]
====
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:
[source,bash]
```

GET: /actuator/

```
The following shows an example of the data shared with the configuration above:
[source,json]
```

{ "_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 }
} }

```
:leveloffset: 2

== Health

:leveloffset: +2




The connector reports its health status using the https://docs.spring.io/spring-
cloud-stream/docs/3.2.2/reference/html/spring-cloud-
stream.html#_health_indicator[Spring Boot Actuator health endpoint].
```

solace.

```
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 https://docs.spring.io/spring-
boot/docs/current/reference/html/actuator.html#actuator.endpoints.health[Spring
Boot documentation].

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

[source,yaml]
```

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`.

:leveloffset: 2
:leveloffset: +2




= 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:

[source,json]
```

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

```
[cols="1m,3", options="header"]
|===
| 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.
|===
```

```
.Workflow Health Resolution Diagram
[plantuml, target="workflow-health-resolution-diagram"]
```

start if (workflow state?) is (**paused**) then (not **paused**) elseif (fail-over in progress?) then (yes) else (no) switch (leader-election state?) case (**active**) if (workflow state?) is (**unknown**) then #pink:health=**DOWN**; stop else (not **unknown**) endif case (**standby**) if (workflow state?) is (not **stopped**) then #pink:health=**DOWN**; stop else (**stopped**) endif endswitch endif

if (workflow's overall binders'/bindings' states?) is (**UP** or **RECONNECTING** or **UNKNOWN**) then else #pink:health=**DOWN**; stop endif #palegreen:health=**UP**; stop

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

[source,properties]
```

management.health.workflows.enabled=false

```
:leveloffset: 2
:leveloffset: +2




= Solace Binder Health

For details, see the https://github.com/SolaceDev/solace-spring-
cloud/tree/master/solace-spring-cloud-starters/solace-spring-cloud-stream-
starter#solace-binder-health-indicator[Solace binder] documentation.

:leveloffset: 2

:leveloffset: +1
```

```
= Leader Election

The connector has three leader election modes for redundancy:

[cols="1,3", options="header"]
|===
| Leader Election Mode
| Description

| Standalone (Default)
| A single instance of a connector without any leader election capabilities.

| Active-Active
| A participant in a cluster of connector instances where all instances are
active.

| 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:

[source,properties]
```

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

```
== Leader Election Modes: Standalone / Active-Active

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:

[source,properties]
```

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
https://docs.spring.io/spring-
boot/docs/current/reference/html/actuator.html#actuator.endpoints[Spring
Actuator].

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

[cols="2m,1,4a", options="header"]
|===
|Endpoint
|Operation
|Payloads

|/leaderelection
|Read +
(HTTP `GET`)
|**Request:** None.

**Response:**
[source,json]
```

```json
{ "mode": { "type": "(standalone \| active_active \| <1> active_standby)", "state": "(active \|
standby)", <2> "source": { <3> "queue": "<management-queue-name>", "host": "<management-
```

solace.

host>", "msgVpn": "<management-vpn>" } } }

```
<1> Mandatory parameter in output
<2> Mandatory parameter in output
<3> Optional section. Appears only when `type` is set to `active_standby`.
|===

:leveloffset: 2

:leveloffset: +1




= Workflow Management

== Workflow Management Endpoint

A custom `workflows` management endpoint using https://docs.spring.io/spring-
boot/docs/current/reference/html/actuator.html#actuator.endpoints[Spring Actuator]
is provided to manage workflows.

To enable the `workflows` management endpoint:

[source,yaml]
```

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

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

[cols="2m,1,4a", options="header"]
|===
| Endpoint
| Operation
| Payloads

| /workflows
| Read +
(HTTP `GET`)
| **Request:** None.

**Response:** +
Same payload as the `/workflows/\{workflowId\}` read operation, but as a list of
all workflows.
```

solace.

```
| /workflows/\{workflowId\}
| Read +
(HTTP `GET`)
| **Request:** None.

**Response:**

[source,json]
```

{ "id": "<workflowId>", "enabled": (true\|false), "state": "(running\|stopped\|paused\|unknown)", "inputBindings": ["<input-binding>"], "outputBindings": ["<output-binding>"] }

```
| /workflows/\{workflowId\}
| Write +
(HTTP `POST`)
| **Request:**

[source,json]
```

{ "state": "STARTED\|STOPPED\|PAUSED\|RESUMED" }

```
**Response:** None.
|===

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

[NOTE]
====
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
https://docs.spring.io/spring-cloud-stream/docs/current/reference/html/spring-
```

```
cloud-stream.html#_actuator[`bindings` management endpoint]) as follows:

[cols="1m,2", options="header"]
|===
| 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.
|===

[NOTE]
====
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 vise-
versa), the workflow state would be `running`.
====



For more information about binding states, see https://docs.spring.io/spring-
cloud-stream/docs/current/reference/html/spring-cloud-stream.html#_actuator[Spring
Cloud Stream: Binding visualization and control].

:leveloffset: 2

== Metrics

:leveloffset: +2




This connector uses https://docs.spring.io/spring-
boot/docs/current/reference/html/actuator.html#actuator.metrics[Spring Boot
Metrics] that leverages Micrometer to manage its metrics.
```

```
:leveloffset: 2
:leveloffset: +2




= Connector Meters

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

[cols="2m,1,2m,1,3a", options="header"]
|===
| Name
| Type
| Tags
| Description
| Notes

| solace.connector.{zwsp}process
| `Timer`
| type: channel


name: <bindingName>


result: (success\|failure)


exception: (none\|exception simple class name)
| The processing time.
| This meter is a rename of https://docs.spring.io/spring-
integration/docs/current/reference/html/system-
management.html#overview[`spring.integration.send`] whose `name` tag matches a
binding name.

| solace.connector.{zwsp}error.process
| `Timer`
| type: channel


name: <bindingNames>


result: (success\|failure)


exception: (none\|exception simple class name)
| The error processing time.
| This meter is a rename of https://docs.spring.io/spring-
integration/docs/current/reference/html/system-
```

management.html#overview[`spring.integration.send`] whose `name` tag matches an input binding's error channel name (`<destination>.<group>.errors`).

Meters might be merged under the same `name` tag (delimited by `\|`) if multiple bindings have the same error channel name (for example, bindings can have a matching `destination`, `group`, or both). **NOTE: Setting a binding's `group` is not supported.**

| solace.connector.{zwsp}message.size.payload
| `DistributionSummary`

Base Units: `bytes`
| name: <bindingName>
| The message payload size.
|

| solace.connector.{zwsp}message.size.total
| `DistributionSummary`

Base Units: `bytes`
| name: <bindingName>
| The total message size.
|

| solace.connector.{zwsp}publish.ack
| `Counter`

Base Units: `acknowledgments`
| name: <bindingName>

result: (success\|failure)

exception: (none\|exception simple class name)
| The publish acknowledgment count.
|

| solace.connector.{zwsp}transform.{zwsp}expressions.count
| `Gauge`

Base Units: `expressions`
| workflow.id: <workflowId>
| Transformation expressions count
|

| solace.connector.{zwsp}transform.{zwsp}time
| `Timer`
| workflow.id: <workflowId>

result: (success\|failure)
| Transformation execution time
| This is the aggregate time for all expressions used to transform a single

solace.

```
message.
|===
```

```
[IMPORTANT]
====
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.
====
```

```
:leveloffset: 2
:leveloffset: +2
```

```
= Add a Monitoring System
```

```
By default, this connector includes the following monitoring systems:
```

```
* https://docs.spring.io/spring-
boot/docs/current/reference/html/actuator.html#actuator.metrics.export.datadog[Dat
adog]
* https://docs.spring.io/spring-
boot/docs/current/reference/html/actuator.html#actuator.metrics.export.dynatrace[D
ynatrace]
* https://docs.spring.io/spring-
boot/docs/current/reference/html/actuator.html#actuator.metrics.export.influx[Infl
ux]
* https://docs.spring.io/spring-
boot/docs/current/reference/html/actuator.html#actuator.metrics.export.jmx[JMX]
* https://docs.spring.io/spring-
boot/docs/current/reference/html/actuator.html#actuator.metrics.export.otlp[OpenTe
lemetry (OTLP)]
* https://docs.spring.io/spring-
boot/docs/current/reference/html/actuator.html#actuator.metrics.export.statsd[Stat
sD]
```

```
To add additional monitoring systems, add the system's `micrometer-registry-
<system>` JAR file and its dependency JAR files to <<Adding External Libraries,
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`.

:leveloffset: 2

== Security

:leveloffset: +2
```

```
= 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 https://docs.spring.io/spring-
boot/docs/current/reference/html/actuator.html#actuator.endpoints[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:
[source,yaml]
```

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`.
```

solace.

IMPORTANT: 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:

[source, shell]

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.

[NOTE, id="note-users-list"]
====
`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 https://docs.spring.io/spring-boot/docs/current/reference/html/features.html#features.external-config.typesafe-configuration-properties.merging-complex-types[Spring Boot - Merging Complex Types].
====

== TLS

TLS is disabled by default.

To configure TLS, see https://docs.spring.io/spring-boot/docs/current/reference/html/howto.html#howto.webserver.configure-ssl[Spring Boot - Configure SSL] and https://www.baeldung.com/spring-tls-setup[TLS Setup in Spring].

:leveloffset: 2

:leveloffset: +1

solace.

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 <<Adding External Libraries, the connector's classpath>>.

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

[TIP]
====
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.
====

:leveloffset: 2

:leveloffset: +1

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 https://docs.spring.io/spring-boot/docs/current/reference/html/executable-jar.html#appendix.executable-jar.property-launcher[Spring Boot - PropertiesLauncher Features] for more info.

To add libraries to the connector's container image, see {connector-container-

url}[the connector's container documentation].

:leveloffset: 2

== Configuration

:leveloffset: +2

= Providing Configuration

For information about about how the connector detects configuration properties,
see https://docs.spring.io/spring-
boot/docs/current/reference/html/features.html#features.external-config[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 https://docs.spring.io/spring-
boot/docs/current/reference/html/features.html#features.external-config.typesafe-
configuration-properties.relaxed-binding.environment-variables[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 https://docs.spring.io/spring-
boot/docs/current/reference/html/features.html#features.external-
config.files.profile-specific[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
https://docs.spring.io/spring-
boot/docs/current/reference/html/features.html#features.external-
config.files[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:<cust
om-config-dir>`.

For more information about configuring locations to find Spring property files,
see https://docs.spring.io/spring-
boot/docs/current/reference/html/features.html#features.external-
config.files[Spring Boot documentation].

:leveloffset: +1

[TIP]
====
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 https://docs.spring.io/spring-
boot/docs/current/reference/html/features.html#features.external-
config.files.profile-specific[Spring Boot Profiles] instead of child directories.
For example:

* Set up your configuration like this:
** `{spring-config-dir}/application-prod.yml`
** `{spring-config-dir}/application-dev.yml`
* Do not do this:
** `{spring-config-dir}/prod/application.yml`
** `{spring-config-dir}/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 https://docs.spring.io/spring-boot/docs/current/reference/html/features.html#features.external-config.files.wildcard-locations[Spring Boot documentation].
====

:leveloffset: 2

== Obtaining Build Information

Build information, including version, build date, time and description is enabled by default via https://docs.spring.io/spring-boot/docs/current/actuator-api/htmlsingle/#info[Spring Boot Actuator Info Endpoint]. By default, every connector shares all information related to its https://docs.spring.io/spring-boot/docs/current/actuator-api/htmlsingle/#info.retrieving.response-structure.build[build] only.

Below is the structure of the output data:
```json
{
  "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`.

:leveloffset: 2
:leveloffset: +2

```
= Spring Configuration Options

This connector packages many libraries for you to customize functionality. Here
are some references to get started:

* https://docs.spring.io/spring-cloud-stream/docs/current/reference/html/spring-
cloud-stream.html#_configuration_options[Spring Cloud Stream]
* https://github.com/SolaceProducts/solace-spring-cloud/tree/master/solace-spring-
cloud-starters/solace-spring-cloud-stream-starter#configuration-options[Spring
Cloud Stream Binder for Solace PubSub+]
* https://docs.spring.io/spring-
boot/docs/current/reference/html/features.html#features.logging[Spring Logging]
* https://docs.spring.io/spring-
boot/docs/current/reference/html/actuator.html#actuator.endpoints[Spring Actuator
Endpoints]
* https://docs.spring.io/spring-
boot/docs/current/reference/html/actuator.html#actuator.metrics[Spring Metrics]

:leveloffset: 2
:leveloffset: +2




= Connector Configuration Options

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

[cols="2m,2a,1m,4", options="header"]
|===
| Config Option
| Type
| Default Value
| Description

| management.{zwsp}leader-election.{zwsp}fail-over.max-attempts
| `int`

**Constraint:** `> 0`
| 3
| The maximum number of attempts to perform a fail-over.

| management.{zwsp}leader-election.{zwsp}fail-over.back-off-initial-interval
| `long`

**Constraint:** `> 0`
```

| 1000
| The initial interval (milliseconds) to back-off when retrying a fail-over.

| management.{zwsp}leader-election.{zwsp}fail-over.back-off-max-interval
| `long`

Constraint: `> 0`
| 10000
| The maximum interval (milliseconds) to back-off when retrying a fail-over.

| management.{zwsp}leader-election.{zwsp}fail-over.back-off-multiplier
| `double`

Constraint: `>= 1.0`
| 2.0
| The multiplier to apply to the back-off interval between each retry of a fail-over.

| management.{zwsp}leader-election.{zwsp}mode
| `enum`

One of:

* `standalone`
* `active_active`
* `active_standby`
| standalone
|The connector s leader election mode.

standalone: +
A single instance of a connector without any leader election capabilities.

active_active: +
A participant in a cluster of connector instances where all instances are active.

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.

| management.{zwsp}queue
| `string`
| null
| The management queue name.

| management.{zwsp}session.{asterisk}
| See https://github.com/SolaceProducts/solace-spring-boot/tree/master/solace-spring-boot-starters/solace-java-spring-boot-starter#updating-your-application-properties[Spring Boot Auto-Configuration for the Solace Java API]
|
| Defines the management session. This has the same interface as that used by `solace.java.{asterisk}`.

solace.

See https://github.com/SolaceProducts/solace-spring-boot/tree/master/solace-
spring-boot-starters/solace-java-spring-boot-starter#updating-your-application-
properties[Spring Boot Auto-Configuration for the Solace Java API] for more info.

| security.{zwsp}enabled
| `boolean`
| true
| If `true`, security is enabled. Otherwise, anyone has access to the connector⎵s
endpoints.

| security.{zwsp}users[<index>].name
| `string`
| null
| The name of the user.

| security.{zwsp}users[<index>].password
| `string`
| null
| The password for the user.

| security.{zwsp}users[<index>].roles
| `list<string>`

Valid values:

* `admin`
| 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.
|===

:leveloffset: 2
:leveloffset: +2

[#_workflow_configuration_options]
= 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.

solace.

```
[cols="2m,2a,1m,4a", options="header"]
|===
| Config Option
| Type
| Default Value
| Description

| enabled
| `boolean`
| false
| If `true`, the workflow is enabled.

[NOTE]
====
Cannot be set at the default workflow level.
====

| transform.enabled
| `boolean`
| false
| If `true`, message transformation is enabled for the workflow.

Disables the legacy `transform-headers.{asterisk}` and `transform-
payload.{asterisk}` options.

See <<_message_transform, Message Transforms>> for more info.

| transform.{zwsp}source-payload.{zwsp}content-type
| `string`

One of:

* `application/{zwsp}vnd.solace.{zwsp}micro-integration.{zwsp}unspecified`
* `application/json`
| application/{zwsp}vnd.solace.{zwsp}micro-integration.{zwsp}unspecified
| The content type to interpret the source payload as.

See <<_message_transform_content_type_interpretation, Content Type
Interpretation>> for more info.

| transform.{zwsp}target-payload.{zwsp}content-type
| `string`

One of:

* `application/{zwsp}vnd.solace.{zwsp}micro-integration.{zwsp}unspecified`
* `application/json`
| application/{zwsp}vnd.solace.{zwsp}micro-integration.{zwsp}unspecified
| The content type to interpret and serialize the target payload as.
```

See <<_message_transform_content_type_interpretation, Content Type
Interpretation>> for more info.

| transform.{zwsp}expressions[<index>].{zwsp}transform
| `string`

A SpEL expression
|
| A SpEL expression at some `<index>` in the ordered list of expressions to
transform the message.

See <<_message_transform, Message Transform>> for more info.

| transform-headers.{zwsp}expressions
| `Map<{zwsp}string, string>`

Key: +
A header name.

Value: +
A SpEL string that accepts `headers` as parameters.
| empty map
|
[WARNING]
====
Deprecated. Use <<_message_transform, Message Transforms>>
(`transform.{asterisk}`) instead.
====

A mapping of header names to header value SpEL expressions.

The SpEL context contains the `headers` parameter that can be used to read the
input message☐s headers.

| acknowledgment.{zwsp}publish-async
| `boolean`
| false
| If `true`, 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.

| acknowledgment.{zwsp}back-pressure-threshold
| 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

solace.

```
acknowledgments to catch up.

| acknowledgment.{zwsp}publish-timeout
| 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.

|===

:leveloffset: 2

== License

This project is licensed under the Solace Community License, Version 1.0. - See
the `LICENSE` file for details.

:leveloffset: +1




== Support

Support is offered best effort via our https://solace.community/[Solace Developer
Community].

Premium support options are available, please https://solace.com/contact-
us[Contact Solace].

:leveloffset: 2
```

solace.