pubsubplus-connector-file-events

User Guide

Solace Corporation

Version 2.0.0

solace.

Table of Contents

Preface

Solace PubSub+ File to Events Connector

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 File To Events 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 File To Events producer destination, `producer-destination`.
- Workflow 1 is consuming messages from the File To Events 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
- File To Events

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-file-events-2.0.0.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:

```
pubsubplus-connector-file-events

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 have 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.

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-file-events-2.0.0.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.

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 File To Events 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> # Queue name
          binder: (solace|file-events) # Input system
        output-<workflow-id>:
          destination: <output-destination> # Topic name
          binder: (solace|file-events) # Output system

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

The connector only supports workflows in the directions of:

- `solace` → `File To Events`
- `File To Events` → `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

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

For example:

```
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
```

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

        # The only file-events binder
        file-events1:
          type: file-events
          # 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: file-events1
        output-0:
          destination: <output-destination>
          binder: solace1 # Reference 1st solace binder
        input-1:
          destination: <input-destination>
          binder: file-events1
        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 `file-events`, 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.

include::../../../snippets/attributes/common.adoc

File Source Connection Details

The Spring Cloud Stream Binder for File uses the following configuration to configure Source Connector

```
spring:
  cloud:
    stream:
      bindings:
        input-0:
          destination: # Absolute file path of source and sink file/directory. In
general for Static, Directory replication we need to configure source and sink paths
in a separate config file since multiple files and directories are supported, provide
the absolute location of config file(Create a file with extension .cfg and add your
path as per format at the end. Each line represents a source and sink path). In case
of dynamic file we can configure the source and sink path without the need for
separate config file since only one dynamic file is supported for replication.
          # Format to configure file location(absolute-source-file-path|absolute-sink-
filepath). This format applies for Static, Directory and Dynamic files.
          binder: file
        output-0:
          destination: # configure solace topic - File events are published to this
topic. This topic should be added as subscription to Sink Connector queue
          binder: solace
```

include::../../../snippets/attributes/common.adoc

File Sink Connection Details

The Spring Cloud Stream Binder for File uses the following configuration to configure Sink Connector

```
spring:
  cloud:
    stream:
      bindings:
        output-0:
          destination: # Absolute base destination path of Sink file or directory.
This value is used when dest_file_name_type property is set to 1, 4 or 5
          binder: file
        input-0:
          destination: # configure sink connector queue where file events sent by
source connector are spooled
          binder: solace
```

User-configured Header Transforms

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

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
`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:

```
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 and file-events 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 User-configured Header Transforms feature to set the following headers:

Header Name	Type	Values	Applies To	Description
`scst_targetDestination`	`string`	Any valid destination name	Solace, File To Events	Specifies the name of the dynamic destination to publish to. Setting this header overrides the configured destination.
`solace_scst_targetDestinationType`	`string`	`(queue|topic)`	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.transform-headers` 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 file-events 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.

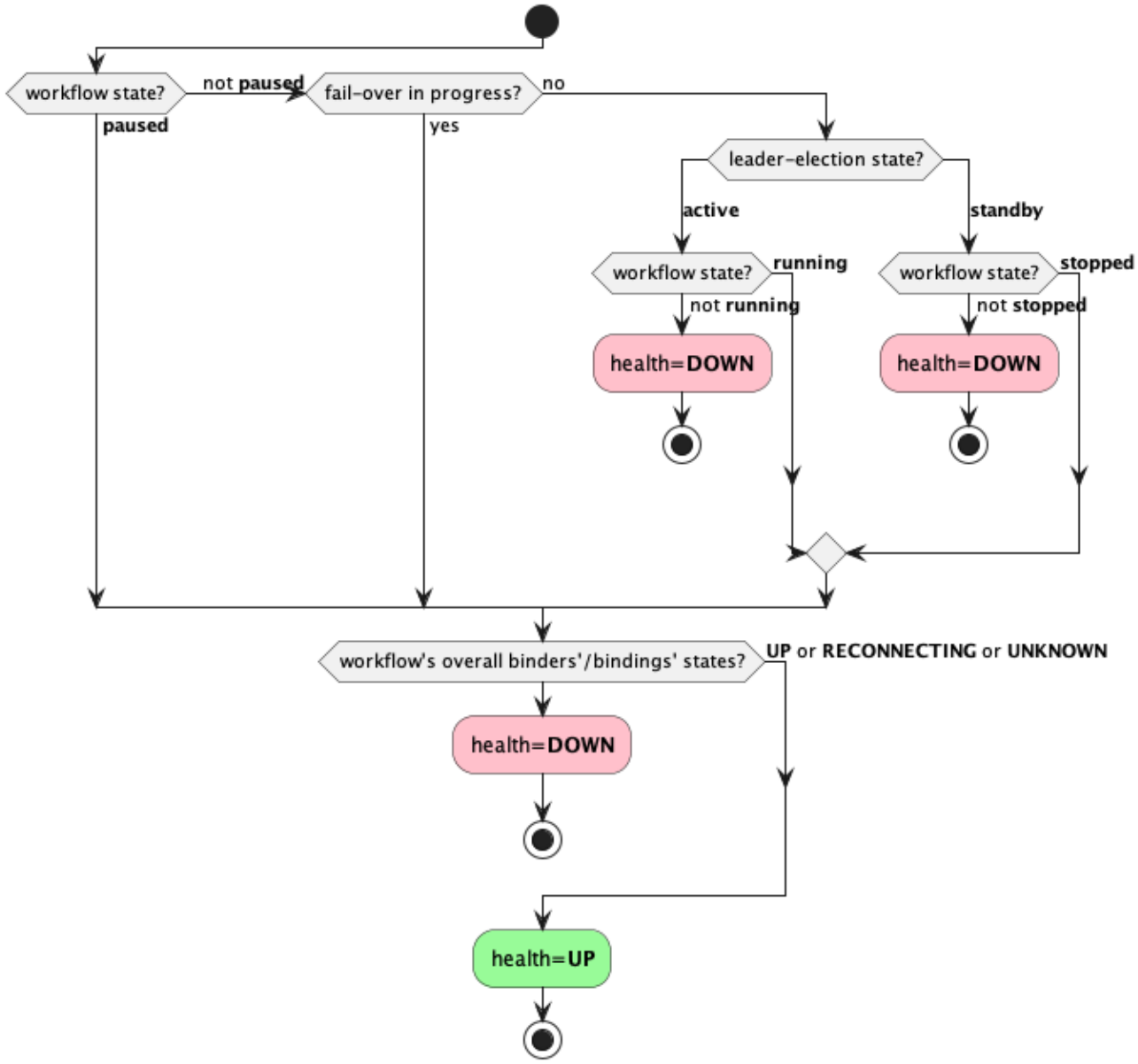

Figure 1. Workflow Health Resolution Diagram

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 three leader election modes for redundancy:

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:

```
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:

```
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).

IIf 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
`/leaderelection`	Read (HTTP `GET`)	**Request:** None. **Response:** `{ "mode": { "type": "(standalone | active_active | active_standby)", ① "state": "(active | standby)", ② "source": { ③ "queue": "<management-queue-name>", "host": "<management-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`.

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

`solace.connector.process`	`Timer`	`type: channel`
		
		`name: <bindingName>`
		
		`result: (success|failure)`
		
		`exception: (none|exception simple class name)`
			The processing time.	This meter is a rename of `spring.integration.send` whose `name` tag matches a binding name.

`solace.connector.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 `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.message.size.payload`	`DistributionSummary` Base Units: `bytes`	`name: <bindingName>`	The message payload size.	

Name	Type	Tags	Description	Notes
solace.connector.message.size.total	DistributionSummary Base Units: bytes	name: <bindingName>	The total message size.	
solace.connector.publish.ack	Counter Base Units: acknowledgments	name: <bindingName> result: (success|failure) exception: (none|exception simple class name)	The publish acknowledgment count.	

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, whehter 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 Sprint 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

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

Config Option	Type	Valid Values	Default Value	Description
`management.leader-election.fail-over.max-attempts`	`int`	`> 0`	`3`	The maximum number of attempts to perform a fail-over.
`management.leader-election.fail-over.back-off-initial-interval`	`long`	`> 0`	`1000`	The initial interval (milliseconds) to back-off when retrying a fail-over.
`management.leader-election.fail-over.back-off-max-interval`	`long`	`> 0`	`10000`	The maximum interval (milliseconds) to back-off when retrying a fail-over.
`management.leader-election.fail-over.back-off-multiplier`	`double`	`>= 1.0`	`2.0`	The multiplier to apply to the back-off interval between each retry of a fail-over.

Config Option	Type	Valid Values	Default Value	Description
management.leader-election.mode	enum	(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.queue	string	any	null	The management queue name.
management.session.*		See Spring Boot Auto-Configuration for the Solace Java API		Defines the management session. This has the same interface as that used by solace.java.*. See Spring Boot Auto-Configuration for the Solace Java API for more info.
security.enabled	boolean	(true|false)	true	If true, security is enabled. Otherwise, anyone has access to the connector's endpoints.
security.users[<index>].name	string	any	null	The name of the user.
security.users[<index>].password	string	any	null	The password for the user.
security.users[<index>].roles	list<string>	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.

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	Applicable Scopes	Type	Valid Values	Default Value	Description
`enabled`	Per-Workflow	`boolean`	`(true|false)`	`false`	If `true`, the workflow is enabled.
`transform-headers.expressions`	Per-Workflow Default	`Map<string,string>`	**Key:** A header name. **Value:** A SpEL string that accepts `headers` as parameters.	`empty map`	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.publish-async`	Per-Workflow Default	`boolean`	`(true|false)`	`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.

Config Option	Applicable Scopes	Type	Valid Values	Default Value	Description
`acknowledgment.back-pressure-threshold`	Per-Workflow Default	`int`	>= 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.
`acknowledgment.publish-timeout`	Per-Workflow Default	`int`	>= -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.

File Events Configuration

File Events Source(Events to file) Configuration Options

These configuration options are all prefixed by `file-events.source.`:

Config Option	Type	Valid Values	Default Value	Description
`scheduler.restart_time_sec`	`int`	`any`	`0`	This property enables source connector to run at periodic intervals. This is needed when you want to replicate your modified files periodically under a directory. Integer value representing number of seconds to wait and start the next replication. Set to Zero if scheduler is not required(Recommended for Dynamic file, since connector first replication runs till EOD is reached). Following is scheduler behaviour w.r.t eod_time_sec 1. restart_time_sec: 10 and eod_time_sec: 104400(5 A.M converted to number of seconds 24+5)*3600 , source connector will stop at 5 A.M 2. restart_time_sec: 10 and eod_time_sec: -1 , source connector will keep on running 3. restart_time_sec: 10 and eod_time_sec: 0 , source connector will exit at midnight 4. restart_time_sec: 0 and eod_time_sec: -1 , source connector will exit after first replication

Config Option	Type	Valid Values	Default Value	Description
general.miID	string	any	empty	Unique ID for the connector instance
general.file_type	int	1,2,3,4	empty	Type of file (1 → static 2 → Dynamic 3 → Directory first level 4 → Directory recursive)
general.state_backup_path	string	any	empty	Absolute file location to store Source Connector's state information. Only used in case file_to_events is enabled. The file should end in .cfg format(Ex:<path>/source_connector_state_backup.cfg. The file will be created by connector if it doesn't exist.
general.max_files_allowed	int	any	99999	Set this value to limit the number of files in a replication. Connector will consider the files for replication until the limit is reached
general.clear_state_on_eod	int	0, 1	0	Connector preserves the last successful file state in backup file, so that it resumes from checkpoint on next restart. If this property is set to 1 connector will clear the file state in backup file when End of Day configured time is reached. Set to 0 if file state in backup file need not be cleared
general.eod_time_sec	int	-1,0, any number > 0	-1	Connector will exit once configured EOD is elapsed. For example if connector need to be exited every day at 5 A.M EOD should be configured to 104400. The formula to calculate EOD is (24 Hours + (number of hours(24 hour format) when connector should exit))*3600. In our example (24+5)*3600 = 104400. Set this to 0 is connector need to exit at midnight every day and -1 to disable this check

Config Option	Type	Valid Values	Default Value	Description
`data_events.connectorStartEvent`	`boolean`	`true, false`	`false`	If Set to true, An additional event with Header(EVENT_TYPE → "START") will be published to the base destination topic name once a new location path is processed Additional Header included with Data Events are are follows SRC_PATH - Source File Path. The file being read DEST_PATH - Destination File Path if configured in the source configuration

Config Option	Type	Valid Values	Default Value	Description
`data_events.connectorCompleteEvent`	`boolean`	`true, false`	`false`	If Set to true, An additional event with Header(EVENT_TYPE → "COMPLETE") will be published to the base destination topic name once all the files are completely processed in the configured location path Additional Header included with Data Events are are follows SRC_PATH - Source File Path. The file being read DEST_PATH - Destination File Path if configured in the source configuration TOTAL_FILES - Total Files Processed TOTAL_EVENTS - Total Events Processed for all files TOTAL_DATA_PROCESSED - Total Data Processed for all files

Config Option	Type	Valid Values	Default Value	Description
data_events.fileStartEvent	boolean	true, false	false	If Set to true, An additional event with Header(EVENT_TYPE → "FILE_START") will be published to the base destination topic name once a new file processing starts Additional Header included with Data Events are are follows SRC_PATH - Source File Path. The file being read DEST_PATH - Destination File Path if configured in the source configuration
data_events.fileCompleteEvent	boolean	true, false	false	If Set to true, An additional event with Header(EVENT_TYPE → "FILE_COMPLETE") will be published to the base destination topic name once the file is completely processed. Additional Header included with Data Events are are follows SRC_PATH - Source File Path. The file being read DEST_PATH - Destination File Path if configured in the source configuration TOTAL_EVENTS - Total Events Processed for the current file TOTAL_DATA_PROCESSED - Total Data Processed for the current file

Config Option	Type	Valid Values	Default Value	Description
advanced.fileFormat	int	1,2,3,4	1	1 = Line by line file streaming. 2 = Delimited file streaming. Delimiter can be defined in the parameter eventDelimiter. 3 = JSON File Streaming. 4 = XML File Streaming
advanced.eventDelimiter	Char	any	empty	provide an event Delimiter. Events in the files are separated by this character.
advanced.paramDelimiter	String	any	empty	In case the fileFormat values are 1 and 2, use a param Delimiter to map the parameters to headers and create a final json payload. Use with paramHeaderMap
advanced.paramHeaderMap	String	any	empty	Provide a header map to use dynamicTopic OR to create a json payload from a delimited file if sendDelimitedEventPayloadAsBinary is set to false. Example - 'SNO,Employee_name,Employee_address,city'. To send payload as raw binary, set sendDelimitedEventPayloadAsBinary to true
advanced.sendDelimitedEventPayloadAsBinary	boolean	true, false	true	If Set to true, the event/record payload for Delimited file (fileFormat values 1 and 2) will be published as binary. If set to false, the payload will be sent as json if advanced.paramDelimiter and advanced.paramHeaderMap are set

Config Option	Type	Valid Values	Default Value	Description
advanced.dynamicTopic	String	any	empty	For FileFormats value 1 and 2 use when paramDelimiter and paramHeaderMap are defined. Add dynamic column values of the delimited file in the topic Structure. For example if paramDelimiter is "," and paramHeaderMap is "employee_id,first_name,last_name,city,country" Dynamic topic can be set as 'solace/dynamic/topic/${param-1}/${param-5}/${city}'
advanced.jsonPath	String	any	empty	provide a jsonPath filter to stream json objects from a json file. Use with fileFormat value 3 i.e. JSON File Streaming
advanced.xPath	String	any	empty	provide a xPath filter to stream xml objects from a xml file. Use with fileFormat value 4 i.e. XML File Streaming
directory_wildcard.wildcard_type	int	0, 1, 2	0	Set this property to apply regular expression on files. 0 → disabled 1 → whitelist files or directory 2 → blacklist files or directory
directory_wildcard.config_path	string	any	empty	provide absolute file path location containing regular expression. The file should be in .cfg format

Config Option	Type	Valid Values	Default Value	Description
directory_replication.start_time	int	-1, 0, any number > 0	-1	Set this property to filter files in directory based on modified time. -1 will consider the timestamp in checkpoint if available or will replicate all files again. 0 will override the timestamp in checkpoint and will consider files modified since midnight. Any value(epoch time) other than 0 or -1 will consider files modified after the epoch time
solace_out.lvq	string	any	empty	Provide the LVQ name configured on Solace broker. Connector will connect to this queue on start to fetch checkpoint information.

Config Option	Type	Valid Values	Default Value	Description
solace_out.destination	string	any	empty	LVQ topic - Connector will publish checkpoint information and the base topic will be same as output destination topic configured in connection details section. LVQ topic is built as follows 1. In case of Static or Directory replication connector will append /checkpoint to base topic(<output-destination-topic>) and will publish data. The LVQ created on Solace broker should have this subscription <output-destination-topic>/checkpoint 2. In case of Dynamic file connector will not append /checkpoint to base topic(<output-destination-topic>) and will publish data. The LVQ created on Solace broker should have this subscription <output-destination-topic>.

File Events Sink (Events to File)

Dynamic Variable expressions

These expressions can be used inside the following configuration values.

1. Output binding destination. (Destination File Path) `spring.cloud.stream.bindings.output-<>.destination`
2. Advanced options `advanced.emptyEventsSubstitute` and `advanced.uniqueEventIdentifyingHeader`
3. File Pre Processors defined under configuration pre_process.<> - `prependToFile`, `prependToFirstEvent`, `prependToEvents`
4. File Post Processors defined under configuration post_process.<> - `appendToEvents`

Dynamic Variable	Expression	Description
UMAP	${UMAP(<key_name>)}	Replaces the dynamic variable expression with the User Property map Value associated with the key_name defined in the expression. For example ${UMAP(UNIQUE_FILE_ID))} will replace this expression with the value of the User Property Map key UNIQUE_FILE_ID
UMAP_E	${UMAP_E(<key_name>)}	Works exactly like ${UMAP(<key_name>)}, but any special characters defined with configuration parameter advanced.escapeCharsRegex present inside the corresponding value will be replaced with advanced.escapeCharsReplacement value
HEADER	${HEADER(<key_name>)}	Replaces the dynamic variable expression with the Header Value associated with the key_name defined in the expression For example ${HEADER(UNIQUE_FILE_ID))} will replace this expression with the value of the Header key UNIQUE_FILE_ID
HEADER_E	${HEADER_E(<key_name>)}	Works exactly like ${HEADER(<key_name>)}, but any special characters defined with configuration parameter advanced.escapeCharsRegex present inside the corresponding value will be replaced with advanced.escapeCharsReplacement value

Dynamic Variable	Expression	Description
TOPIC	${TOPIC(<level>)}	Replaces the dynamic variable expression with the Source topic level associated with the level defined in the expression For example if source topic is solace/acme/orders/books/canada/init ${TOPIC(1)} will replace the expression with value solace ${TOPIC(3)} will replace the expression with value orders ${TOPIC(-2)} will replace the expression with second level value from the end i.e. canada ${TOPIC(0)} will replace the expression with entire source topic value solace/acme/orders/books/canada/init
TOPIC_E	${TOPIC_E(<level>)}	Works exactly like ${TOPIC(<level>)}, but any special characters defined with configuration parameter advanced.escapeCharsRegex present inside the corresponding value will be replaced with advanced.escapeCharsReplacement value
DATE	${DATE(<specifier>)}	Replaces the dynamic variable expression with the Date specifier value associated with the specifier defined in the expression For example, ${DATE%1$tY%1$tm%1$td} will replace the expression with value "YYYYMMDD" i.e. "20240101"
DATE_E	${DATE_E(<specifier>)}	Works exactly like ${DATE(<specifier>)}, but any special characters defined with configuration parameter advanced.escapeCharsRegex present inside the corresponding value will be replaced with advanced.escapeCharsReplacement value
AUTO_INCREMENT	${AUTO_INCREMENT}	Replaces the dynamic variable expression with the auto increment integer value associated with the file count or the event count depending on the field it is used inside. This count is maintained by the application in the application state.

Date Format Specifiers

These specifiers can be used in the DATE expressions such as ${DATE(<specifier>)} and ${DATE_E(<specifier>)} wherever allowed -

1. Output binding destination. (Destination File Path) `spring.cloud.stream.bindings.output-<>.destination`
2. Advanced option `advanced.emptyEventsSubstitute`
3. File Pre Processors defined under configuration pre_process.<> - `prependToFile`, `prependToFirstEvent`, `prependToEvents`
4. File Post Processors defined under configuration post_process.<> - `appendToEvents`

Character	Specifier %1$t<c>	Description	Example
c	%1$tc	Complete date and time	Mon May 04 09:51:52 CDT 2009
F	%1$tF	ISO 8601 date	2004-02-09
D	%1$tD	U.S. formatted date (month/day/year)	02/09/2004
T	%1$tT	24-hour time	18:05:19
r	%1$tr	12-hour time	06:05:19 pm
R	%1$tR	24-hour time, no seconds	18:05
Y	%1$tY	Four-digit year (with leading zeroes)	2004
y	%1$ty	Last two digits of the year (with leading zeroes)	04
C	%1$tC	First two digits of the year (with leading zeroes)	20
B	%1$tB	Full month name	February
b	%1$tb	Abbreviated month name	Feb
m	%1$tm	Two-digit month (with leading zeroes)	02
d	%1$td	Two-digit day (with leading zeroes)	03
e	%1$te	Two-digit day (without leading zeroes)	9
A	%1$tA	Full weekday name	Monday
a	%1$ta	Abbreviated weekday name	Mon
j	%1$tj	Three-digit day of the year (with leading zeroes)	069
H	%1$tH	Two-digit hour (with leading zeroes), between 00 and 23	18
k	%1$tk	Two-digit hour (without leading zeroes), between 0 and 23	18

Character	Specifier %1$t<c>	Description	Example
I	%1$tI	Two-digit hour (with leading zeroes), between 01 and 12	06
l	%1$tl	Two-digit hour (without leading zeroes), between 1 and 12	6
M	%1$tM	Two-digit minutes (with leading zeroes)	05
S	%1$tS	Two-digit seconds (with leading zeroes)	19
L	%1$tL	Three-digit milliseconds (with leading zeroes)	047
N	%1$tN	Nine-digit nanoseconds (with leading zeroes)	047000000
P	%1$tP	Uppercase morning or afternoon marker	PM
p	%1$tp	Lowercase morning or afternoon marker	pm
z	%1$tz	RFC 822 numeric offset from GMT	-0800
Z	%1$tZ	Time zone	PST
s	%1$ts	Seconds since 1970-01-01 00:00:00 GMT	1078884319
Q	%1$tQ	Milliseconds since 1970-01-01 00:00:00 GMT	1078884319047

Examples

`${DATE(%1$tF %1$tT.%1$tL)}` will give output as `"2024-08-30 12:00:05.878"`

`${DATE(%1$tF %1$tr)}` will give output as `"2024-08-30 12:00:05 PM"`

`${DATE(%1$tT.%1$tN)}` will give output as `"12:00:05.047000000"`

File Events Sink (Events to File) Configuration Options

These configuration options are all prefixed by `file-events.sink.`:

Config Option	Type	Valid Values	Default Value	Description
`general.miID`	`string`	`any`	`empty`	Unique ID for the connector instance
`general.storeBackupStateRemotely`	`boolean`	`true,false`	`false`	When set to true, Sink connector will create the backup state file on the remote Channel (GCS, SFTP, FTP, FTPs) configured
`general.state_backup_path`	`string`	`any`	`empty`	Absolute file location to store Sink Connector's checkpoint information. The file should end in .cfg format(Ex:<path>/sink_connector_state_backup.cfg. The file will be created by connector if it doesn't exist.
`advanced.maxEventsPerFile`	`int`	`0, 1, >1`	`0`	Default value 0 (unlimited). Maximum events allowed to be written in a file. To respect this, destination file name must include dynamic fields such as ${AUTO_INCREMENT} or any header parameter for successful rollover to a new file. If not configured properly, same file will be appended as a result.

Config Option	Type	Valid Values	Default Value	Description
advanced.maxFileSize	int	0, >0	0	Default value 0 (unlimited). Maximum file size in bytes allowed. To respect this, destination file name must include dynamic fields such as ${AUTO_INCREMENT} or any header parameter for successful rollover to a new file. If not configured properly, same file will be appended as a result. Irrespective of the maxFileSize defined, atleast 1 event will always be written to a file.
advanced.checkDynamicFileNameChangesEverytime	boolean	true, false	false	Default value false. If set to true, on every new event will check if file name changes are required. For example if a date is present in the file name. Or a Header which is expected to change often. ${AUTO_INCREMENT} will only take affect if there are other dynamic variables in the file name which are expected to change. ${AUTO_INCREMENT} will not change the name of the file in case checkDynamicFileNameChangesEverytime is set to true

Config Option	Type	Valid Values	Default Value	Description
advanced.escapeCharsRegex	string	any	empty	Regex to escape characters present inside the dynamic variable. Works if variable categories are used as ${UMAP_E(<>)}, ${TOPIC_E(<>)}, ${DATE_E(<>)}, ${HEADER_E(<>)}
advanced.escapeCharsReplacement	string	any	"@"	Default Value "@". Only works when escapeCharsRegex is configured
advanced.ignoreEmptyEvents	boolean	true, false	true	Default value true. If set to false, need to provide value for emptyEventsSubstitute. If no value for emptyEventsSubstitute is defined, message will be acked as a failure.
advanced.emptyEventsSubstitute	string	any	empty	String value to be replaced in case of empty events. Doesn't work if ignoreEmptyEvents is set to true OR if empty event is a file close trigger event Dynamic Variable expression can be used with this option. You can define multiple expressions inside the string value of this configuration For example "${UMAP(<key_name>)}, ${HEADER(<key_name>)}, ${TOPIC(<level>)}, ${DATE(<specifier>)}, ${AUTO_INCREMENT}" The ${AUTO_INCREMENT} would give the event count associated with the current destination file.

Config Option	Type	Valid Values	Default Value	Description
advanced.preventDuplicateEvents	boolean	true, false	false	Default value false. If set to true duplicate events will be ignored. Max events in Duplicate check cache 260.
advanced.preventDuplicateOnlyForRedeliveredEvents	boolean	true, false	true	Default value true. Used only when preventDuplicateEvents set to true. If set to true duplicate key will only be checked for Redelivered events. If false, will be checked for all events.

Config Option	Type	Valid Values	Default Value	Description
advanced.uniqueEventIdentifyingHeader	string	any	empty	Used only when preventDuplicateEvents set to true. If value is set, will be treated as duplicate check key. If not set then complete event payload will be set as Duplicate key. You can either add the header name directly like uniqueEventIdentifyingHeader: "PROPERTY1". OR Can also add a combination of headers and merge the values using the Dynamic Variable Replacement like "${HEADER(PROPERTY1)}-${HEADER(PROPERTY2)}" You can add more variables to make the identifier value unique. Chose from ${UMAP(<key_name>)}, ${HEADER(<key_name>)}, ${TOPIC(<level>)}, ${DATE(<specifier>)}, ${AUTO_INCREMENT} The ${AUTO_INCREMENT} would give the event count associated with the current destination file.
advanced.closeFileTriggerHeaderName	string	any	empty	If value is set, File will be closed if this header field exists in the event. Works in combination with closeFileTriggerHeaderValue

Config Option	Type	Valid Values	Default Value	Description
advanced.closeFileTriggerHeaderValue	string	any	empty	Default value "". Works if value of closeFileTriggerHeaderName is set. File will be closed if set to "*" i.e. all values OR specific value configured matches the exact string value with the event header
advanced.ignorePayloadOfCloseTriggerEvent	boolean	true, false	false	Default Value false. If set to true any payload present in the file closing trigger event will not be written to file. Works in combination with closeFileTriggerHeaderName and closeFileTriggerHeaderValue
advanced.ignoreWritingPayloadOfAllEvents	boolean	true, false	false	Default Value false. If set to true any payload present in the event will not be written to file. Will make sense to use if only headers or dynamic parameters need to be written to the destination file using prependToEvents or appendToEvents options.

Config Option	Type	Valid Values	Default Value	Description
pre_process.prependToFile	string	any	empty	If value is set, Content of this configuration will be added to the beginning of the file. Dynamic Variable expression can be used with this option. You can define multiple expressions inside the string value of this configuration For example “${UMAP(<key_name>)}, ${HEADER(<key_name>)}, ${TOPIC(<level>)}, ${DATE(<specifier>)}, ${AUTO_INCREMENT}” The ${AUTO_INCREMENT} would give the file count associated with the current destination file.
pre_process.prependToFirstEvent	string	any	empty	If value is set, the content of this configuration only gets added before the first event. Dynamic Variable expression can be used with this option. You can define multiple expressions inside the string value of this configuration For example “${UMAP(<key_name>)}, ${HEADER(<key_name>)}, ${TOPIC(<level>)}, ${DATE(<specifier>)}, ${AUTO_INCREMENT}” The ${AUTO_INCREMENT} would give the event count associated with the current destination file. Helps to manage creation of specific file formats such as JSON.

Config Option	Type	Valid Values	Default Value	Description
pre_process.prependToEvents	string	any	empty	If value is set, the content of this configuration gets added before all event. Would be ignored for first event if prependToFirstEvent parameter is configured. Dynamic Variable expression can be used with this option. You can define multiple expressions inside the string value of this configuration For example “${UMAP(<key_name>)}, ${HEADER(<key_name>)}, ${TOPIC(<level>)}, ${DATE(<specifier>)}, ${AUTO_INCREMENT}” The ${AUTO_INCREMENT} would give the event count associated with the current destination file.
post_process.appendToFile	string	any	empty	If value is set, Content of this configuration will be added to the end of the file.

Config Option	Type	Valid Values	Default Value	Description
post_process.appendToEvents	string	any	empty	If value is set, the content of this configuration gets added to the end of all events. Dynamic Variable expression can be used with this option. You can define multiple expressions inside the string value of this configuration For example “${UMAP(<key_name>)}, ${HEADER(<key_name>)}, ${TOPIC(<level>)}, ${DATE(<specifier>)}, ${AUTO_INCREMENT}” The ${AUTO_INCREMENT} would give the event count associated with the current destination file.

Remote Protocol Configuration

By default, Micro-Integration will automatically read and write files on the local file system unless the supported remote protocols are enabled in the configuration.

The following remote protocols are supported

1. Google Cloud Storage
2. SSH File Transfer Protocol or Secure File Transfer Protocol (SFTP)
3. File Transfer Protocol (FTP/FTPs)

The current version only supports remote connections for Sink flow (Events to File)

Google Cloud Storage

Google Cloud Storage is a service for storing objects in Google Cloud.

An object is an immutable piece of data consisting of a file of any format. You store objects in containers called buckets.

All buckets are associated with a project, and you can group your projects under an organization.

These configuration options are all prefixed by `file-events.sink.`

Config Option	Type	Valid Values	Default Value	Description
`gcs.enabled`	`boolean`	`false, true`	`false`	Set this value to false to disable and true to enable Google Cloud Storage as source location. Default false, the file will be searched for on the local source machine and if set to true, the files/blobs will be pulled from a remote location on the Google Cloud Storage If enabled, advanced.maxEventsPerFile property will automatically set to 1.

Config Option	Type	Valid Values	Default Value	Description
gcs.projectId	string	any	empty	ProjectId value from Google Cloud Storage.
gcs.credentialsFilePath	string	any	empty	Credentials file path in JSON format to access Google Cloud Storage.
gcs.enableBulkComposeStrategy	boolean	true,false	true	When set to true, the sink connector will request to compose multipart blobs in bulk at the end of receiving the last Multipart. If set to false, sequential compose strategy will be applied to compose all multiparts to create the final file
gcs.retrySettings.maxAttempts	int	Integer Value	10	Maximum Retry Attempts in case of connection failure
gcs.retrySettings.retryDelayMultiplier	double	Double Value	3.0	Retry Delay Multiplier in case of connection failure
gcs.retrySettings.maxDurationMinutes	int	Integer Value	5	Maximum Duration of retries in minutes in case of connection failure

Secure File Transfer Protocol (SFTP)

Secure File Transfer Protocol (SFTP) is a network protocol for securely accessing, transferring and managing large files and sensitive data.

SFTP uses SSH to transfer files and requires that the client be authenticated by the server.

Commands and data are encrypted to prevent passwords and other sensitive information from being exposed to the network in plain text.

The authenticaiton methods supported are `Password` OR using `Private Key`

These configuration options are all prefixed by `file-events.sink.`

Config Option	Type	Valid Values	Default Value	Description
`sftp_settings.enabled`	`boolean`	`false, true`	`false`	Set this value to false to disable and true to enable SFTP source location. Default false, the file will be searched for on the local source machine and if set to true, the files will be pulled from a remote location on the sftp server
`sftp_settings.ip`	`string`	`any`	`empty`	SFTP Server IP
`sftp_settings.port`	`int`	`any`	`22`	SFTP server port number
`sftp_settings.user`	`string`	`any`	`empty`	SFTP user username
`sftp_settings.password`	`string`	`any`	`empty`	SFTP user password
`sftp_settings.strictHostKeyChecking`	`string`	`yes, no`	`yes`	Enable or disable the Strict Host Key checking while creating a connection to the sftp server.
`sftp_settings.privateKeyPath`	`string`	`any`	`empty`	Path to the privateKey File to authenticate using private key instead of password

File Transfer Protocol (FTP/FTPs)

The File Transfer Protocol (FTP) is a standard communication protocol used for the transfer of computer files from a server to a client on a computer network.

FTP is built on a client–server model architecture using separate control and data connections between the client and the server.

FTP users may authenticate themselves with a plain-text sign-in protocol, normally in the form of a username and password, but can connect anonymously if the server is configured to allow it.

For secure transmission that protects the username and password, and encrypts the content, FTP is often secured with SSL/TLS (FTPS)

Both `FTP` or `FTP secure` are supported

These configuration options are all prefixed by `file-events.sink.`

Config Option	Type	Valid Values	Default Value	Description
`ftp_settings.enabled`	`boolean`	`false, true`	`false`	Set this value to false to disable and true to enable FTP/FTPs source location. Default false, the file will be searched for on the local source machine and if set to true, the files will be pulled from a remote location on the ftp server
`ftp_settings.ip`	`string`	`any`	`empty`	FTP Server IP
`ftp_settings.port`	`int`	`any`	`21`	FTP server port number
`ftp_settings.user`	`string`	`any`	`empty`	FTP user username
`ftp_settings.password`	`string`	`any`	`empty`	FTP user password
`ftp_settings.secured`	`boolean`	`true,false`	`false`	For connecting over FTPs - TLS/secured
`ftp_settings.localPassiveMode`	`boolean`	`true,false`	`false`	Sets the data connection mode to PASSIVE_LOCAL_DATA_CONNECTION_MODE

File Mesh Manager Configuration

All the file replications can be monitored on File Mesh Manager dashboard. The connector publishes the health, replication state to File Mesh Manager which analyses and represents the data in graphical representation.

File Mesh Manager is a separate package that need to be deployed as separate instance, which is built on ReactJS, NodeJS and uses postgres or oracle as databases.

File Mesh Manager can connect to the queue on configured Solace Instance to read and process file event state.

Once processed the information is represented on Dashboard.

Configuration Options

These configuration options are all prefixed by `file-events.<sink/source>.file_mesh_manager`:

Config Option	Type	Valid Values	Default Value	Description
`file_mesh_manager.enabled`	`boolean`	`true, false`	`false`	Set this value to false to disable and true to enable sending state events to file mesh manager
`file_mesh_manager.useOutputDestinationSolaceCredentials`	`boolean`	`true, false`	`true`	true → uses solace java credentials configured, false → If different Solace instance details need to be used, configure using below properties
`file_mesh_manager.solace_ip`	`string`	`any`	`empty`	Full Solace Host address(tcp://host-name:55555)
`file_mesh_manager.solace_vpn`	`string`	`any`	`empty`	Solace VPN name
`file_mesh_manager.solace_user`	`string`	`any`	`empty`	Solace client username
`file_mesh_manager.solace_password`	`string`	`any`	`empty`	Solace client password

Config Option	Type	Valid Values	Default Value	Description
file_mesh_manager.solace_base_publish_topic	string	any	empty	Base publish topic for file mesh manager event. Connector will append adapter_id in first row and event state(start, complete, error, warning) to the topic when publishing data.Ex(<file_mesh_manager.solace_base_publish_topic>/<adapter_id>/<event_state>). This following topic subscription need to be added to File Mesh Manager queue 1. <file_mesh_manager.solace_base_publish_topic>/*/start 2. <file_mesh_manager.solace_base_publish_topic>/*/complete 3. <file_mesh_manager.solace_base_publish_topic>/*/error 4. <file_mesh_manager.solace_base_publish_topic>/*/warning
file_mesh_manager.solace_messaging_mode	string	DIRECT, PERSISTENT, NON-PERSISTENT	PERSISTENT	Set the message mode (DIRECT, PERSISTENT, NON-PERSISTENT)
file_mesh_manager.heartbeat_enabled	boolean	false, true	false	Set this to true to enable sending heartbeats to File Mesh Manager. (File Mesh Manager option need to be enabled as well). Setting it to false will disable sending heart beat events.

Config Option	Type	Valid Values	Default Value	Description
file_mesh_manager.heartbeat_interval	int	any	30	Integer value representing number of seconds to wait before sending heartbeat. This will work only if heartbeat is enabled.
file_mesh_manager.solace_base_publish_topic_heartbeat	string	any	empty	Base publish topic for heartbeat event. Connector will append adapter_id in first row and event state(heartbeat) to the topic when publishing data.Ex(<file_mesh_manager.solace_base_publish_topic_heartbeat>/<adapter_id>/<event_state>). This following topic subscription need to be added to heartbeat queue 1. <file_mesh_manager.solace_base_publish_topic_heartbeat>/*/heartbeat
file_mesh_manager.events.fileStart	boolean	true, false	false	true → send file start events for every file to File Mesh Manager false → No file start event
file_mesh_manager.events.fileComplete	boolean	true, false	false	true → send file complete events for every file to File Mesh Manager false → No file complete event

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.