pubsubplus-connector-file-events

User Guide

Solace Corporation

Version 4.0.0

solace.

Table of Contents

Preface . . . 1
Configuring Micro Integration . . . 2
File Source Connection Details . . . 2
File Sink Connection Details . . . 2
Message Mapping & Transforms . . . 3
Management and Monitoring Micro Integration . . . 4
Configuration . . . 5
File Events Source (File to Events) . . . 6
Configuring Input Destination . . . 6
FileType 1: Static Files . . . 6
FileType 2: Dynamic Files (Continuously Written) . . . 7
FileType 3: Directory Single Level (Non-Recursive) . . . 8
FileType 4: Directory Nested (Recursive) . . . 9
Adding Dynamic Dates in Paths . . . 9
Understanding Topic Configurations . . . 11
Important Note . . . 11
Topic Priority . . . 12
Last Value Queue (LVQ Checkpoint) Topic . . . 12
Configuring Queues on Solace . . . 12
Topic Subscriptions for File Types . . . 12
File Events Source (File to Events) Configuration Options . . . 14
Scheduler Configuration . . . 14
General Configuration . . . 15
Data Events Configuration . . . 16
Advanced Configuration . . . 18
Directory Wildcard Configuration . . . 25
LVQ Configuration . . . 25
Dynamic Expressions . . . 27
Introduction . . . 27
Expression Syntax . . . 27
DATE Expressions . . . 28
FILE Expressions . . . 28
TOPIC Expressions . . . 29
UMAP and HEADER Expressions . . . 29
Legacy Placeholders . . . 30
Level 2 Operators (Post-Processing Tools) . . . 30
Example Scenarios . . . 31
Troubleshooting and Best Practices . . . 31

Summary . . . 31
Date Format Specifiers . . . 33
Examples . . . 34
CSV/Delimited Events Filter Expressions . . . 35
1. Basic Comparisons . . . 35
2. Logical Operators . . . 35
3. String Operations . . . 35
4. Numeric Operations . . . 36
5. Null and Empty Checks . . . 36
6. Date Operations . . . 36
7. List and Collection Filtering . . . 36
8. Using Column Numbers Instead of Headers . . . 36
9. Regex Matching . . . 37
Source File Decryption . . . 38
Scope . . . 38
How it works . . . 38
Configuration . . . 39
Failures . . . 44
Restart and resume . . . 45
Security . . . 46
Logging . . . 47
File Mesh Manager events . . . 47
Example: custom PGP jar (EncryptDecrypt.jar) . . . 48
Example: GnuPG . . . 50
Custom decryptor class (design, not implemented) . . . 51
Implementation map . . . 53
File Streaming Modes . . . 53
File Format 1: CSV Line by Line File Streaming (Version 1) . . . 54
File Format 2: CSV - Custom Delimited Events File Streaming (Version 1) . . . 59
File Format 11: CSV Line by Line (Version 2 - Enhanced) . . . 60
File Format 12: CSV - Custom Delimited Events File Streaming (Version 2 - Enhanced) . . . 63
 Differences Between FileFormat 1/2 and 11/12 . . . 64
Column Delimiter Auto Detection . . . 65
Column Specs Configuration Guide . . . 69
Fixed-Length File Processing Configuration Guide . . . 73
File Format 6: Lines as Events . . . 76
Output Formatting (`output`) . . . 77
Example Configurations and Outputs . . . 79
Extended Example Configurations and Outputs . . . 82
File Events Sink (Events to File) . . . 85
Configuring Output Destination . . . 86

Example Usage Dynamic Expressions . . . 86
Examples . . . 86
Dynamic Variable expressions Quick guide . . . 87
File Events Sink (Events to File) Configuration Options . . . 90
General Configuration . . . 90
Advanced Configuration . . . 90
Event Pre Processor Configuration . . . 96
Event Post Processor Configuration . . . 98
Example Configurations for Quick Setup . . . 100
Single Event per File . . . 100
No Limits for Event count per file . . . 100
Max Size per file . . . 100
No Size limits per file . . . 101
Event Count and File Size limits . . . 101
Event Pre Processors . . . 101
Event Post Processors . . . 102
Generating XML Output File . . . 102
Generating JSON Output File . . . 102
Sink configuration for Source MI Events . . . 103
Remote Protocol Configuration . . . 104
Google Cloud Storage . . . 104
Secure File Transfer Protocol (SFTP) . . . 106
SFTP Proxy Configuration . . . 107
File Transfer Protocol (FTP/FTPs) . . . 109
File Mesh Manager Configuration . . . 114
Configuration Options . . . 114
License . . . 118
Support . . . 119

Preface

File Mesh, a purpose-built solution that bridges the gap between legacy file systems and modern event-driven architectures. Powered by Solace, File Mesh enables files to seamlessly participate in the event mesh ecosystem, ensuring businesses can modernize their workflows without losing the value locked in file-based processes.

With support for multiple protocols such as SFTP, FTP, FTPs, Google Cloud Storage, and S3, File Mesh provides unparalleled flexibility, enabling organizations to leverage various storage platforms as part of their event-driven architecture.

`The File Events Micro-Integration enables us to stream events from files seamlessly to Solace Event Mesh.` There are two types of File Events Micro-Integrations

`File Events Source` or `File-to-Events`: File Events can process files in formats such as CSV, delimited, JSON, and XML. It reads individual records from these files and publishes events dynamically across the event mesh on dynamic topics. This allows businesses to extract valuable data from file-based workflows and inject it into real-time streams, triggering immediate actions or processes based on that data.

`File Events Sink` or `` `Events-to-File ``: Events can be consumed from a Solace Queue, and based on dynamic, configurable settings, these events can be written to a file. This is ideal for organizations that need to store events in a persistent manner, ensuring that event data can be easily accessed or processed later as part of a file-based workflow.

- Getting Started
- Resource Requirements for Self-Managed Micro-Integrations
- Enabling Workflows

Configuring Micro Integration

- Configuring the Datapath for Your Self-Managed Micro-Integration

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
```

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
```

Message Mapping & Transforms

Mapping Message Headers and Payloads

Management and Monitoring Micro Integration

Managing and Deploying Your Self-Managed Micro-Integration

Configuration

Configuring Self-Managed Micro-Integrations

Workflow Configuration

File Events Source (File to Events)

Configuring Input Destination

The configuration property `spring.cloud.stream.bindings.input-0.destination` defines the input destination for the source micro integration (MI) instance.

This property varies based on the fileType being processed. The behavior and required setup differ for each type, as detailed below.

For dynamic expressions used in topic configurations (e.g., ${FNAME}), refer to the Dynamic Expressions section for more details.

FileType 1: Static Files

For static files (processed sequentially), create a new configuration file with the following format for each entry:

```
SourceFilePath|OptionalDestinationFilePath|OptionalCustomBaseDestinationTopic|Optional
CustomDynamicDestinationTopic
```

The configuration file can contain multiple lines (entries), and the MI will process each source path sequentially.

Column	Description
`SourceFilePath`	Absolute path of the static source file.
`OptionalDestinationFilePath`	Absolute path of the destination file. This value is included in the header metadata. The sink MI instance can use this to reference the original file name provided by the source MI.
`OptionalCustomBaseDestinationTopic`	Custom base topic for each file path, if needed. If not defined, the default output-0.destination value is used.
`OptionalCustomDynamicDestinationTopic`	Fine-grained topic per file, if needed. Metadata events are published to the base topic (if configured), while actual file events are published to this custom dynamic topic. Dynamic expressions can be used to populate topic values at runtime. For more detailed usage and to know about more dynamic expressions that can be used, please refer to the section Dynamic Expressions

Once the file is created provide the absolute path of this new configuration file to destination

```
spring.cloud.stream.bindings.input-0.destination: /path/to/static_files.cfg
```

Routing Logic:

If the dynamic topic is not defined, all events are sent to the optional custom base topic. If neither the dynamic nor base topic is defined, the MI uses the default output-0.destination.

Example

A sample configuration file entry for a static file:

```
/home/centos/local-source/sftp|/home/centos/local-
sink/sftp|solace/fc/source/sftp|solace/fc/source/sftp/${FNAME}
```

This example processes the source file at

```
/home/centos/local-source/sftp,
with destination /home/centos/local-sink/sftp,
base topic solace/fc/source/sftp,
and dynamic topic solace/fc/source/sftp/${FNAME} (where ${FNAME} resolves to the file
name).
```

The file can have multiple such lines for sequential processing.

FileType 2: Dynamic Files (Continuously Written)

For dynamic files that are continuously written to, only one file is supported per source MI instance. No new configuration file is required. Instead, assign the single line entry directly to the property:

```
spring.cloud.stream.bindings.input-0.destination:
SourceFilePath|OptionalDestinationFilePath|OptionalCustomBaseDestinationTopic|Optional
CustomDynamicDestinationTopic
```

The format follows the same structure as FileType 1.

Column	Description
`SourceFilePath`	Absolute path of the dynamic source file.
`OptionalDestinationFilePath`	Absolute path of the destination file. This value is included in the header metadata.
`OptionalCustomBaseDestinationTopic`	Custom base topic, if needed. If not defined, the default output-0.destination value is used.
`OptionalCustomDynamicDestinationTopic`	Custom dynamic topic, if needed. Supports dynamic expressions.

Routing Logic:

Same as FileType 1.

Example

Direct property assignment:

```
    `spring.cloud.stream.bindings.input-0.destination`: /var/log/dynamic-
app.log|/var/log/dynamic-sink.log|solace/dynamic/base|solace/dynamic/events/${FNAME}
This monitors the continuously written file at /var/log/dynamic-app.log, routing
metadata to solace/dynamic/base and events to solace/dynamic/events/${FNAME}.
```

FileType 3: Directory Single Level (Non-Recursive)

For single-level directories (non-recursive), use the same approach as static files, but specify directory paths instead of file paths.

Create a new configuration file with the following format:

```
SourceDirectoryPath|OptionalDestinationDirectoryPath|OptionalCustomBaseDestinationTopi
c|OptionalCustomDynamicDestinationTopic
```

The configuration file can contain multiple lines, processed sequentially. All files in the specified directories (one level deep) will be streamed.

The column descriptions are analogous to FileType 1, with "FilePath" replaced by "DirectoryPath".

Once the file is created provide the absolute path of this new configuration file to destination

```
spring.cloud.stream.bindings.input-0.destination: /path/to/dir_paths.cfg
```

Routing Logic:

Same as FileType 1.

Example

A sample configuration file entry for a single-level directory:

```
/home/centos/local-source/docs|/home/centos/local-
sink/docs|solace/fc/source/docs|solace/fc/source/docs/${FNAME}
```

This streams all files from /home/centos/local-source/docs (non-recursively), with destination directory /home/centos/local-sink/docs, base topic solace/fc/source/docs, and dynamic topic solace/fc/source/docs/${FNAME}. The file can have multiple such lines for multiple directories.

FileType 4: Directory Nested (Recursive)

For nested directories, the setup is identical to FileType 3, but the search is recursive—all subdirectories and their files will be streamed.

Use the same configuration file format:

```
SourceDirectoryPath|OptionalDestinationDirectoryPath|OptionalCustomBaseDestinationTopi
c|OptionalCustomDynamicDestinationTopic
```

The column descriptions are analogous to FileType 3.

Once the file is created provide the absolute path of this new configuration file to destination

```
spring.cloud.stream.bindings.input-0.destination: /path/to/dir_paths.cfg
```

Routing Logic:

Same as FileType 1.

Example

A sample configuration file entry for a nested directory:

```
/home/centos/local-source/projects|/home/centos/local-
sink/projects|solace/fc/source/projects|solace/fc/source/projects/${FNAME}
```

This recursively streams all files from /home/centos/local-source/projects and subdirectories, with destination directory /home/centos/local-sink/projects, base topic solace/fc/source/projects, and dynamic topic solace/fc/source/projects/${FNAME}. The file can have multiple such lines for multiple root directories.

Adding Dynamic Dates in Paths

Dynamic date expressions can be added in Source/Destination file/directory paths or topics

Example 1

```
/home/centos/local-source/${DATE(%1$tY%1$tm%1$td)}/file
```

The above configured path will resolve to basic the system's current date

```
/home/centos/local-source/20251001/file
```

Example 2

```
/home/centos/local-source/${DATE[-1d](%1$tY%1$tm%1$td)}/file
```

The above configured path will resolve to basic the system's current date minus 1 day

```
/home/centos/local-source/20250930/file
```

For more details on the usage of these dynamic expressions refer to the sections Dynamic Expressions and Date Format Specifiers

Understanding Topic Configurations

The topics all events will publish on are defined at different levels to achieve flexible routing per file/dir/path

Here's the list of different topics defined in the configuration

Topic Type	Description	Configuration
`Default Base Topic Global`	`Mandatory`	Default topic is used to stream all events - meta data events as well as file events. Configured at `spring.cloud.stream.bindings.output-0.destination`
`Default Dynamic Topic Global`	`Optional`	Default dynamic topic is used to stream file events on a topic expression that's dynamically resolved in real time. Configured at `file-events.source.advanced.dynamicTopic`
`Custom Base Topic per File/Dir Path`	`Optional`	Default Topic can be overwritten if each configured path for static/directory fileType needs to be published on different topic. Configured as Optional Custom Base Topic (Column 3) in the configuration file created for configuring paths. or In case of dynamic file type - `spring.cloud.stream.bindings.input-0.destination`
`Custom Dynamic Topic per File/Dir Path`	`Optional.`	Default dynamic Topic can be overwritten if each configured path for static/directory fileType needs to be published on different topic. Configured as Optional Custom Dynamic Topic Expression (Column 4) in the configuration file created for configuring paths. or In case of dynamic file type - `spring.cloud.stream.bindings.input-0.destination`

Important Note

The Meta Data events are always published on the base Topics (Default Base Topic Global/ Custom Base Topic per Path)

The File Events can be published on any of the topics based on how the topic priority is resolved

Topic Priority

For Meta Data events:

```
`Default Topic per File/Dir Path` >
`Default Topic`
```

For File Events

```
`Custom Dynamic Topic per File/Dir Path` >
`Custom Base Topic per File/Dir Path` >
`Default Dynamic Topic Global` >
`Default Base Topic Global`
```

Which means if `Custom Dynamic Topic per File/Dir Path` is configured it will supersede all other topic and if no other topic is configured `Default Base Topic Global` will be used

Last Value Queue (LVQ Checkpoint) Topic

The checkpoint event meant for LVQ is used by the source MI to track the its state.

This is important for preventing retransmission of files in case of scheduled or unexpected restarts

The LVQ checkpoint topic will be created by the source MI itself and will resolve to `` `${Default Base Topic Global}/checkpoint ``

To `override` the base topic for LVQ checkpoint - configure `file-events.source.lvq.destination`.

If configured the source MI will publish all checkpoints to `` `${file-events.source.lvq.destination}/checkpoint ``

`Note`: The source MI will internally append the lvq topic with "/checkpoint"

Configuring Queues on Solace

Apart from the management queues(optional), Two mandatory queues are required to be created on Solace Broker for the very least

1. The File Events Queue - For streaming file events and data events(Optional). The Subscriber can consume file events from this queue
2. The Last Value Queue - For storing the intermediate state of source MI

Topic Subscriptions for File Types

Topic subscriptions define the queues and associated topics used for message routing.

Static and Dynamic File Types

For static and dynamic file types, the following queue and topic subscriptions apply:

Queue Name	Topic Subscriptions Examples
q.file.connector.lvq (Last Value Queue)	solace/fc/source/test/checkpoint (LVQ Checkpoint topic) solace/fc/source/test (Resolved File Events topic) solace/fc/source/test (Optional Resolved Meta Data Events topic)
q.file.event.queue (File Events Queue)	solace/fc/source/test (Resolved File Events topic) solace/fc/source/test (Optional Resolved Meta Data Events topic)

Directory and Recursive Directory Options

For directory (single-level and recursive) file types, the following queue and topic subscriptions apply:

Queue Name	Topic Subscriptions Examples
q.file.connector.lvq (Last Value Queue)	solace/fc/source/test/checkpoint (LVQ Checkpoint topic)
q.file.event.queue (File Events Queue)	solace/fc/source/test (Resolved File Events topic) solace/fc/source/test (Optional Resolved Meta Data Events topic)

File Events Source (File to Events) Configuration Options

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

Scheduler Configuration

Config Option	Type	Valid Values	Default Value	Description
`scheduler.restart_time_sec`	`int`	`any`	`0`	This property enables the source connector to run at periodic intervals. This is needed when you want to replicate modified files under a directory periodically. The integer value is the number of seconds to wait before starting the next replication. Set to 0 if the scheduler is not required (recommended for dynamic files, since the connector's first replication runs until EOD is reached). Scheduler behaviour with respect to `eod_time_sec`: 1. `restart_time_sec: 10` and `eod_time_sec: 104400` (5 A.M. converted to seconds, `(24 + 5) * 3600`): the source connector will stop at 5 A.M. 2. `restart_time_sec: 10` and `eod_time_sec: -1`: the source connector will keep running. 3. `restart_time_sec: 10` and `eod_time_sec: 0`: the source connector will exit at midnight. 4. `restart_time_sec: 0` and `eod_time_sec: -1`: the source connector will exit after the first replication.

General Configuration

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 the source connector's state information. Only used when file_to_events is enabled. The file should use the .cfg format (for example, <path>/source_connector_state_backup.cfg). The connector will create the file if it doesn't exist.
general.max_files_allowed	int	any	99999	Limits the number of files in a replication. The connector will consider files for replication until the limit is reached.
general.clear_state_on_eod	int	0, 1	0	The connector preserves the last successful file state in the backup file so that it resumes from the checkpoint on the next restart. If set to 1, the connector will clear the file state in the backup file when the configured End of Day time is reached. Set to 0 if the file state does not need to be cleared.

Config Option	Type	Valid Values	Default Value	Description
general.eod_time_sec	int	-1, 0, any number > 0	-1	The connector will exit once the configured EOD has elapsed. The formula to calculate EOD is (24 + hour at which the connector should exit, in 24-hour format) * 3600. For example, to exit every day at 5 A.M., set EOD to (24 + 5) * 3600 = 104400. Set to 0 if the connector needs to exit at midnight every day, and -1 to disable this check.

Data Events Configuration

Config Option	Type	Valid Values	Default Value	Description
data_events.connectorStartEvent	boolean	true, false	false	If set to true, an additional event with the header EVENT_TYPE -> "START" will be published to the base destination topic once a new location path is processed. Additional headers included with data events are as 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 the header `EVENT_TYPE -> "COMPLETE"` will be published to the base destination topic once all files in the configured location path are completely processed. Additional headers included with data events are as 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
`data_events.fileStartEvent`	`boolean`	`true`, `false`	`false`	If set to `true`, an additional event with the header `EVENT_TYPE -> "FILE_START"` will be published to the base destination topic once processing of a new file starts. Additional headers included with data events are as 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.fileCompleteEvent`	`boolean`	`true`, `false`	`false`	If set to `true`, an additional event with the header `EVENT_TYPE -> "FILE_COMPLETE"` will be published to the base destination topic once the file is completely processed. Additional headers included with data events are as 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

Advanced Configuration

Config Option	Type	Valid Values	Default Value	Description
advanced.fileFormat	int	1, 2, 3, 4, 5, 6, 11, 12	1	• 1 = Line-by-line CSV file streaming (configure advanced.delimited) • 2 = Delimited lines file streaming. The delimiter can be defined in the eventDelimiter parameter (configure advanced.delimited) • 3 = JSON file streaming • 4 = XML file streaming • 5 = Fixed-length file (configure advanced.fixed) • 6 = Line events file. Streams the entire line as an event. A custom line separator or eventDelimiter can be configured; the default is a new line (configure advanced.line) • 11 = Line-by-line CSV file streaming, version 2 (configure advanced.delimited) • 12 = Delimited lines file streaming. The delimiter can be defined in the eventDelimiter parameter (configure advanced.delimited)

Config Option	Type	Valid Values	Default Value	Description
advanced.dynamicTopic	String	any	empty	For fileFormat values 1 and 2, use when columnDelimiter and columnHeaderMap are defined. Adds dynamic column values of the delimited file to the topic structure. For example, if columnDelimiter is , and columnHeaderMap is employee_id,first_name,last_name,city,country, the dynamic topic can be set as solace/dynamic/topic/${column1}/${column5}/${city}. You can use column<position> or the column name defined in the header as the variable expression.
advanced.jsonPath	String	any	empty	Provide a JSONPath filter to stream JSON objects from a JSON file. Use with fileFormat value 3 (JSON file streaming).
advanced.xPath	String	any	empty	Provide an XPath filter to stream XML objects from an XML file. Use with fileFormat value 4 (XML file streaming).
advanced.postProcessFileMovePathExpr	String	any	empty	Provide a path, which can be a regular absolute path for the new file or can use dynamic expressions, as follows: `postProcessFileMovePathExpr: "/home/archived/done_${DATE(%1$td%1$tm%1$tY)}/${FNAME}"` For more detailed usage and the full list of dynamic expressions that can be used in this file move path expression, refer to Dynamic Expressions.

Lines as Events Configuration

Config Option	Type	Valid Values	Default Value	Description
`advanced.line.eventDelimiter`	`Char`	`any`	`'New line'`	Defines the delimiter used to separate events in file format `6` (entire line as event).
`advanced.line.rootObjectName`	`String`	`any`	`'Empty'`	Name of the root element, if the output needs to be enclosed in a root object. Typically used with the `JSON`, `XML` or `Text` output formats. Does not work if the output format is `raw`.

CSV/Delimited Configuration

Config Option	Type	Valid Values	Default Value	Description
`advanced.delimited.eventDelimiter`	`Char`	`any`	`'\n'`	Defines the delimiter used to separate events within a single file.
`advanced.delimited.columnDelimiter`	`String`	`any`	`','`	Specifies the character or sequence of characters used to separate columns in the file. When `fileFormat` is `1` or `2`, use a column delimiter to map the columns to headers and create a final JSON payload. Used with `columnHeaderMap`, or when `useFirstEventAsHeader` is set to `true`.

Config Option	Type	Valid Values	Default Value	Description
advanced.delimited.columnHeaderMap	String	any	empty	Specifies the column headers when they are not explicitly present in the file. Provide a header map to use dynamicTopic, or to create a JSON payload from a delimited file when sendDelimitedEventPayloadAsBinary is set to false. Example: SNO,Employee_name,Employee_address,city. To send the payload as raw binary, set sendDelimitedEventPayloadAsBinary to true.
advanced.delimited.ignoreFirstEventOfDelimitedFile	boolean	true, false	false	Determines whether the first event (row) should be ignored. When set to false, all rows are processed.
advanced.delimited.useFirstEventAsHeader	boolean	true, false	false	When set to true, the first row of the file is treated as the header that defines the column names. This map is then used to create a dynamicTopic or a JSON payload. A value for columnDelimiter must be set to use this property.
advanced.delimited.autoDetectColumnDelimiter	boolean	true, false	false	When set to true, the column delimiter of each file is detected from its header row, from autoDetectColumnDelimiterCandidates, instead of using columnDelimiter. For fileFormat values 1, 2, 11 and 12. Requires useFirstEventAsHeader set to true, and cannot be used with ignoreFirstEventOfDelimitedFile set to true. See Column Delimiter Auto Detection.

Config Option	Type	Valid Values	Default Value	Description
advanced.delimited.autoDetectColumnDelimiterCandidates	list of strings	non-empty delimiters	empty	The delimiters to try on the header row, in order. The first that splits the header into at least 2 non-empty column names is used. Required when autoDetectColumnDelimiter is true. Example: [",", ";", "|", "\t"] (a tab in double quotes).
advanced.delimited.trimColumns	boolean	true, false	false	When true, whitespace around column values is removed.
advanced.delimited.quoteCharacter	Char	any	'"'	Defines the character used for quoting values. The default is a double quote ("). Use "null" for no quote character
advanced.delimited.eventFilterExpression	String	any	empty	Filters events based on conditions on the column values in the event. For example, if one of the columns is Age, use Age < 30 to publish only the events that pass this condition. For more detailed usage, refer to List of CSV/Delimited Events Filter Expressions.
advanced.delimited.columnFilter	String	any	empty	Specifies which columns are included in the output event. Provide a columnDelimiter -separated list of the column headers to include in the payload. Example: SNO,Employee_name,Employee_address.

Config Option	Type	Valid Values	Default Value	Description
advanced.delimited.sendDelimitedEventPayloadAsBinary	boolean	true, false	true	If set to true, the event/record payload for delimited files (fileFormat values 1 and 2) will be published as binary. If set to false, the payload will be sent as JSON when advanced.delimited.columnDelimiter and advanced.delimited.columnHeaderMap are set.
advanced.delimited.outputColumnDelimiter	String	any	Same as advanced.delimited.columnDelimiter	Specifies the character or sequence of characters used to separate columns in the output. Used only when advanced.delimited.sendDelimitedEventPayloadAsBinary is set to true and columnHeaderMap or useFirstEventAsHeader is set.
advanced.delimited.parsingFailBehavior	Enum	THROW, SKIP_ROW	THROW	Behaviour if the CSV parsing for an event fails. THROW will throw an exception and the MI will shut down. SKIP_ROW will log the failure, optionally publish the event on the fail topic, and continue processing.
advanced.delimited.parsingFailPublishTopic	String	any	null	Specifies the topic to which all events that fail CSV parsing will be published. Works when parsingFailBehavior is set to SKIP_ROW.
advanced.delimited.eventFilterFailPublishTopic	String	any	null	Specifies the topic to which all events that do not pass the event filter will be published.
advanced.delimited.regexFailPublishTopic	String	any	null	Specifies the topic to which all events that fail a column's regex validation will be published. Works only with fileFormat: 11 (CSV parsing V2).

Directory Wildcard Configuration

Config Option	Type	Valid Values	Default Value	Description
directory_wildcard.wildcard_type	int	0, 1, 2	0	Set this property to apply a regular expression to files: • 0 = Disabled • 1 = Whitelist files or directories • 2 = Blacklist files or directories
directory_wildcard.config_path	string	any	empty	Absolute path of the file containing the regular expression. The file should be in .cfg format.
directory_replication.start_time	int	-1, 0, any number > 0	-1	Filters files in the directory based on modified time: • -1 will use the timestamp in the checkpoint if available, or will replicate all files again. • 0 will override the timestamp in the checkpoint and consider files modified since midnight. • Any other value (epoch time) will consider files modified after that epoch time.

LVQ Configuration

Config Option	Type	Valid Values	Default Value	Description
solace_out.lvq	string	any	empty	The LVQ name configured on the Solace broker. The 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. The connector will publish checkpoint information here, and the base topic will be the same as the output destination topic configured in the connection details section. The LVQ topic is built as follows: 1. For static or directory replication, the connector will append `/checkpoint` to the base topic (`<output-destination-topic>`) and publish the data. The LVQ created on the Solace broker should have the subscription `<output-destination-topic>/checkpoint`. 2. For dynamic files, the connector will not append `/checkpoint` to the base topic (`<output-destination-topic>`). The LVQ created on the Solace broker should have the subscription `<output-destination-topic>`.

Dynamic Expressions

Introduction

The **Dynamic Expressions Resolver** allows configuration files, file paths, topic names, and event payloads to include **dynamic, self-resolving placeholders**. At runtime, these placeholders are replaced with values derived from context — such as date/time, file attributes, topics, headers, or user-provided properties.

This makes configurations adaptive without manual edits, enabling automation in file naming, routing, and message enrichment.

Example

```
input.path = /data/incoming/${DATE(%1$tY%1$tm%1$td)}/input.csv
output.file = ${FILE(FBASE)}_${DATE[+1d](%1$tY%1$tm%1$td)}.done
encoded.file = #ENCODE_BASE64(${FILE(FBASE)}_${DATE(%1$tY%1$tm%1$td)})
```

Resolved example:

Input Expression	Resolved Value
`/data/incoming/${DATE(%1$tY%1$tm%1$td)}/input.csv`	`/data/incoming/20251004/input.csv`
`${FILE(FBASE)}_${DATE[+1d](%1$tY%1$tm%1$td)}.done`	`orders_20251005.done`
`#ENCODE_BASE64(${FILE(FBASE)}_${DATE(%1$tY%1$tm%1$td)})`	`b3JkZXJzXzIwMjUxMDA0`

Expression Syntax

Every **Level 1 expression** follows this general format:

```
${CATEGORY[OFFSET](PARAM)}
```

Component	Description	Example
`CATEGORY`	Expression type — e.g., `DATE`, `FILE`, `TOPIC`	`${DATE}`, `${FILE(FNAME)}`
`[OFFSET]`	(Optional) For date/time adjustments	`${DATE[+1d]}` adds one day
`(PARAM)`	(Optional) Format or key parameter	`${DATE(%1$tY-%1$tm-%1$td)}`

A **Level 2 operator** can then wrap the entire Level 1 expression:

```
#FUNCTION(${CATEGORY(args...)})
```

Important: - Only one Level 2 operator is allowed per expression (no chaining). - The Level 1 expression must be enclosed in `${…}` inside the Level 2 wrapper.

DATE Expressions

The `DATE` expression inserts a formatted timestamp. It supports arithmetic offsets and uses `java.util.Formatter` syntax.

Offset Options

Offset	Meaning	Example
`[+3d]`	Add 3 days	`${DATE[+3d](%1$tY-%1$tm-%1$td)}`
`[-2h]`	Subtract 2 hours	`${DATE[-2h](%1$tH:%1$tM)}`
`[+1w]`	Add 1 week	`${DATE[+1w](%1$tY%1$tm%1$td)}`
`[+15m]`	Add 15 minutes	`${DATE[+15m](%1$tH:%1$tM)}`

Examples

Expression	Output Example
`${DATE(%1$tY-%1$tm-%1$td)}`	`2025-10-04`
`${DATE[+1d](%1$tY%1$tm%1$td_%1$tH%1$tM)}`	`20251005_1425`
`${DATE[-1w](Week_%1$tU)}`	`Week_39`

FILE Expressions

The `FILE` category extracts metadata from file names or paths. Useful for generating output files, logs, or routing keys dynamically.

Parameters

Parameter	Description	Example	Output
`FNAME`	File name with extension	`${FILE(FNAME)}`	`orders.csv`
`FBASE`	File base name (no extension)	`${FILE(FBASE)}`	`orders`
`FEXT`	File extension	`${FILE(FEXT)}`	`csv`
`FPATH_A`	Absolute path	`${FILE(FPATH_A)}`	`/data/in/orders.csv`

FPATH_R	Relative path	${FILE(FPATH_R)}	incoming/orders.csv
FPATH_P	Parent folder path	${FILE(FPATH_P)}	/data/in
FNAME_P	Parent folder name	${FILE(FNAME_P)}	incoming

Case Variants

- ${FNAME_U} → ORDERS.CSV
- ${FBASE_L} → orders
- ${FEXT_U} → CSV

Examples

Expression	Output
${FILE(FBASE)}_${DATE(%1$td%1$tm%1$tY)}.done	orders_04102025.done
${FILE(FEXT_U)}	CSV
${FILE(FPATH_P)}	/mnt/data/input

TOPIC Expressions

Extract topic hierarchy components by index.

Topic	Expression	Result
acme/sales/canada/order/created	${TOPIC(1)}	acme
acme/sales/canada/order/created	${TOPIC(3)}	canada
acme/sales/canada/order/created	${TOPIC(-1)}	created

UMAP and HEADER Expressions

Reference user-defined properties or message headers.

Property Map	Expression	Output
{ "region": "NA", "type": "invoice" }	${UMAP(region)}	NA
{ "Content-Type": "application/json" }	${HEADER(Content-Type)}	application/json

Legacy Placeholders

Legacy forms remain supported for backward compatibility.

Legacy	Equivalent	Example
`${fname}`	`${FILE(FNAME)}`	`data.csv`
`${fpath_r}`	`${FILE(FPATH_R)}`	`incoming/data.csv`
`${fext_u}`	`${FILE(FEXT_U)}`	`CSV`
`${fpath_p}`	`${FILE(FPATH_P)}`	`/data/incoming`

Level 2 Operators (Post-Processing Tools)

Level 2 operators transform the **result of a single Level 1 expression**. They are always used in this form:

```
#FUNCTION(${EXPR})
```

Available Operators

Operator	Description	Example	Output Example
`ENCODE_BASE64`	Base64-encode the result	`#ENCODE_BASE64(${DATE(%1$tY-%1$tm-%1$td)})`	`MjAyNS0xMC0wNA==`
`DECODE_BASE64`	Decode Base64 text	`#DECODE_BASE64(${UMAP(encodedValue)})`	Decoded text
`ENCODE_URL`	URL-encode the result	`#ENCODE_URL(${FILE(FNAME)})`	`orders.csv` → `orders.csv`
`DECODE_URL`	Decode URL text	`#DECODE_URL(${UMAP(urlValue)})`	Decoded string
`MD5`	MD5 hash	`#MD5(${FILE(FBASE)})`	`e1cbb0c3879af8347246f12c559a86b5`
`SHA1`	SHA-1 hash	`#SHA1(${DATE(%1$tY)})`	`d3486ae9136e7856bc42212385ea797094475802`
`SHA256`	SHA-256 hash	`#SHA256(${FILE(FNAME)})`	`b94d27b9934d3e08…`
`SUBSTR(start | end)`	Extract substring (end optional)	`#SUBSTR(${FILE(FBASE)}	0	3)`	`ord`	`UPPER`
Uppercase conversion	`#UPPER(${FILE(FBASE)})`	`ORDERS`	`LOWER`
Lowercase conversion	`#LOWER(${FILE(FBASE)})`	`orders`	`UUID`
Generate a random UUID (ignores inner expr)	`#UUID(${ANYTHING})`	`b7e9a84d-f23a-4e4a-9cc3-ec7f27cd3f65`	`RANDOM(min

max)`	Generate random integer	`#RANDOM(${DATE(%1$tY)}	10
99)`	45	TRIM	Trim spaces
#TRIM(${UMAP(name)})	Deepak	`REPLACE(target	replacement)`
Replace substring	`#REPLACE(${FILE(FBASE)}	ord	buy)`

Example Scenarios

Scenario	Expression	Output
Base64 encode timestamp	#ENCODE_BASE64(${DATE(%1$tY-%1$tm-%1$td)})	MjAyNS0xMC0wNA==
Create MD5 of file base	#MD5(${FILE(FBASE)})	e1cbb0c3879af8347246f12c559a86b5
Convert topic region to upper case	#UPPER(${TOPIC(2)})	SALES
Random suffix for file	`${FILE(FBASE)}_#RANDOM(${DATE(%1$tY)}	100
999)`	orders_457	Replace substring in name
`#REPLACE(${FILE(FBASE)}	ord	sale)`
sales	Trim header value	#TRIM(${HEADER(User-Name)})

Troubleshooting and Best Practices

Issue	Possible Cause	Resolution
Expression not replaced	Unsupported syntax or null context	Verify ${…} and pass context parameters
Level 2 not applied	Missing ${} inside #FUNCTION()	Use #FUNC(${EXPR}) format
Wrong substring result	Index out of range	Ensure valid positions for SUBSTR
Unexpected Base64 result	Expression already encoded	Check if data is pre-encoded

Summary

Dynamic Expressions provide a powerful, declarative way to build adaptable configuration strings. With Level 1 categories (DATE, FILE, TOPIC, etc.) and Level 2 post-processors (ENCODE_BASE64, MD5, UPPER, etc.), users can construct dynamic paths, identifiers, and names **without modifying code or restarting systems**. They are especially useful for operations teams automating event file naming,

topic routing, and integration workflows.

Date Format Specifiers

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

1. Dynamic Topic the source MI publishes events on
2. Source Paths from where the files will be read
3. Output binding destination. (Destination File Path) `spring.cloud.stream.bindings.output-<>.destination`
4. Advanced option `advanced.emptyEventsSubstitute`
5. File Pre Processors defined under configuration pre_process.<> - `prependToFile`, `prependToFirstEvent`, `prependToEvents`
6. 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

solace.

Character	Specifier %1$t<c>	Description	Example
k	%1$tk	Two-digit hour (without leading zeroes), between 0 and 23	18
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$tp	Lowercase morning or afternoon marker	pm
z	%1$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"`

CSV/Delimited Events Filter Expressions

This Micro Integration provides various expressions for filtering data based on different conditions that can be used in the delimited eventFilterExpression Below is a categorized list of expressions you can use in your Java application.

1. Basic Comparisons

Expression	Description
`age > 30`	Matches rows where age is greater than 30
`salary < 50000`	Matches rows where salary is less than 50,000
`status == "active"`	Matches rows where the status is "active"
`age >= 18 && age <= 60`	Matches rows where age is between 18 and 60
`name == "Alice"`	Matches rows where the name is "Alice"
`isPremiumMember == true`	Matches rows where the user is a premium member

2. Logical Operators

Expression	Description
`age > 30 && salary > 50000`	Matches rows where age is greater than 30 and salary is above 50,000
`status == "active"		
status == "pending"`	Matches rows where status is "active" or "pending"
`!(age < 18)`	Matches rows where age is NOT less than 18
`(age > 25		
salary > 40000) && status == "active"`	Complex conditions with multiple logical operations

3. String Operations

Expression	Description
`name.startsWith("A")`	Matches rows where the name starts with "A"
`name.endsWith("son")`	Matches rows where the name ends with "son"
`name.contains("Smith")`	Matches rows where the name contains "Smith"
`name.length() > 5`	Matches rows where the name has more than 5 characters
`name.toUpperCase() == "JOHN"`	Matches rows where the uppercase version of name is "JOHN"
`name.toLowerCase().contains("admin")`	Case-insensitive check if the name contains "admin"

solace.

4. Numeric Operations

Expression	Description
`(salary * 1.1) > 55000`	Matches rows where a 10% salary increase exceeds 55,000
`(age + 5) > 30`	Matches rows where age + 5 is greater than 30
`age % 2 == 0`	Matches rows where age is even
`Math.sqrt(salary) > 200`	Uses Math functions (square root of salary is greater than 200)

5. Null and Empty Checks

Expression	Description
`name != null`	Matches rows where name is not null
`name == null`	Matches rows where name is null
`name != null && name != ""`	Matches rows where name is not null and not empty
`status.isEmpty()`	Matches rows where the status field is empty

6. Date Operations

Expression	Description
`registrationDate.after("2023-01-01")`	Matches rows where registration date is after Jan 1, 2023
`registrationDate.before("2022-12-31")`	Matches rows where registration date is before Dec 31, 2022
`registrationDate.year == 2024`	Matches rows where registration happened in 2024
`registrationDate.month == 5`	Matches rows where registration happened in May

7. List and Collection Filtering

Expression	Description
`roles.contains("admin")`	Matches rows where the roles list contains "admin"
`tags.size() > 3`	Matches rows where the tags list has more than 3 elements
`categories.contains("electronics") && price > 100`	Matches rows where categories include "electronics" and price is greater than 100

8. Using Column Numbers Instead of Headers

If headers are not available, use column indices instead.

Expression	Equivalent With Headers
`column3 > 30`	`age > 30`

solace.

Expression	Equivalent With Headers
`column4 == "active"`	`status == "active"`
`column2.startsWith("A")`	`name.startsWith("A")`
`column5 * 1.1 > 50000`	`salary * 1.1 > 50000`

9. Regex Matching

Expression	Description
`name ~= "^[A-Z].*"`	Matches names starting with an uppercase letter
`email ~= ".*@gmail.com$"`	Matches emails ending with `@gmail.com`

Source File Decryption

The Source MI can decrypt a file before it is read and streamed as events. Decryption is done by a tool the user supplies (for example a PGP jar, `gpg` or `openssl`), which the MI runs as a separate process for every matching file. The MI reads only the decrypted plaintext; everything else (LVQ checkpoints, backup state, event headers, File Mesh Manager events, post-process file move) keeps referring to the original encrypted file.

Scope

Supported	Local file system source (`sftp`, `ftp` and `gcs` disabled), file types other than DYNAMIC, all file formats.
Not supported (yet)	Remote protocols (SFTP / FTP / GCS). Remote files are streamed directly from the remote system, so decryption would need a local download first. Dynamic files (`fileType` DYNAMIC), because a file that is still being written cannot be decrypted.

The MI refuses to start when decryption is enabled together with an unsupported setting, see Startup validation.

How it works

For every file returned by the file listing, after the LVQ / backup state checks and before any file start event:

1. **Pattern check.** The file name is matched against `encrypted-file-name-pattern`. A file that does not match is read unchanged.
2. **Clean working folder.** The file's working folder `<work-dir>/<fileName>_<hash>` is deleted if it exists and created again, with owner-only permissions (`rwx------`) on POSIX file systems. `<hash>` is the first 12 hex characters of the SHA-256 of the absolute source path, so the folder name is the same on every run for the same source file.
3. **Input.** The encrypted file is hard linked into `<folder>/in/`. If a hard link is not possible (for example the work dir is on a different file system), it is copied.
4. **Config.** If `config-template-path` is set, the template is rendered into `<folder>/decrypt.properties` (owner-only, `rw-------`) with all placeholders replaced.
5. **Run.** `command` is run as a separate process, with the working folder as its current directory, the extra `environment` variables, no standard input and a timeout of `timeout-sec`.
6. **Result.** Decryption succeeded only when the command exits with code 0 **and** `<folder>/out/` contains exactly one regular file. That file is the plaintext the reader opens.
7. **Secrets removed.** Whatever the outcome, the rendered config and the `in/` copy are deleted as soon as the command has finished.
8. **Read.** The file is read and published exactly as an unencrypted file would be. The readers still receive the original source path, so `SOURCE_FILE_NAME`, dynamic topic expressions, file stats and the LVQ checkpoint all refer to the encrypted file.

9. **Cleanup.** After the file is complete (all events acknowledged), the whole working folder is removed if `delete-decrypted-file-after-processing` is `true`.

By default a decryption failure stops the connector (see Failures). With `skip-file-on-failure: true` the file is skipped instead (see Skipped files).

Working folder layout

```
<work-dir>/
 └─── orders.csv.pgp_7e531a4a6c1a/        rwx------, name is stable per source file
     ├─── in/orders.csv.pgp               hard link / copy of the source file
(deleted right after the command)
     ├─── decrypt.properties              rendered config template, rw-------
(deleted right after the command)
     ├─── out/orders.csv                  the decrypted plaintext that is read
     └─── ...                             anything else the tool creates (e.g. its
own temp folders)
```

Only regular files directly inside `out/` count. The output file name does not matter, so tools that pick their own output name (for example from the name stored inside the PGP data) work without any extra configuration.

Configuration

Decryption is configured under `file-events.source.decryption`. When the source binding uses per-binding properties (`general.mi-id` set on the binding), use `spring.cloud.stream.file-events.bindings.<binding-name>.consumer.decryption` instead; the keys are the same.

Property	Type	Default	Description
`enabled`	boolean	`false`	Decrypt matching files before processing.
`delete-decrypted-file-after-processing`	boolean	`true`	Remove the file's working folder, including the plaintext, once the file has been fully processed. Only applies when `enabled` is `true`. With `false`, the plaintext stays in `out/` until the same source file is decrypted again.
`cleanup-on-application-start`	boolean	`false`	Application start cleanup. When `true`, all working folders left in `work-dir` by earlier runs are removed once when the MI starts, see Application start cleanup. Requires a `work-dir` that no other MI uses.
`skip-file-on-failure`	boolean	`false`	What happens to a file that fails to decrypt. `false`: the MI stops, see Failures. `true`: the file is skipped with a warning and the MI continues with the next file, see Skipped files.

Property	Type	Default	Description
encrypted-file-name-pattern	string	*	* or empty: every file is decrypted. Anything else is a Java regular expression that must match the whole file name (not the path), for example (?i).*\.(pgp|gpg). Files that do not match are read unchanged.
command	list of strings	*none, required*	The decryption command, one list entry per argument. It is **not** run through a shell, so no quoting, pipes or &&. Placeholders are replaced in every argument.
config-template-path	string	empty	Optional template rendered per file into ${DECRYPT_CONFIG_FILE}, for tools that take a config file. Must be readable at startup.
work-dir	string	*none, required*	Folder for the per-file working folders. Created if missing. Must be writable by the MI user; plaintext is written here, so it should not be shared or world-readable.
timeout-sec	int	300	Maximum run time of the command per file. The process is killed when exceeded. Must be greater than 0.
environment	map	empty	Extra environment variables for the command, for example a passphrase for tools that read one from the environment. They also override the MI's own environment for ${DECRYPT_ENV_<NAME>} placeholders.

Placeholders

Placeholders can be used in command arguments and in the config template. They are uppercase and prefixed with DECRYPT_ (like the existing FNAME / FPATH_A expressions) so they never clash with environment variables or Spring properties. Spring leaves them untouched when binding the configuration.

Placeholder	Value
${DECRYPT_WORK_DIR}	This file's working folder.
${DECRYPT_IN_DIR}	<folder>/in
${DECRYPT_OUT_DIR}	<folder>/out
${DECRYPT_IN_DIR_NAME}	in, the name only. For tools that join a base folder and a relative folder name themselves.
${DECRYPT_OUT_DIR_NAME}	out, the name only.
${DECRYPT_IN_FILE}	The encrypted file inside in/.

Placeholder	Value
${DECRYPT_OUT_FILE}	out/ + the file name without its last extension (orders.csv.pgp → out/orders.csv). For tools that need an explicit output file, such as gpg --output.
${DECRYPT_FILE_NAME}	The encrypted file's name, e.g. orders.csv.pgp.
${DECRYPT_CONFIG_FILE}	The rendered config template.
${DECRYPT_JAVA_BIN}	The java executable of the JVM running the MI. Use it to run Java based tools without depending on PATH.
${DECRYPT_ENV_<NAME>}	The value of environment variable <NAME>: the MI's own environment, overridden by decryption.environment. Use it to keep secrets such as passphrases out of the template file, e.g. recipientpassphrase=${DECRYPT_ENV_PGP_PASSPHRASE}.

All values are worked out per file by the MI; the user never defines them. ${DECRYPT_ENV_<NAME>} placeholders are replaced after all others, in a single pass, so a substituted value is never expanded again. A referenced variable that is not set stops the MI at startup.

All paths use / as separator, on Windows too. Java and most tools accept it, and unlike \ it is not an escape character in .properties files.

How the placeholder values are derived

The user never sets any DECRYPT_ value. They are all computed by the MI (CommandFileDecryptor) for every file, just before the tool runs, from three inputs.

Inputs

Input	Where it comes from
Source path	The absolute path of the encrypted file as returned by the file listing of the configured source path, e.g. /data/in/invoices_20261005.csv.pgp. It is made absolute and normalised (. and .. removed) before anything is derived from it.
work-dir	decryption.work-dir, made absolute and normalised once at startup. A relative value is resolved against the current directory of the MI process, so an absolute path is recommended.
Environment	The MI process environment (System.getenv()), with decryption.environment entries added on top (same name: decryption.environment wins). Read once at startup. Only used for ${DECRYPT_ENV_<NAME>}.
JVM	The java.home system property of the JVM running the MI. Only used for ${DECRYPT_JAVA_BIN}.

Derivation, step by step

1. **Per-file working folder** (${DECRYPT_WORK_DIR})

`<work-dir>/<safeName>_<hash>`

- `safeName`: the source file name (not the path) with every character other than `A-Z a-z 0-9 . _ -` replaced by `_`, so the folder name is valid on every file system.
- `hash`: the first 12 hexadecimal characters of the SHA-256 of the absolute source path (UTF-8). It keeps files with the same name in different source folders apart, and it is the same on every run for the same source path, which is what lets a restart clear the folder an interrupted run left behind.

To find the folder of a given file on the MI host:

```
printf '%s' '/data/in/invoices_20261005.csv.pgp' | sha256sum | cut -c1-12
# 77999aafeec5  ->  <work-dir>/invoices_20261005.csv.pgp_77999aafeec5
```

(`shasum -a 256` instead of `sha256sum` on macOS.) The path must be exactly the absolute source path, without a trailing newline.

2. **Fixed sub-folders and files** inside it. The names are fixed in the MI and cannot be configured:

`${DECRYPT_IN_DIR}`	`${DECRYPT_WORK_DIR}` + `/in`
`${DECRYPT_OUT_DIR}`	`${DECRYPT_WORK_DIR}` + `/out`
`${DECRYPT_IN_DIR_NAME}`	the literal name `in`
`${DECRYPT_OUT_DIR_NAME}`	the literal name `out`
`${DECRYPT_CONFIG_FILE}`	`${DECRYPT_WORK_DIR}` + `/decrypt.properties` (always this name, whatever the template is called)

`${DECRYPT_IN_DIR}` and `${DECRYPT_WORK_DIR}/in` give exactly the same value; the `_NAME` variants exist for tools that join a base folder and a relative folder name themselves, like the custom PGP jar below (`path` + `inputDir`).

3. **File names**

`${DECRYPT_FILE_NAME}`	The source file name, unchanged (not the `safeName`), e.g. `orders.csv.pgp`.
`${DECRYPT_IN_FILE}`	`${DECRYPT_IN_DIR}` + `/` + `${DECRYPT_FILE_NAME}`. This is the hard link (or copy) of the source file.
`${DECRYPT_OUT_FILE}`	`${DECRYPT_OUT_DIR}` + `/` + the file name without its last extension: everything before the last `.` (`orders.csv.pgp` → `orders.csv`). A name without a `.`, or whose only `.` is the first character (`.hidden`), is used unchanged. Only a suggestion for tools that need an explicit output file; the MI accepts any single file the tool writes to `out/`.

4. **Java executable** (`${DECRYPT_JAVA_BIN}`): `<java.home>/bin/java`, or `<java.home>/bin/java.exe` on Windows. It is the same for every file.

5. **Environment values** (${DECRYPT_ENV_<NAME>}): the value of <NAME> from the environment described above. <NAME> must start with a letter or _ and contain only letters, digits and _ (${DECRYPT_ENV_PGP_PASSPHRASE} → variable PGP_PASSPHRASE). Names are case sensitive, as environment variables are on Linux.
6. **Path format**: every path value is converted to text with \ replaced by /.

When and where they are replaced

For each file, after the working folder has been cleared and in/ / out/ created:

1. the values above are computed,
2. if config-template-path is set, the template's full text has every placeholder replaced and is written to ${DECRYPT_CONFIG_FILE} (owner-only),
3. every command argument has every placeholder replaced, argument by argument,
4. the command is run.

Replacement is plain text substitution, not an expression language:

- the fixed placeholders are replaced first, in the order of the tables above. None of them is a prefix of another (the closing } differs), so the order cannot change the result,
- ${DECRYPT_ENV_<NAME>} placeholders are replaced last, in a single pass, so a value that happens to contain ${DECRYPT_...} text is never expanded again,
- any other ${...} text is left exactly as written.

Spring Boot does not resolve these placeholders when it binds application.yml: it leaves placeholders it cannot resolve unchanged, and the DECRYPT_ prefix makes a clash with a real property or environment variable unlikely.

Values known at startup

Only ${DECRYPT_JAVA_BIN} and ${DECRYPT_ENV_<NAME>} do not depend on a file, so only they are resolved for the startup checks (environment variables set, program and -jar jar exist). Arguments containing any other DECRYPT_ placeholder are not checked at startup.

Worked example

Source file /data/in/invoices_20261005.csv.pgp, work-dir: /opt/decrypt-work, MI running on /opt/java, environment PGP_PASSPHRASE=⋯:

Placeholder	Value
${DECRYPT_WORK_DIR}	/opt/decrypt-work/invoices_20261005.csv.pgp_77999aafeec5
${DECRYPT_IN_DIR}	/opt/decrypt-work/invoices_20261005.csv.pgp_77999aafeec5/in
${DECRYPT_OUT_DIR}	/opt/decrypt-work/invoices_20261005.csv.pgp_77999aafeec5/out
${DECRYPT_IN_DIR_NAME}	in
${DECRYPT_OUT_DIR_NAME}	out

Placeholder	Value
${DECRYPT_IN_FILE}	…/invoices_20261005.csv.pgp_77999aafeec5/in/invoices_20261005.csv.pgp
${DECRYPT_OUT_FILE}	…/invoices_20261005.csv.pgp_77999aafeec5/out/invoices_20261005.csv
${DECRYPT_FILE_NAME}	invoices_20261005.csv.pgp
${DECRYPT_CONFIG_FILE}	…/invoices_20261005.csv.pgp_77999aafeec5/decrypt.properties
${DECRYPT_JAVA_BIN}	/opt/java/bin/java
${DECRYPT_ENV_PGP_PASSPHRASE}	the value of PGP_PASSPHRASE

A file name with special characters only changes safeName: /data/in/My Orders (v2).csv.pgp gets the folder <work-dir>/My_Orders__v2_.csv.pgp_43fa5c92669e, while ${DECRYPT_FILE_NAME} and ${DECRYPT_IN_FILE} keep the original name My Orders (v2).csv.pgp.

Startup validation

When enabled is true, the MI logs FA_FILE_DECRYPTION_CONFIG_ERROR and stops if:

- SFTP, FTP or GCS is enabled,
- fileType is DYNAMIC,
- command is empty or its first entry is blank,
- work-dir is empty,
- timeout-sec is 0 or less,
- config-template-path is set but not readable,
- encrypted-file-name-pattern is not a valid regular expression,
- an environment variable referenced with ${DECRYPT_ENV_<NAME>} in command or the template is not set,
- the program (first command entry, with ${DECRYPT_JAVA_BIN} resolved) is an absolute path that does not exist or is not executable. Relative programs such as gpg (found through PATH) are not checked,
- the entry after -jar in command is an absolute path that does not exist or is not readable,
- cleanup-on-application-start is true and work-dir is locked by another MI, or the cleanup itself fails.

It logs FA_FILE_DECRYPTION_CONFIG_WARNING (and starts) if the config template contains no ${DECRYPT_...} placeholder, because such a template cannot point the tool at the per-file working folder.

Failures

Decryption fails when the command:

- exits with a non-zero code,
- exceeds `timeout-sec` (it is killed),
- exits with 0 but leaves zero or more than one file in `out/`. This catches tools that report failures only in their output and still exit with 0. With zero files the message is "Decryption command exited with code 0 but wrote no file to … The tool may have skipped or filtered the input, see the decryption tool output logged above".

Failures to prepare the working folder (permissions, disk full, missing template) are decryption failures too.

On a failure, with `skip-file-on-failure: false` (the default), the MI:

1. logs `FA_FILE_DECRYPTION_ERROR` with the reason, followed by the last 50 lines of the tool's output,
2. logs `FA_FILE_STATS_PARTIAL` for the file,
3. sends a File Mesh Manager error event with reason code `1010` / title `FA_FILE_DECRYPTION_ERROR` (when File Mesh Manager is enabled),
4. stops the connector (exit code 1).

No file start event is sent for a file that failed to decrypt, because decryption runs before it. The working folder is left as is for investigation (without the rendered config and `in/`) and is cleared on the next attempt.

Skipped files

With `skip-file-on-failure: true`, a file that fails to decrypt is skipped and the MI continues with the next file:

1. logs `FA_FILE_DECRYPTION_SKIPPED` (WARN) with the file and the reason; the last 50 lines of the tool's output are logged before it, as for a failure,
2. removes the file's working folder, so no partial output stays behind,
3. sends a File Mesh Manager **warning** event with reason code `1012` / title `FA_FILE_DECRYPTION_SKIPPED` (when File Mesh Manager is enabled).

The skipped file has no events, no file start or file complete event, and is not counted in the totals of the run. It is not moved by `postProcessFileMovePathExpr`, so it stays in the source folder.

For directory sources (`file_type` 3 and 4) the file is not tried again in the next runs unless it is modified, for example uploaded again after the cause was fixed. A static file (`file_type` 1) is tried again in every run.

A file is not skipped when the MI is stopping while the decryption command runs.

Restart and resume

If the MI stops while a decrypted file is being processed, the file is decrypted again when it is resumed: the working folder is always cleared first, so leftovers of the interrupted run (partial or stale output) are never used.

Resume itself is unchanged. The LVQ checkpoint stores the event number and the offset into the **plaintext**; because decrypting the same file again produces the same plaintext, the reader continues exactly where it stopped. Because the working folder name is derived from the source path, the restart clears the same folder the interrupted run used.

Files the backup state reports as already completed are not decrypted again.

Application start cleanup

A working folder is normally cleared when the same source file is decrypted again. If the MI stops and that file is never processed again (for example an operator archived it), its plaintext would stay in `work-dir`. With `cleanup-on-application-start: true` (default `false`) the MI therefore, once at startup:

1. takes an exclusive lock on `work-dir/.decrypt-work.lock`, held until the MI stops. If another MI holds it, startup fails with "decryption.work-dir … is in use by another MI", so the cleanup can never remove the folders of another running MI,
2. removes every direct sub-folder whose name matches the MI's working folder pattern (`<name>_<12 hex characters>`), without following symbolic links. The lock file and anything else in `work-dir` are left untouched,
3. logs `FA_FILE_DECRYPTION | Application start cleanup: removed N working folder(s) from …`.

It runs once per application start, not on scheduled re-runs. It is independent of `delete-decrypted-file-after-processing`: with that set to `false`, decrypted files kept from earlier runs are removed at the next start. The lock is advisory and not reliable on some network file systems (e.g. NFS), so use a local disk for `work-dir`.

Security

- The plaintext only exists inside the file's working folder, which is owner-only on POSIX file systems, and is removed after processing by default.
- The rendered config (often containing passphrases) is created owner-only and deleted as soon as the command has finished, also on failure. The template itself is managed by the user and should be protected (`chmod 600`).
- Command arguments and `environment` values are never logged; only the program (first `command` entry) is.
- Passphrases can come from environment variables through `${DECRYPT_ENV_<NAME>}`, so the template on disk holds no secret. Command arguments are visible to other users through `ps`; prefer the template or environment for secrets.
- The DEBUG log of the rendered config masks the value of every key containing `pass`, `pwd`, `secret`, `token` or `credential`, and of keys containing `key` unless they end with `path`, `dir` or `file`. Lines that are not key/value pairs (e.g. XML) are never logged, only counted.
- The MI only ever deletes inside `work-dir`, and does not follow symbolic links when deleting.
- `in/` is a hard link when possible. A tool that modifies its input file in place would therefore modify the original; PGP / GPG tools do not do this.

Logging

There is no separate decryption log file; everything goes through the MI's normal logging (and its rotation / retention configuration).

Log code	Level	When
`FA_FILE_DECRYPTION`	INFO	Decryption enabled (settings without secrets), application start cleanup, decrypting a file, command started (program only), exit code, decrypted file with size and duration, working folder removed, file read unchanged because it does not match the pattern.
`FA_FILE_DECRYPTION`	DEBUG	The rendered config of every file, values in `[]` so trailing spaces show, secret values masked (see Security).
`FA_FILE_DECRYPTION_TOOL`	DEBUG	Every line the tool writes to stdout / stderr (lines longer than 4096 characters are truncated).
`FA_FILE_DECRYPTION_ERROR`	ERROR	Decryption failed, plus the last 50 lines of the tool's output.
`FA_FILE_DECRYPTION_CONFIG_ERROR`	ERROR	Invalid decryption configuration at startup.
`FA_FILE_DECRYPTION_CONFIG_WARNING`	WARN	The config template contains no `${DECRYPT_...}` placeholder.
`FA_FILE_DECRYPTION_SKIPPED`	WARN	With `skip-file-on-failure: true`: decryption failed and the file is skipped.

To see the tool's output and the rendered config for every file, set the log level of `com.solacecoe.spring.cloud.stream.binders.file_events.util.decryption.CommandFileDecryptor` to `DEBUG`.

File Mesh Manager events

When decryption is enabled, a `decryption` object is added to the File Mesh Manager **FILE_START** (event type 11) and **FILE_COMPLETE** (event type 12) events, under `fileStateEvent`. With decryption disabled the events are unchanged.

Decrypted file

```
"fileStateEvent": {
  "...": "existing fields",
  "decryption": {
    "decrypted": true,
    "decryptedFileName": "orders_20261004.csv",
    "decryptedFileSize": 6301920,
    "decryptionTimeMs": 1840
  },
  "stats": { "...": "FILE_COMPLETE only" }
```

```
}
```

File that did not match the pattern (read unchanged)

```
"decryption": { "decrypted": false }
```

File related **error** events (reader error `1001`, reject acknowledgement `5001`, decryption failure `1010`) carry the same object under `adapterState`. For `1010` it is `{"decrypted": false}`.

A skipped file (`skip-file-on-failure: true`) sends a **warning** event (event type 5) with reason code `1012` / title `FA_FILE_DECRYPTION_SKIPPED` and the reason in `reasonDetails`.

Notes for consumers:

- `size` and the `FILE_SIZE` event header remain the size of the **encrypted** file (they come from the file listing); `decryptedFileSize` is the size of what was actually read.
- FILE_START is not sent again when a file is resumed after a restart (unchanged behaviour). FILE_COMPLETE carries the values of the latest decryption.

Example: custom PGP jar (EncryptDecrypt.jar)

This custom jar is a command line tool: `java -jar EncryptDecrypt.jar <config.properties>`. It decrypts every `.pgp` / `.gpg` / `.asc` file in `path` + `inputDir` into `path` + `outputDir`, and exits with 0 even when decryption fails. The per-file working folder takes care of all of this:

- `path=${DECRYPT_WORK_DIR}` with `inputDir=${DECRYPT_IN_DIR_NAME}` / `outputDir=${DECRYPT_OUT_DIR_NAME}` points the jar at the folder that contains only the current file,
- the jar only decrypts input files **modified within the last** `decryptionThresholdInMinutes` (its `hasRecentlyDecryptedOutput` check looks at the encrypted file's modification time). The file in `in/` keeps the source file's modification time, so the threshold must be set very high (see the template below), otherwise older files are skipped and the run fails with "wrote no file to ...",
- its output name does not matter, and a failure is detected because `out/` stays empty,
- its `temp_decrypt` folder is created inside the working folder and removed with it.

The jar is compiled for Java 8, so it runs on the MI's own JVM via `${DECRYPT_JAVA_BIN}`; no separate Java installation is needed.

Files on the MI host

```
/opt/pgp-tool/
 ├──── EncryptDecrypt.jar
 ├──── config.template.properties       <- new, see below (chmod 600)
 └──── keys/
     └──── recipient_private_key.asc
/opt/decrypt-work/                   <- work-dir, created by the MI if missing
```

MI configuration

Add to the existing Source MI configuration (`application.yml`). Everything outside `decryption` is the existing source configuration and stays as it is.

```
file-events:
  source:
    decryption:
      enabled: true
      delete-decrypted-file-after-processing: true
      encrypted-file-name-pattern: "(?i).*\\.(pgp|gpg|asc)"
      work-dir: /opt/decrypt-work
      timeout-sec: 300
      config-template-path: /opt/pgp-tool/config.template.properties
      cleanup-on-application-start: false   # true: remove leftovers of earlier runs
at startup (work-dir must not be shared)
      command:
        - "${DECRYPT_JAVA_BIN}"
        - "-jar"
        - "/opt/pgp-tool/EncryptDecrypt.jar"
        - "${DECRYPT_CONFIG_FILE}"
```

The passphrase comes from environment variable `PGP_PASSPHRASE` of the MI process (see the template below), e.g. set in the MI's service definition or start script:

```
export PGP_PASSPHRASE='<recipient passphrase>'
```

In YAML double-quoted strings `\\.` is a literal `\.` in the regular expression. The pattern matches the extensions the jar can decrypt; files with any other extension in the source path are streamed unchanged. Use `*` (or leave the property out) to decrypt every file.

config.template.properties

Copy the tool's existing `config.properties`, change the folder lines and replace the passphrase with an environment placeholder. The jar's decrypt operation reads `path`, `inputDir`, `outputDir`, `keypath`, `recipientprivateKeyPath`, `recipientpassphrase` and `decryptionThresholdInMinutes`, and joins them with `Paths.get(…)`.

```
# Rendered per file by the MI. Keep this file readable only by the MI user (chmod
600).
# (Writing the passphrase here directly instead of ${DECRYPT_ENV_...} also works.)
operation=decrypt

# Per-file working folder prepared by the MI. inputDir/outputDir must stay relative
names.
path=${DECRYPT_WORK_DIR}
```

```
inputDir=${DECRYPT_IN_DIR_NAME}
outputDir=${DECRYPT_OUT_DIR_NAME}

# Keys: keypath is absolute, the key file name is relative to it.
keypath=/opt/pgp-tool/keys
recipientprivateKeyPath=recipient_private_key.asc
# From the MI's environment, so this file holds no secret
recipientpassphrase=${DECRYPT_ENV_PGP_PASSPHRASE}

# The jar only decrypts input files modified within this many minutes. The MI hands it
# exactly one file per run, so the age filter must never exclude it: 100 years.
# Do not use values near Long.MAX_VALUE, the jar's Duration/Instant arithmetic
overflows.
decryptionThresholdInMinutes=52560000
```

`inputDir`, `outputDir` and `recipientprivateKeyPath` must be relative. The jar joins them to `path` / `keypath`, so an absolute value would be appended to it instead of replacing it.

Checklist

- The MI user can read the jar, the template and the key file, and can create / write `work-dir`.
- `PGP_PASSPHRASE` is set in the MI's environment (or in `decryption.environment`); otherwise the MI does not start.
- `work-dir` is on the same file system as the source path, so the input is hard linked instead of copied (optional, saves copying large files).
- Encrypted files arrive in the source path fully written (as with any static source file).
- To archive the encrypted file after processing, use the existing post-process file move setting of the source.

Expected log for one file

```
FA_FILE_DECRYPTION | Decrypting file: /data/in/invoices_20261005.csv.pgp Working
folder: /opt/decrypt-work/invoices_20261005.csv.pgp_77999aafeec5
FA_FILE_DECRYPTION | Running decryption command: /opt/java/bin/java (arguments not
logged)
FA_FILE_DECRYPTION | Decryption command finished with exit code: 0
FA_FILE_DECRYPTION | Decrypted file: ... -> .../out/invoices_20261005.csv Size: ... in
... ms
FA_FILE_STATS | File: /data/in/invoices_20261005.csv.pgp recordsRead=...
eventsPublished=...
FA_FILE_DECRYPTION | Removed decryption working folder: ...
```

Example: GnuPG

No config template is needed when the tool takes everything as arguments:

```
file-events:
  source:
    decryption:
      enabled: true
      encrypted-file-name-pattern: "(?i).*\\.(pgp|gpg)"
      work-dir: /opt/fc/decrypt-work
      command: ["gpg", "--batch", "--yes", "--pinentry-mode", "loopback",
                "--passphrase-file", "/secure/gpg-passphrase",
                "--output", "${DECRYPT_OUT_FILE}", "--decrypt", "${DECRYPT_IN_FILE}"]
```

Custom decryptor class (design, not implemented)

The current implementation runs the tool as a separate process. That works with any existing tool unchanged, keeps the tool's libraries (for example its own BouncyCastle version) away from the MI's classpath, and lets the MI kill a tool that hangs. The code was nevertheless structured so that a decryptor written in Java can be plugged in later.

The interface

The adapter only depends on `FileDecryptor`:

```
public interface FileDecryptor {
    /** Whether the source file has to be decrypted. Non-matching files are read
unchanged. */
    boolean isEncrypted(String sourcePath);

    /** Decrypts the source file and returns the plaintext file to read. */
    DecryptionResult decrypt(String sourcePath) throws FileDecryptionException;

    /** Removes everything created while decrypting the source file, including the
plaintext. */
    void cleanup(String sourcePath) throws IOException;
}
```

`CommandFileDecryptor` is the only implementation today and is created in the adapter when `decryption.enabled` is `true`. Everything else in the adapter (when to decrypt, which path the reader opens, cleanup after completion, File Mesh Manager `decryption` object, stopping on `FileDecryptionException`) works on the interface, so a second implementation needs no adapter changes beyond choosing which one to create.

Intended user-facing contract

Users should not have to re-implement the working folder handling, pattern matching, cleanup and the "exactly one output file" check, all of which matter for resume and security. The plan is therefore a small user-facing interface plus a built-in `FileDecryptor` that wraps it:

```
/** Implemented by users. Must decrypt encryptedFile into a single file inside
```

```
outputDir. */
public interface CustomFileDecryptor {

    /** Called once at startup with decryption.properties from the MI configuration.
*/
    default void init(Map<String, String> properties) throws Exception { }

    /** Decrypt one file. Throw on any failure; the MI then stops the connector. */
    void decrypt(Path encryptedFile, Path outputDir) throws Exception;
}
```

```
/** Built-in adapter between the MI and a user's CustomFileDecryptor. */
public class ClassFileDecryptor implements FileDecryptor {
    // Same pattern matching, stable per-file working folder, clear-before-decrypt,
"exactly one file in out/",
    // cleanup and DecryptionResult as CommandFileDecryptor (shared via a common base
class).
    // Instead of starting a process it calls customFileDecryptor.decrypt(inFile,
outDir).
}
```

Planned configuration:

```
file-events:
  source:
    decryption:
      enabled: true
      mode: class                                    # "command" (default, current
behaviour) or "class"
      class-name: com.example.crypto.PgpDecryptor     # implements
CustomFileDecryptor, public no-arg constructor
      jar-path: /opt/decryptors/custom-decryptor.jar # loaded in its own class loader
      properties:                                     # passed to init()
        private-key-path: /secure/keys/recipient_private_key.asc
        passphrase-env: PGP_PASSPHRASE
      work-dir: /opt/fc/decrypt-work
      timeout-sec: 300
```

How it would be loaded and run:

1. At startup the adapter creates `ClassFileDecryptor` instead of `CommandFileDecryptor` when `mode` is `class`.
2. The jar at `jar-path` is loaded in a dedicated child-first `URLClassLoader`, so the user's dependency versions (for example BouncyCastle) win over, and do not leak into, the MI's own. Only the `CustomFileDecryptor` interface itself is shared from the MI's class loader.
3. `class-name` is instantiated through its public no-arg constructor and `init(properties)` is called once. A missing class, a class that does not implement the interface or an `init` failure is a startup

configuration error (`FA_FILE_DECRYPTION_CONFIG_ERROR`).

4. Per file, `decrypt` runs on a separate worker thread so that `timeout-sec` can be applied. A Java thread cannot be killed safely, so on timeout the MI stops the connector instead of continuing next to a hanging thread. This is the main reason the process based mode was implemented first.
5. Results, failures, cleanup, resume and File Mesh Manager events behave exactly as in command mode.

Requirements for a user implementation:

- Write exactly one file into `outputDir`; the name is free.
- Produce the same plaintext every time for the same input (needed for resume by offset).
- Do not modify or delete `encryptedFile` (it may be a hard link to the original).
- Do not keep state between files; files are decrypted one at a time on the same instance.

The custom `EncryptDecrypt.jar` cannot be used in class mode as it is: it exposes only static, folder based methods driven by a `Properties` object. It would need a small `CustomFileDecryptor` wrapper, which is why command mode is the right fit for it.

Implementation map

Class	Responsibility
`properties.DecryptionSettings`	`decryption.*` configuration. `toString()` leaves out command arguments and environment values.
`util.decryption.FileDecryptor`	Interface the adapter uses.
`util.decryption.CommandFileDecryptor`	Process based implementation: working folder, template rendering, running the command, output capture, result check, cleanup.
`util.decryption.DecryptionResult`	Decrypted file, size and duration; builds the File Mesh Manager `decryption` object.
`util.decryption.FileDecryptionException`	Any decryption failure.
`inbound.FileEventsInboundChannelAdapter`	Startup validation, decrypt before file start events, reader opens the decrypted file, cleanup after completion, `decryption` object in FILE_START / FILE_COMPLETE / error events, reason code `1010`.

File Streaming Modes

The `fileFormat` property defines how file content is read and streamed as events. Supported modes include:

fileFormat	Description
1	CSV Line by Line File Streaming (Version 1 - Deprecated)

solace.

fileFormat	Description
2	CSV - Custom Delimited Events File Streaming (Version 1 - Deprecated)
3	JSON File Streaming
4	XML File Streaming
5	Fixed Length File Streaming
11	CSV Line by Line File Streaming (Version 2 - Enhanced)
12	CSV - Custom Delimited Events File Streaming (Version 2 - Enhanced)

File Format 1: CSV Line by Line File Streaming (Version 1)

This mode supports **line-by-line streaming of delimited files (e.g., CSV)**, where each row is read as an event.

Events are parsed row-wise with optional transformation and filtering using a lightweight configuration.

Key Features

- Line-by-line streaming
- Supports CSV or any delimited format (e.g., pipe `|`, comma `,`, semicolon `;`)
- Column mapping using header row or explicit definition
- Event filtering with expression language
- Dynamic topic generation using column placeholders
- Column trimming and selective output

Example CSV File Content

```
1|123|John|'Doe'|25|OpenAI|Toronto|Canada|john.doe@example.com|1234567890
2|124|Jane|'Smith'|35|Google|New York|USA|jane.smith@example.com|0987654321
```

Configuration Properties

Field	Type	Allowed Values	Default	Description
`advanced.delimited.columnDelimiter`	String	Column Separator or Delimiter	","	Defines how the columns are separated in the file. Examples: ",", "|", "\t", "||" Must be provided when using `columnHeaderMap` or `useFirstEventAsHeader`.

Field	Type	Allowed Values	Default	Description
advanced.delimited.columnHeaderMap	String	Delimited string	*(empty)*	Specifies column names when the header is not in the file. Required for JSON transformation and dynamic topic generation. Example: "Index|Customer Id|First Name|Last Name|Age|Company|City|Country|Email|Phone"
advanced.delimited.ignoreFirstEventOfDelimitedFile	Boolean	true, false	false	Skips the first row of the file. Useful when the file starts with a non-data row (e.g., metadata or comment).
advanced.delimited.useFirstEventAsHeader	Boolean	true, false	false	Uses the first row in the file as the header row. If enabled, columnHeaderMap is ignored.
advanced.delimited.autoDetectColumnDelimiter	Boolean	true, false	false	Detects the column delimiter of each file from its header row, from autoDetectColumnDelimiterCandidates. Requires useFirstEventAsHeader: true; not with ignoreFirstEventOfDelimitedFile: true. See Column Delimiter Auto Detection.
advanced.delimited.autoDetectColumnDelimiterCandidates	List of Strings	Non-empty delimiters	*(empty)*	The delimiters to try, in order, e.g. [",", ";", "|", "\t"] (a tab in double quotes). Required when autoDetectColumnDelimiter is true.
advanced.delimited.trimColumns	Boolean	true, false	false	Trims whitespace around column values. Helpful for cleaner matching and filtering.
advanced.delimited.quoteCharacter	Char	Any single character	" (double quote)	Character used to quote column values. Example: 'John Doe' or "New York"
advanced.delimited.escapeCharacter	Char	Any single character	\0 (null)	Character used as an escape character
advanced.delimited.eventFilterExpression	String	Any valid expression	*(empty)*	Filters rows using expression logic. Example: "Age < 31" will allow only events where Age < 31.
advanced.delimited.columnFilter	String	Delimited string	*(empty)*	Restricts the output JSON payload to specific columns. Example: "Index|First Name|Age|Country"

Field	Type	Allowed Values	Default	Description
`advanced.delimited.parsingFailBehavior`	String	THROW, SKIP_ROW	THROW	THROW - throws error and stops processing more file data, SKIP_ROW skips the failed event and logs it
`advanced.dynamicTopic`	String	Topic template with placeholders	*(empty)*	Enables event publishing to dynamic Solace topics using column values. Example: `"file/events/${column2}/${Country}"`

Configuration Example 1 – Basic (with explicit headers)

```
file-events:
  source:
    advanced:
      fileFormat: 1
      delimited:
        columnDelimiter: "|"
        columnHeaderMap: "Index|Customer Id|First Name|Last
Name|Age|Company|City|Country|Email|Phone"
      dynamicTopic: file/events/${column2}/${Country}
```

Explanation: - The CSV rows are parsed using `|` as delimiter. - Explicit header names are provided for mapping columns. - Dynamic topic is generated using `column2` and `Country`.

Configuration Example 2 – Use first row as header

```
file-events:
  source:
    advanced:
      fileFormat: 1
      delimited:
        columnDelimiter: "|"
        useFirstEventAsHeader: true
        trimColumns: true
      dynamicTopic: file/events/${Customer Id}/${Country}
```

Explanation: - The first row of the file is used as headers. - Trimming is enabled. - Dynamic topic uses named placeholders based on the header.

Configuration Example 3 – Filtering and selective columns

```
file-events:
  source:
    advanced:
      fileFormat: 1
      delimited:
        columnDelimiter: "|"
        columnHeaderMap: "Index|Customer Id|First Name|Last
Name|Age|Company|City|Country|Email|Phone"
        eventFilterExpression: "Age < 31"
        columnFilter: "Index|First Name|Age|Country"
        quoteCharacter: "'"
        escapeCharacter: '\\'
      dynamicTopic: file/events/${column2}/${Country}
```

Explanation: - Events are filtered to only include those with Age < 31. - Only a subset of columns is included in the final event. - Quote character is customized to `'`.

Sample Input Payload

Given row:

```
1|123|John|'Doe'|25|OpenAI|Toronto|Canada|john.doe@example.com|1234567890
```

And `columnFilter: "Index|First Name|Age|Country"`

The output JSON would be:

```
{
  "Index": "1",
  "First Name": "John",
  "Age": 25,
  "Country": "Canada"
}
```

Best Practices

- Always define `columnDelimiter` explicitly to avoid ambiguity.
- Prefer `useFirstEventAsHeader` when headers are present in the file.
- Avoid this mode for complex CSV parsing – migrate to `fileFormat: 11` or `12`.
- Validate `eventFilterExpression` syntax with sample data.

- Use `columnFilter` to reduce payload size and publish only required data.

File Format 2: CSV - Custom Delimited Events File Streaming (Version 1)

The configuration for File Format 2 and 1 works in the similar way except we can define a custom event delimiter inside the configuration as below:

Configuration Properties

Field	Type	Allowed Values	Default	Description
`advanced.delimited.eventDelimiter`	Char	any	'\n'	This setting would define a delimiter for separating events. If enabled, it would likely be used to distinguish different events within a single file.

Example File with multiple rows separate by "@"

Given row:

```
1|123|John|'Doe'|25|OpenAI|Toronto|Canada|john.doe@example.com|1234567890@2|124|Jane|'
Smith'|35|Google|New York|USA|jane.smith@example.com|0987654321
```

File Format 11: CSV Line by Line (Version 2 - Enhanced)

This mode is an **enhanced replacement for the fileFormat: 1** currently a beta release, providing better validation, column typing, and expression-based filtering.

It allows advanced control over how columns are parsed, validated, and included in the output. Suitable for high-integrity CSV parsing and structured event generation.

⚙️ Configuration Properties

Field	Type	Allowed Values	Default	Description
`advanced.delimited.columnDelimiter`	String	Column Separator or Delimiter	","	Defines how the columns are separated in the file. Examples: ",", "|", "\t", "||" Must be provided when using `columnHeaderMap` or `useFirstEventAsHeader`.
`advanced.delimited.columnHeaderMap`	String	Delimited string of headers	*(empty)*	Explicit header list if headers are not in the file.
`advanced.delimited.ignoreFirstEventOfDelimitedFile`	Boolean	`true`, `false`	`false`	If true, skips the first row entirely.
`advanced.delimited.useFirstEventAsHeader`	Boolean	`true`, `false`	`false`	Treats the first row of the file as the header row.
`advanced.delimited.autoDetectColumnDelimiter`	Boolean	`true`, `false`	`false`	Detects the column delimiter of each file from its header row, from `autoDetectColumnDelimiterCandidates`. Requires `useFirstEventAsHeader: true`; not with `ignoreFirstEventOfDelimitedFile: true`. See Column Delimiter Auto Detection.
`advanced.delimited.autoDetectColumnDelimiterCandidates`	List of Strings	Non-empty delimiters	*(empty)*	The delimiters to try, in order, e.g. `[",", ";", "|", "\t"]` (a tab in double quotes). Required when `autoDetectColumnDelimiter` is `true`.
`advanced.delimited.trimColumns`	Boolean	`true`, `false`	`false`	Trims leading and trailing spaces of all column values.
`advanced.delimited.quoteCharacter`	Char	Any character	`"` (double quote)	Column values can be quoted. This sets the quote character.
`advanced.delimited.escapeCharacter`	Char	Any single character	`\0` (null)	Character used as an escape character

Field	Type	Allowed Values	Default	Description
`advanced.delimited.eventFilterExpression`	String	Expression language	*(empty)*	Allows rows to be included or excluded based on values (e.g., `Age < 31`).
`advanced.delimited.restrictToColumnSpecs`	Boolean	`true`, `false`	`false`	If true, only includes columns defined in `columnSpecs` in output.
`advanced.delimited.columnSpecs`	List	Defined column constraints	*(empty)*	Optional. Allows regex, defaulting, coercion, and validation per column.

Example Configuration – Enhanced CSV Parsing

```
file-events:
  source:
    advanced:
      fileFormat: 11
      dynamicTopic: solace/fc/source/demo/dynamic/${Country}/${City}/${param-
3}/${column5}
      delimited:
        columnDelimiter: "|"
        columnHeaderMap: Index,Customer Id,First Name,Last
Name,Age,Company,City,Country,Phone 1,Phone 2,Email,Subscription Date,Website
        ignoreFirstEventOfDelimitedFile: false
        useFirstEventAsHeader: true
        trimColumns: true
        quoteCharacter: '"'
        escapeCharacter: '\\'
        strictValidation: true
        eventFilterExpression: "Age < 31"
        restrictToColumnSpecs: false
        columnSpecs:
          - name: "First Name"
            type: STRING
            regex: "[A-Za-z]+"
            onRegexFail: SKIP_COLUMN
            defaultValue: "Unknown"
            required: OPTIONAL
          - name: "Customer Id"
            type: STRING
            required: REQUIRED
          - name: Age
            type: INT
            regex: "\\d+"
            onRegexFail: COERCE
            required: REQUIRED
          - name: Company
            type: STRING
```

```
      required: OPTIONAL
    - name: Country
      type: STRING
      required: OPTIONAL
    - name: City
      type: STRING
      required: OPTIONAL
```

Explanation: - Strongly typed configuration with validation rules and defaults - Topic is dynamically generated using column values - Filters out rows where Age >= 31 - Allows optional or required columns, validation failure handling

File Format 12: CSV - Custom Delimited Events File Streaming (Version 2 - Enhanced)

The configuration for File Format 2 and 1 works in the similar way except we can define a custom event delimiter inside the configuration as below:

Configuration Properties

Field	Type	Allowed Values	Default	Description
advanced.delimited.event Delimiter	Char	any	'\n'	This setting would define a delimiter for separating events. If enabled, it would likely be used to distinguish different events within a single file.

Example File with multiple rows separate by "@"

Given row:

```
1|123|John|'Doe'|25|OpenAI|Toronto|Canada|john.doe@example.com|1234567890@2|124|Jane|'
Smith'|35|Google|New York|USA|jane.smith@example.com|0987654321
```

□ Differences Between FileFormat 1/2 and 11/12

Feature	fileFormat: 1 & 2	fileFormat: 11 & 12
Validation	Minimal	Strict, configurable per column
Deprecation Status	Deprecated	Recommended
Header Handling	Yes	Yes + advanced mapping
Filtering	Basic	Advanced (typed, expression-based)
Column Trimming	Optional	Optional
Regex Validation	No	Yes
Column Specs	No	Yes
Quote Character	Yes	Yes
Event Filtering	Basic	Enhanced
Output Restriction	Yes (`columnFilter`)	Yes (`restrictToColumnSpecs`)
Event Delimiter	fileFormat: 2 only	fileFormat: 12 only

Column Delimiter Auto Detection

For the CSV and delimited file formats, the Source MI can detect the column delimiter of each file from its header row, instead of using the configured `columnDelimiter` for every file. This helps when the files of one source use different delimiters, for example some files with `,` and others with `;` or tab.

The MI tries a configured list of candidate delimiters, in order, on the header row of every file, and reads the file with the first one that splits the header into valid columns.

Scope

Supported	File formats `1`, `2`, `11` and `12` (CSV and delimited, version 1 and 2), all protocols (local, SFTP, FTP, GCS), all file types.
Requires	`useFirstEventAsHeader: true`: the first row of every file is its header.
Not combined with	`ignoreFirstEventOfDelimitedFile: true`. That setting means the configured `columnDelimiter` and `columnHeaderMap` apply to all files, so there is nothing to detect.

The MI refuses to start when auto detection is enabled with an unsupported setting, see Startup validation.

Configuration

Field	Type	Allowed Values	Default	Description
`advanced.delimited.autoDetectColumnDelimiter`	Boolean	`true`, `false`	`false`	Detects the column delimiter of each file from its header row. Requires `useFirstEventAsHeader: true` and cannot be used with `ignoreFirstEventOfDelimitedFile: true`.
`advanced.delimited.autoDetectColumnDelimiterCandidates`	List of Strings	Non-empty delimiters	*(empty)*	The delimiters to try, in order. Required when `autoDetectColumnDelimiter` is `true`. A delimiter can have more than one character, for example `"||"`.

```
file-events:
  source:
    advanced:
      fileFormat: 11
      delimited:
        useFirstEventAsHeader: true
        autoDetectColumnDelimiter: true
        autoDetectColumnDelimiterCandidates: [",", ";", "|", "\t"]
```

- Write a tab as `"\t"` in **double** quotes. In single quotes (`'\t'`) or without quotes, YAML keeps the two characters `\` and `t`.
- The order of the candidates matters, see How it works. Put a space candidate (`" "`), if any, last: header names often contain spaces.
- `columnDelimiter` is still used for the configuration values `columnHeaderMap` and `columnFilter`, which are split with it. It is not used for the content of the files.

How it works

For every file, before its data rows are read, the MI takes the header row and tries each candidate in the configured order. A candidate **matches** when:

- the header can be parsed with it, and
- the header splits into **at least 2 columns**, and
- **no column name is empty** (after trimming).

The header is parsed with the same parser and settings as the data rows of that file format (`quoteCharacter`, `escapeCharacter`, `trimColumns`, strict parsing for the version 2 formats). So a delimiter inside quotes does not count, and a header that is not valid with a candidate (for example unbalanced quotes) does not match it.

- **One match**: the file is read with that delimiter.
- **Several matches**: the file is read with the **first** matching candidate, and `FA_COLUMN_DELIMITER_MULTIPLE_MATCHES` is logged as a warning, naming all matching candidates.
- **No match**: the file fails, see Failures.

The delimiter is detected separately for every file, so the files of a directory source can use different delimiters. When a file is resumed after a restart, its header is read again and the delimiter detected again.

Table 1. Examples with the candidates `[",", ";", "|", "\t"]`

Header row	Delimiter used	Why
`id,name,city`	`,`	Only `,` gives 3 columns.
`id;name;city`	`;`	Only `;` gives 3 columns.
`"id";"full, name";"city"`	`;`	The `,` is inside quotes, so it is not a delimiter. With `,` the header is not valid.
`code;label,group;kind`	`,` (with a warning)	Both `,` (2 columns) and `;` (3 columns) match; `,` comes first.
`id||name`, candidates `["|", "||"]`	`||`	With `|` the middle column name would be empty.
`onlycolumn`	none, the file fails	No candidate gives 2 columns.

Header row	Delimiter used	Why
`id,,city`	none, the file fails	With `,` the middle column name is empty.

Failures

When no candidate matches the header of a file, the MI:

1. logs `FA_COLUMN_DELIMITER_NOT_DETECTED` with the file and the candidates,
2. logs `FA_FILE_STATS_PARTIAL` for the file,
3. sends a File Mesh Manager error event with reason code `1011` / title `FA_COLUMN_DELIMITER_NOT_DETECTED` (when File Mesh Manager is enabled),
4. stops the connector (exit code 1), like other file errors.

No event of that file is published, because the header is its first row.

Startup validation

When `autoDetectColumnDelimiter` is `true`, the MI logs `FA_COLUMN_DELIMITER_CONFIG_ERROR` and stops (exit code 1) if:

- `fileFormat` is not `1`, `2`, `11` or `12`,
- `useFirstEventAsHeader` is not `true` (neither `advanced.useFirstEventAsHeader` nor `advanced.delimited.useFirstEventAsHeader`),
- `ignoreFirstEventOfDelimitedFile` is `true` (`advanced.ignoreFirstEventOfDelimitedFile` or `advanced.delimited.ignoreFirstEventOfDelimitedFile`),
- `autoDetectColumnDelimiterCandidates` is empty or contains an empty value.

Logging

Log code	Level	When
`FA_COLUMN_DELIMITER`	INFO	At startup: auto detection is enabled, with the candidates.
`FA_COLUMN_DELIMITER_DETECTED`	INFO	For every file: the delimiter used.
`FA_COLUMN_DELIMITER_MULTIPLE_MATCHES`	WARN	The header matches several candidates; the first is used.
`FA_COLUMN_DELIMITER_NOT_DETECTED`	ERROR	No candidate matches the header; the file fails.
`FA_COLUMN_DELIMITER_CONFIG_ERROR`	ERROR	Invalid configuration at startup.

Delimiters are written in quotes in the logs, with a tab as `'\t'`, for example `[',', ';', '|', '\t']`.

File Mesh Manager events

When auto detection is enabled, a `columnDelimiter` object is added to the File Mesh Manager **FILE_COMPLETE** (event type 12) events, under `fileStateEvent`. With auto detection disabled the events are unchanged.

```
"fileStateEvent": {
  "...": "existing fields",
  "columnDelimiter": {
    "detected": ";",
    "matchingCandidates": [";"],
    "candidates": [",", ";", "|", "\t"]
  },
  "stats": { "...": "existing fields" }
}
```

- `detected`: the delimiter used for the file, `null` when no candidate matched.
- `matchingCandidates`: all candidates that matched the header, in configuration order. More than one means the warning above was logged.
- `candidates`: the configured candidates.

File related **error** events carry the same object under `adapterState` once the header of the file was read. For reason code `1011` it is `{"detected": null, "matchingCandidates": [], "candidates": [⋯]}`.

FILE_START events have no `columnDelimiter` object: they are sent before the file is read, so the delimiter is not known yet.

Column Specs Configuration Guide

This document explains how to configure the `columnSpecs` section used in `fileFormat: 5`, `fileFormat: 11` or `fileFormat: 12`.

Each item in `columnSpecs` defines parsing, validation, coercion, and output behavior for a specific column in your delimited file.

Configurations Explained

Setting	What It Does	Example	Mandatory	Default
`name`	Column name as it appears in the header row or `columnHeaderMap`. Used to match the column.	`"Name"`	MANDATORY	null
`length`	Optional. Maximum number of characters for this column. Used for fixed-length validation.	`10`	OPTIONAL	`0`
`type`	Declares the data type for parsing: - `STRING`: text - `INT`: whole numbers - `DOUBLE`: decimal numbers - `BOOLEAN`: true/false - `DATE`: date format	`STRING`	OPTIONAL	`STRING`
`inputDateFormat`	For `DATE` type. Format used in the incoming data file.	`"dd/MM/yyyy"` (example: 25/12/1990)	OPTIONAL	null
`outputDateFormat`	For `DATE` type. Format to be used in the output event.	`"dd-MM-yyyy"` (example: 25-12-1990)	OPTIONAL	null
`regex`	Regex pattern that the value must match. Can be used to enforce formats like alphabets only, numeric only, etc.	`"[A-Za-z]+"`	OPTIONAL	null

solace.

Setting	What It Does	Example	Mandatory	Default
`onRegexFail`	Action to take when `regex` validation fails: - `THROW`: raises an exception and fails processing - `SKIP_COLUMN`: sets value to null - `SKIP_ROW`: discards the row - `LOG`: logs the error and continues - `DEFAULT_VALUE`: uses the value from `defaultValue` - `COERCE`: attempts to fix the value (e.g., remove non-matching characters)	`"DEFAULT_VALUE"`	OPTIONAL	`THROW`
`defaultValue`	Value to use if validation fails and `onRegexFail` is set to `DEFAULT_VALUE`.	`"Unknown"`	OPTIONAL	null
`required`	Defines how the column is handled in the output: - `REQUIRED`: must appear in output and must not be null - `OPTIONAL`: may appear in output; null values allowed - `DROP`: column is processed for logic but excluded from output event	`"REQUIRED"`	OPTIONAL	`REQUIRED`

String Format (Simple)

Use this for basic setups. Each column is a single line with values separated by commas.

```
columnSpecs:
  - "Name, 10, STRING"
  - "Age, 3, INT"
  - "Balance, 10, DOUBLE"
  - "Active, 5, BOOLEAN"
  - "Birthday, 10, DATE, dd/MM/yyyy, dd-MM-yyyy"
```

What Each Part Means

1. **Name**: What to call the column (e.g., `"Name"`).
2. **Length**: How many characters the column takes up (e.g., `10`).
3. **Type**: The data type (`STRING`, `INT`, `DOUBLE`, `BOOLEAN`, `DATE`).
4. **Input Date Format** (optional, for `DATE` only): How dates are written in the file (e.g., `"dd/MM/yyyy"`).

5. **Output Date Format** (optional, for `DATE` only): How dates should look in the output (e.g., `"dd-MM-yyyy"`).

and so on.

Example File

John 25 123.45 true 25/12/1990

- `"Name, 10, STRING"`: Takes `John ` (10 characters).
- `"Age, 3, INT"`: Takes `25 ` (3 characters).
- Total row length: 38 (if `eventFixedLength: 38`).

Notes: - Use this format for simple files without complex rules. - All columns are required in the output by default, and errors stop processing if `strictValidation` is `true`.

Map Format (Detailed)

Use this for more control, like adding validation rules or excluding columns from the output. Each column is a block with named settings.

```
columnSpecs:
  - name: Name
    length: 10
    type: STRING
    regex: "[A-Za-z]+"
    onRegexFail: DEFAULT_VALUE
    defaultValue: "Unknown"
    required: REQUIRED
  - name: Age
    length: 3
    type: INT
    regex: "\\d+"
    onRegexFail: COERCE
    required: OPTIONAL
  - name: Balance
    length: 10
    type: DOUBLE
    regex: "[0-9]*\\.?[0-9]*(?:[eE][-+]?[0-9]+)?"
    onRegexFail: SKIP_ROW
    required: REQUIRED
  - name: Active
    length: 5
    type: BOOLEAN
    required: OPTIONAL
  - name: Birthday
    length: 10
    type: DATE
    inputDateFormat: "dd/MM/yyyy"
    outputDateFormat: "dd-MM-yyyy"
```

```
    required: DROP
```

Example File and Output

John 25 123.45 true 25/12/1990 Jane 2x 678.90 false 01/01/2000

- **Output (JSON)**:
- Row 1: `{"Name": "John", "Age": 25, "Balance": 123.45, "Active": true}` (`Birthday` dropped).
- Row 2: `{"Name": "Jane", "Age": 2, "Balance": 678.90, "Active": false}` (`COERCE` fixes `2x` to `2`).

Tips: - Use `regex` to enforce data quality (e.g., `"\\d+"` for digits only). - Set `required: DROP` for columns you need to process but not include in the output. - Combine `onRegexFail` and `defaultValue` for fallback values.

Fixed-Length File Processing Configuration Guide

This guide explains how to configure the application to process fixed-length files. Fixed-length files have data organized in rows where each column has a specific width, and this configuration tells the application how to read, validate, and format that data into output events (e.g., JSON, CSV). You'll use a YAML file to set up the structure, columns, and output preferences.

Overview

The configuration lives under the `file-events.source.advanced.fixedLengthFile` section of your YAML file. It controls: - How rows are split (fixed length or separator-based). - How columns are defined and validated. - How the output is formatted (e.g., JSON, XML).

You can define columns in two ways: a simple **string format** (comma-separated) or a detailed **map format** (key-value pairs). The map format offers more options, like validation rules and output control.

Sample Configuration

Here's a complete example to get you started:

```yaml
file-events:
  source:
    advanced:
      fixedLengthFile:
        isRowSeparatorPresent: false
        eventFixedLength: 51
        rowSeparator: "|"
        strictValidation: true
        columnSpecs:
          - "Name, 10, STRING"
          - "Age, 3, INT"
          - "Balance, 10, DOUBLE"
          - "Active, 5, BOOLEAN"
          - "Birthday, 10, DATE, dd/MM/yyyy, dd-MM-yyyy"
```

Let's break down each part.

File Structure Settings

These settings define how the file is split into rows.

Setting	What It Does	Example
`isRowSeparatorPresent`	Set to `true` if rows end with a separator (like `	`). Set to `false` if rows are a fixed length.
`false` (rows are 51 characters long)	`eventFixedLength`	Number of characters per row (excluding separator). Required if `isRowSeparatorPresent` is `false`.

solace.

Setting	What It Does	Example
51	rowSeparator	The character that separates rows (used only if isRowSeparatorPresent is true).
"|"	strictValidation	If true, stops processing on errors (e.g., bad data). If false, logs errors and continues with missing values.

Tips: - Use `isRowSeparatorPresent: false` for most fixed-length files and set `eventFixedLength` to match your row size. - Set `strictValidation: false` if you want to process files with occasional bad data without stopping.

Column Definitions (`columnSpecs`)

The `columnSpecs` section lists each column in your file—its name, width, type, and optional rules. You can use two formats:

Refer to Column Specifications Configuration for more details.

Filtering Rows (Optional)

Add an `eventFilterExpression` to skip rows based on conditions:

```
eventFilterExpression: "Age < 30 && Balance > 100"
```

- Example: Only processes rows where `Age` is under 30 and `Balance` exceeds 100.
- Use column names from `columnSpecs` and operators like `<`, `>`, `==`, `&&`.

Putting It All Together

Example File

John 25 123.45 true 25/12/1990 Jane 2x 678.90 false 01/01/2000

String Config

```
fixedLengthFile:
  eventFixedLength: 38
  strictValidation: true
  columnSpecs:
    - "Name, 10, STRING"
    - "Age, 3, INT"
    - "Balance, 10, DOUBLE"
```

- Row 1: `{"Name": "John", "Age": 25, "Balance": 123.45}`
- Row 2: Stops with an error (`"2x"` isn't an `INT`).

Map Config

```
file-events:
  source:
    advanced:
      fixedLengthFile:
        eventFixedLength: 38
        strictValidation: true
        columnSpecs:
          - name: Name
            length: 10
            type: STRING
            regex: "[A-Za-z]+"
            onRegexFail: DEFAULT_VALUE
            defaultValue: "Unknown"
            required: REQUIRED
          - name: Age
            length: 3
            type: INT
            regex: "\\d+"
            onRegexFail: COERCE
            required: OPTIONAL
```

- Row 1: `{"Name": "John", "Age": 25}`
- Row 2: `{"Name": "Jane", "Age": 2}` (`COERCE` fixes `"2x"`).

Troubleshooting

- **Row Length Errors**: Check `eventFixedLength` matches your file's row size (sum of column lengths).
- **Validation Failures**: Set `strictValidation: false` to log issues instead of stopping, or use `onRegexFail: COERCE`.
- **Missing Columns**: Ensure `required: REQUIRED` columns have valid data, or adjust `regex` rules.

File Format 6: Lines as Events

This mode allows the streaming of individual line as events from the configured source files

Additionally, a line separator (default: newline) can be configured if newline is not the event delimiter in the file

Configuration Properties

Field	Type	Allowed Values	Default	Description
advanced.line.eventDelimiter	Char	any	'\n'	This setting would define a delimiter for separating events. If configured, it would be used to distinguish different events within a single file.
`advanced.line.rootObjectName`	String	any	*(empty)*	If defined the event can be enclosed inside the rootObjectName, primarily used with output formats json or xml.

Example Configuration

```
file-events:
  source:
    advanced:
      fileFormat: 6
      line:
        eventDelimiter: "|"
        rootObjectName: "data"
```

Output Formatting (`output`)

This transformation options allows controlling how the output events will be streamed in case of the below file formats only -

fileFormat	Description
1	CSV Line by Line File Streaming (Version 1 - Deprecated)
2	CSV - Custom Delimited Events File Streaming (Version 1 - Deprecated)
5	Fixed Length File Streaming
11	CSV Line by Line File Streaming (Version 2 - Enhanced)
12	CSV - Custom Delimited Events File Streaming (Version 2 - Enhanced)

Controls how the processed data is formatted (e.g., JSON, CSV, XML, or text).

Setting	What It Does	Example	Default
`format`	Output format type. Valid values: `json`, `csv`, `xml`, `text`, `raw`	`"json"`	`"raw"`
`encloseInRootObjectName`	If the output needs to be enclosed in a new root object - Name of the new root element. Typically used in JSON or XML.	`"root"`	`"empty"`
`prettyPrintJson`	Adds line breaks and indentation for JSON output.	`true`	`false`
`ignoreNulls`	If `true`, omits fields with null values in JSON, XML, text.	`true`	`false`
`csvDelimiter`	Character used to separate fields in CSV.	`","`	`","`
`csvQuoteChar`	Character used to wrap field values in CSV.	`"\""`	`"\""`

solace.

Setting	What It Does	Example	Default
csvQuoteMode	Quote Mode used to wrap field values in CSV. Values allowed are as follows - ALL : Quotes all fields, regardless of content. - ALL_NON_NULL : Quotes all non-null fields. - MINIMAL : Only quotes fields when necessary, i.e., if they contain: The delimiter (e.g., comma) A quote character A newline character - NON_NUMERIC : Quotes only non-numeric fields (e.g., Strings), leaving numbers unquoted. - NONE : No fields are quoted, even if they contain special characters.	ALL	MINIMAL
csvRecordSeparator	Record Separator String Value. Default is "" which uses System.lineSeparator	"@"	""
xmlRootElement	Name of the root element in XML.	"customer"	"root"
prettyPrintXml	If true, indents XML output for readability.	true	false
includeXmlDeclaration	Adds XML declaration header (<?xml ⋯ ?>) if true.	true	true
xmlVersion	XML version to declare.	"1.0"	"1.0"
xmlEncoding	Encoding to declare in XML.	"UTF-8"	"UTF-8"
xmlStandalone	Sets standalone declaration in XML (yes or no).	"no"	"no"
xmlMethod	Specifies XML output method (xml, html, text), if applicable.	"xml"	null
textValuesOnly	If true, only outputs values (no field names) in text format.	false	false
textValueSeparator	Separator between field names and values in text format.	": "	":"
textLineSeparator	Line separator for text format.	"\n"	"\n"

Setting	What It Does	Example	Default
`fieldOrder`	Ordered list of fields to output. Unlisted fields come after, unless `strictFieldOrder` is `true`.	`["Name", "Balance", "Birthday"]`	`[]`
`strictFieldOrder`	If `true`, only fields listed in `fieldOrder` are included.	`false`	`false`

Example YAML

```yaml
file-events:
  source:
    advanced:
      output:
        outputFormat: "json"
        prettyPrintJson: false
        ignoreNulls: true
        csvDelimiter: ","
        csvQuoteChar: "\""
        csvQuoteMode: MINIMAL
        csvRecordSeparator: ""
        xmlRootElement: "customer"
        prettyPrintXml: true
        includeXmlDeclaration: true
        xmlVersion: "1.0"
        xmlEncoding: "UTF-8"
        xmlStandalone: "no"
        xmlMethod: "xml"
        textValuesOnly: false
        textValueSeparator: ": "
        textLineSeparator: "\n"
        fieldOrder:
          - "Name"
          - "Balance"
          - "Birthday"
        strictFieldOrder: false
```

Example Configurations and Outputs

Below are formatted examples showing how the output would look for different `outputFormat` settings using the following sample input data:

```
Name: Alice
Balance: 1234.56
Birthday: 1990-12-25
Notes: null
```

JSON Output

YAML Configuration:

```
outputFormat: "json"
prettyPrintJson: true
ignoreNulls: false
```

Output:

```
{
  "Name": "Alice",
  "Balance": 1234.56,
  "Birthday": "1990-12-25",
  "Notes": null
}
```

With `ignoreNulls: true`:

```
{
  "Name": "Alice",
  "Balance": 1234.56,
  "Birthday": "1990-12-25"
}
```

CSV Output

YAML Configuration:

```
outputFormat: "csv"
csvDelimiter: ","
csvQuoteChar: "\""
csvQuoteMode: ALL
fieldOrder:
  - "Name"
  - "Balance"
  - "Birthday"
  - "Notes"
```

Output:

```
"Name","Balance","Birthday","Notes"
"Alice","1234.56","1990-12-25",""
```

XML Output

YAML Configuration:

```
outputFormat: "xml"
xmlRootElement: "customer"
prettyPrintXml: true
includeXmlDeclaration: true
```

Output:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<customer>
  <Name>Alice</Name>
  <Balance>1234.56</Balance>
  <Birthday>1990-12-25</Birthday>
  <Notes/>
</customer>
```

Text Output

YAML Configuration:

```
outputFormat: "text"
textValueSeparator: ": "
textLineSeparator: "\n"
textValuesOnly: false
```

Output:

```
Name: Alice
Balance: 1234.56
Birthday: 1990-12-25
Notes: null
```

With `textValuesOnly: true`:

```
Alice
1234.56
1990-12-25
null
```

Raw Output (Default)

If `outputFormat` is not specified or explicitly set to "raw", the application may output the original file

or record as-is, without formatting.

Extended Example Configurations and Outputs

1. JSON Output Example

YAML Configuration:

```yaml
file-events:
  source:
    advanced:
      output:
        outputFormat: "json"
        prettyPrintJson: true
        ignoreNulls: true
        fieldOrder:
          - "Name"
          - "Balance"
          - "Birthday"
        strictFieldOrder: false
```

Expected JSON Output:

```
{
  "Name": "Alice",
  "Balance": 1234.56,
  "Birthday": "1990-12-25"
}
```

- `prettyPrintJson: true` adds indentation and line breaks.
- `ignoreNulls: true` removes fields with `null` values.

2. CSV Output Example

YAML Configuration:

```yaml
file-events:
  source:
    advanced:
      output:
        outputFormat: "csv"
        csvDelimiter: ","
        csvQuoteChar: "\""
        csvQuoteMode: ALL
        fieldOrder:
          - "Name"
          - "Balance"
          - "Birthday"
```

```
        strictFieldOrder: true
```

Expected CSV Output:

```
"Name","Balance","Birthday"
"Alice","1234.56","1990-12-25"
```

- `csvDelimiter: ","` separates values with commas.
- `csvQuoteChar: "\""` wraps values in double quotes.

3. XML Output Example

YAML Configuration:

```
file-events:
  source:
    advanced:
      output:
        outputFormat: "xml"
        xmlRootElement: "customer"
        prettyPrintXml: true
        includeXmlDeclaration: true
        xmlVersion: "1.0"
        xmlEncoding: "UTF-8"
        xmlStandalone: "no"
```

Expected XML Output:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<customer>
  <Name>Alice</Name>
  <Balance>1234.56</Balance>
  <Birthday>1990-12-25</Birthday>
</customer>
```

- `xmlRootElement: "customer"` sets the root element to `<customer>`.
- `prettyPrintXml: true` adds indentation for readability.

4. Text Output Example

YAML Configuration:

```
file-events:
  source:
    advanced:
      output:
        outputFormat: "text"
        textValuesOnly: false
```


solace.

```
        textValueSeparator: ": "
        textLineSeparator: "\n"
```

Expected Text Output:

```
Name: Alice
Balance: 1234.56
Birthday: 1990-12-25
```

- `textValuesOnly: false` means field names are included with their values.
- `textValueSeparator: ": "` uses `: ` to separate field names from values.
- `textLineSeparator: "\n"` adds a newline after each field.

5. Raw Output Example

YAML Configuration:

```
file-events:
  source:
    advanced:
      output:
        outputFormat: "raw"
```

Expected Raw Output:

```
Name: Alice
Balance: 1234.56
Birthday: 1990-12-25
Notes: null
```

- The raw output is just the original input data with no formatting.

File Events Sink (Events to File)

Configuring Output Destination

The configuration property `spring.cloud.stream.bindings.output-0.destination` defines the output destination path for the sink micro integration (MI) instance.

This property supports dynamic file naming using the Dynamic Expressions resolver, allowing the output path and filename to be constructed at runtime based on context such as dates, topic components, message headers, and more.

For more details on dynamic expressions, refer to the Dynamic Expressions section.

Example Usage Dynamic Expressions

Set the property to a path template that includes dynamic expressions enclosed in ${...}. At runtime, the expressions are resolved and substituted into the path, creating unique output files.

Common expressions useful for output destinations include:

Expression Category	Description
`DATE`	Inserts a formatted timestamp, with optional offsets and formatting. Example: ${DATE(%1$tY%1$tm%1$td)} resolves to 20251015.
`TOPIC`	Extracts components from the topic hierarchy by index. Example: ${TOPIC(2)} for the third segment (0-based indexing); ${TOPIC(-2)} for the second-to-last segment.
`HEADER`	References values from message headers. Example: ${HEADER(header)} resolves to the value of the specified header.
`FILE`	Extracts metadata from the source file, such as name or path. Example: ${FILE(FBASE)} for the base filename without extension.
`UMAP`	References user-defined properties. Example: ${UMAP(region)} for a property named "region".
`AUTO_INCREMENT`	Replaces the dynamic variable expression with the auto increment integer value associated with the file count or the event count depending on the field it is used inside. This count is maintained by the application in the application state.
`Level 2 Operators`	Post-processes the result of a Level 1 expression, e.g., #UPPER(${TOPIC(2)}) to uppercase a topic segment.

The resolved path must be a valid absolute or relative file path, ending with a filename (optionally dynamic).

For more details on dynamic expressions, refer to the Dynamic Expressions section.

Examples

Basic Date-Based Naming

```
spring.cloud.stream.bindings.output-0.destination = /output/daily-report-
${DATE(%1$tY%1$tm%1$td)}.txt
```

This generates a file like /output/daily-report-20251015.txt based on the current date.

Topic and Header Integration

```
spring.cloud.stream.bindings.output-0.destination = /output/logs/${TOPIC(1)}-
${HEADER(source-id)}-${DATE(%1$tH%1$tM)}.log
```

For a topic like acme/sales/order/created and header source-id: app1, this resolves to /output/logs/acme-app1-1430.log.

Advanced with File Metadata and Increment

```
spring.cloud.stream.bindings.output-0.destination = /output/files/file-
${DATE(%1$tY%1$tm%1$td)}-${TOPIC(2)}-${TOPIC(-2)}-${HEADER(header)}-
${AUTO_INCREMENT}.txt
```

This creates a file like /output/files/file-20251015-sales-created-user123-1.txt, where ${AUTO_INCREMENT} appends a sequential number for uniqueness (e.g., 1, 2). Note: ${AUTO_INCREMENT} is a special expression for auto-incrementing counters per run or batch.

For more complex scenarios, combine expressions with Level 2 operators, such as #UPPER(${TOPIC(2)}) to enforce case conventions in filenames.

For more details on dynamic expressions, refer to the Dynamic Expressions section.

Dynamic Variable expressions Quick guide

These expressions can be used inside the following configuration values when configuring sink MI.

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`

For more details on dynamic expressions, refer to the Dynamic Expressions section.

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
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
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
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"

Dynamic Variable	Expression	Description
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.

File Events Sink (Events to File) Configuration Options

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

General Configuration

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 Configuration

Config Option	Type	Valid Values	Default Value	Description
`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.
advanced.alwaysVerifyIfFileExists	boolean	true, false	false	Default Value false. If set to true, the MI will check if the previously opened destination file exists before writing any new event to the file. If the file doesn't exist, it will either throw an exception or if recreateFileIfNotExists is set to true it will recreate the file.

Config Option	Type	Valid Values	Default Value	Description
advanced.recreateFileIfNotExists	boolean	true, false	false	Default Value false. If set to true, MI will recreate the destination file if it was removed or moved and doesn't exist. Only works if alwaysVerifyIfFileExists is set to true.
advanced.alwaysAddLineSeparator	boolean	true, false	true	Default Value true. Sink MI will add a line separator at the end of each event by default

Event Pre Processor Configuration

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.

Config Option	Type	Valid Values	Default Value	Description
`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.

Event Post Processor Configuration

Config Option	Type	Valid Values	Default Value	Description
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.

Example Configurations for Quick Setup

Always configure destination to unique names using ${AUTO_INCREMENT} or some other unique value depending on the business requirement. Existing files will be appended.

Single Event per File

```
spring:
  application:
    name: File Events (Sink)
  cloud:
    stream:
      bindings:
        input-0:
          destination: q.file.events.mesh.queue
          binder: solace
        output-0:
          destination: /output/files/file-${DATE(%1$tY%1$tm%1$td)}-
${AUTO_INCREMENT}.txt
          binder: file-events
file-events:
  sink:
    general:
      miID: SINK_ID1
      state_backup_path: backup_state.cfg
    advanced:
      maxEventsPerFile: 1
```

No Limits for Event count per file

```
file-events:
  sink:
    general:
      miID: SINK_ID1
      state_backup_path: backup_state.cfg
    advanced:
      maxEventsPerFile: 0
```

Max Size per file

Maximum size in bytes

```
file-events:
  sink:
    general:
      miID: SINK_ID1
      state_backup_path: backup_state.cfg
```

```
    advanced:
      maxFileSize: 2048
```

No Size limits per file

```
file-events:
  sink:
    general:
      miID: SINK_ID1
      state_backup_path: backup_state.cfg
    advanced:
      maxFileSize: 0
```

Event Count and File Size limits

New events will be appended to file until one of the conditions is reached.

Note - Output destination file names should resolve to unique files

```
file-events:
  sink:
    general:
      miID: SINK_ID1
      state_backup_path: backup_state.cfg
    advanced:
      maxEventsPerFile: 10
      maxFileSize: 4096
```

Event Pre Processors

The `pre_process` sections allow you to add prefixes to files and events during processing. Useful when generating json, xml or log files

These support dynamic expressions; refer to the Dynamic Expressions section for details.

```
file-events:
  sink:
    general:
      miID: SINK_ID1
      state_backup_path: backup_state.cfg
    advanced:
      maxEventsPerFile: 0
      maxFileSize: 0
    pre_process:
      prependToFile: "START" #Adds a String to the start of file
      prependToFirstEvent: "${AUTO_INCREMENT},${DATE(%1$tc)}," #Adds a String before
writing the first event. Supports Dynamic Expressions
```

```
      prependToEvents: "${AUTO_INCREMENT},${DATE(%1$tc)}," #Adds a String before
writing the event. Supports Dynamic Expressions
```

Event Post Processors

The `post_process` sections allow you to add suffixes to files and events during processing. Useful when generating json, xml or log files

These support dynamic expressions; refer to the Dynamic Expressions section for details.

```
file-events:
  sink:
    general:
      miID: SINK_ID1
      state_backup_path: backup_state.cfg
    advanced:
      maxEventsPerFile: 0
      maxFileSize: 0
    post_process:
      appendToFile: "This is file Close" #Appends to the end of each file
      appendToEvents: ",END" #Appends to every event. Supports Dynamic Expressions
```

Generating XML Output File

The `pre_process` and `post_process` sections allow you to add prefixes and suffixes to files and events during processing.

```
file-events:
  sink:
    general:
      miID: SINK_ID1
      state_backup_path: backup_state.cfg
    advanced:
      maxEventsPerFile: 0
      maxFileSize: 0
    pre_process:
      prependToFile: "<xml>"
      prependToEvents: "<event>"
    post_process:
      appendToEvents: "</event>"
      appendToFile: "</xml>"
```

Generating JSON Output File

The `pre_process` and `post_process` sections allow you to add prefixes and suffixes to files and events during processing.

```
file-events:
  sink:
    general:
      miID: SINK_ID1
      state_backup_path: backup_state.cfg
    advanced:
      maxEventsPerFile: 0
      maxFileSize: 0
    pre_process:
      prependToFile: "{["
      prependToFirstEvent: "{"
      prependToEvents: ",{"
    post_process:
      appendToEvents: "}"
      appendToFile: "]}"
```

Sink configuration for Source MI Events

```
spring:
  application:
    name: File Events (Sink)
  cloud:
    stream:
      bindings:
        input-0:
          destination: q.file.events.mesh.queue
          binder: solace
        output-0:
          destination: ${header(DEST_FILE_NAME)}
          binder: file-events
file-events:
  sink:
    general:
      miID: SINK_ID1
      state_backup_path: ./backup_state.cfg
    advanced:
      maxEventsPerFile: 0
      maxFileSize: 0
      ignoreEmptyEvents: true
      preventDuplicateEvents: true
      preventDuplicateOnlyForRedeliveredEvents: false
      uniqueEventIdentifyingHeader: "${header(DEST_FILE_NAME)}-${HEADER(EVENT_NO)}"
      closeFileTriggerHeaderName: "EVENT_TYPE"
      closeFileTriggerHeaderValue: "FILE_COMPLETE"
      ignorePayloadOfCloseTriggerEvent: true
```

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)

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.(source|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.
`gcs.projectId`	`string`	`any`	`empty`	ProjectId value from Google Cloud Storage.

Config Option	Type	Valid Values	Default Value	Description
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 that provides a secure way to access, transfer, and manage large files and sensitive data over a network.

SFTP leverages SSH for secure file transfers and mandates client authentication by the server to ensure only authorized access.

All commands and data are encrypted end-to-end, safeguarding passwords and other sensitive information from exposure in plain text during transit.

Authentication methods supported include `Password` or `Private Key` (optionally protected by a passphrase for added security).

These configuration options are all prefixed by `file-events.(source|sink).`

Config Option	Type	Valid Values	Default Value	Description
`sftp.enabled`	`boolean`	`false, true`	`false`	Enable or disable SFTP support. When set to false, file operations are performed on the local machine; when true, files are retrieved from or sent to a remote SFTP server, enabling secure remote file handling.
`sftp.ip`	`string`	`any`	`empty`	The IP address or hostname of the target SFTP server to connect to for remote file access.
`sftp.port`	`int`	`any`	`22`	The port number used by the SFTP server. The default (22) is the standard port for SSH/SFTP services; adjust if your server uses a non-standard port.
`sftp.user`	`string`	`any`	`empty`	The username associated with the SFTP account for authentication purposes.
`sftp.password`	`string`	`any`	`empty`	The password corresponding to the SFTP username. For enhanced security, consider using private key authentication instead of passwords.

Config Option	Type	Valid Values	Default Value	Description
sftp.strictHostKeyChecking	string	yes, no	yes	Determines whether to perform strict host key verification during connection establishment. Set to 'yes' (default) for secure validation against known hosts; 'no' disables it, which may be useful for testing but reduces security.
sftp.privateKeyPath	string	any	empty	The absolute or relative file path to the private key file used for key-based authentication, as an alternative to password-based login. Ensure the file permissions are restricted for security.
sftp.privateKeyPassphrase	string	any	empty	The passphrase used to decrypt an encrypted private key file. Omit if the private key is unencrypted.
sftp.connectTimeout	int	any	5000	The maximum duration in milliseconds to attempt establishing a connection to the SFTP server before declaring a timeout and retrying or failing.
sftp.keepAliveInterval	int	any	60000	The frequency in milliseconds for sending keep-alive packets to the SFTP server, helping to prevent connection drops due to perceived inactivity.

SFTP Proxy Configuration

Proxy settings allow routing SFTP connections through an intermediary HTTP or SOCKS proxy for scenarios like network restrictions or enhanced privacy. These options are prefixed by file-events.(source|sink).sftp.proxy.. The proxy is disabled by default and only activated when at least the proxy type, host, and port are configured.

Config Option	Type	Valid Values	Default Value	Description
sftp.proxy.proxyType	string	HTTP, SOCKS4, SOCKS5	empty	Specifies the proxy protocol: 'HTTP' for HTTP proxies, 'SOCKS4' for SOCKS version 4, or 'SOCKS5' for the more feature-rich SOCKS version 5 (supports authentication and UDP).
sftp.proxy.proxyHost	string	any	empty	The hostname or IP address of the proxy server that will forward SFTP traffic.
sftp.proxy.proxyPort	int	any	0	The listening port on the proxy server for incoming connections. Common defaults are 8080 for HTTP and 1080 for SOCKS.
sftp.proxy.proxyUsername	string	any	empty	The username for proxy server authentication, required for proxies that enforce user credentials (common in SOCKS5 and some HTTP setups).
sftp.proxy.proxyPassword	string	any	empty	The password paired with the proxy username for secure proxy authentication.

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|source).

Config Option	Type	Valid Values	Default Value	Description
`ftp.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. Enabling this shifts the connector from local file system operations to remote FTP/FTPs interactions, allowing integration with external servers for file polling, listing, and transfer.
`ftp.traceEnabled`	`boolean`	`true,false`	`false`	Enable detailed FTP command/response tracing. When true, logs all FTP protocol exchanges (e.g., commands sent, server replies) to the TRACE level via SLF4J for debugging; disable in production to reduce log volume.
`ftp.ip`	`string`	`any`	`empty`	FTP Server IP. Specify the hostname or IP address of the remote FTP/FTPs server to connect to. This is required when enabled is true; empty value defaults to local file system if not set.

Config Option	Type	Valid Values	Default Value	Description
ftp.port	int	any	21	FTP server port number. The default port is 21 for plain FTP; use 990 for implicit FTPS. Ensure the port matches the server's configuration to establish successful connections.
ftp.user	string	any	empty	FTP user username. The credentials for authenticating with the FTP server. Anonymous access may use 'anonymous' if supported by the server, but a valid username is typically required.
ftp.password	string	any	empty	FTP user password. The password corresponding to the username for server authentication. For security, store this in a secure vault or environment variable rather than plaintext configuration.
ftp.useFTPS	boolean	true,false	false	For connecting over FTPs - TLS/secured. When true, enables FTPS for encrypted control and data channels, protecting against eavesdropping and man-in-the-middle attacks. Requires proper SSL/TLS configuration.
ftp.implicit	boolean	true,false	false	Use implicit FTPS mode (typically for port 990). In implicit mode, the entire session is encrypted from the start without needing an AUTH command; set to true only for servers requiring this legacy behavior.

Config Option	Type	Valid Values	Default Value	Description
ftp.strictHostKeyChecking	string	yes,no	yes	Strict host key checking for FTPS (yes/no). When 'yes', verifies the server's hostname against its certificate to prevent spoofing; set to 'no' for self-signed or untrusted certificates, but use cautiously.
ftp.keyStorePath	string	any	empty	Path to key store for client certificate authentication. Specifies the location of the JKS keystore file containing the client's private key and certificate for mutual TLS authentication in FTPS.
ftp.keyStorePassword	string	any	empty	Password for the key store. The passphrase to unlock the keystore file; required if keyStorePath is provided. Handle securely to avoid exposure.
ftp.trustStorePath	string	any	empty	Path to trust store for server certificate validation. Points to the JKS truststore containing trusted CA certificates to validate the FTPS server's identity.
ftp.trustStorePassword	string	any	empty	Password for the trust store. The passphrase for the truststore file; necessary when trustStorePath is set for custom certificate validation.
ftp.enabledProtocols	string	any	TLSv1.2,TLSv1.3	Comma-separated list of enabled TLS protocols. Restricts the TLS versions used in FTPS connections for security (e.g., 'TLSv1.3' only); defaults to modern secure protocols.

Config Option	Type	Valid Values	Default Value	Description
ftp.passiveMode	boolean	true,false	false	Sets the data connection mode to PASSIVE_LOCAL_DATA_CONNECTION_MODE. Enables passive mode where the server opens a port for data transfer, ideal for clients behind firewalls/NAT; active mode may be better for server-side restrictions.
ftp.activeMinPort	int	1-65535	0	Minimum local port for active mode data connections (0 for system-assigned). Defines the lower bound of ephemeral ports used in active mode to comply with firewall rules; set range for predictability.
ftp.activeMaxPort	int	1-65535	0	Maximum local port for active mode data connections (0 for system-assigned). The upper bound for active mode ports; must be >= activeMinPort if both are set.
ftp.listHiddenFiles	boolean	true,false	false	Include hidden files in directory listings. When true, uses server-specific commands (e.g., -a in UNIX) to list dot-files or hidden attributes, useful for complete file discovery.
ftp.useEPSVwithIPv4	boolean	true,false	false	Enable EPSV command in passive mode for IPv4 networks. Uses extended passive mode (EPSV) for better NAT/firewall compatibility by avoiding IP embedding in responses; fallback to PASV if unsupported.
ftp.remoteVerificationEnabled	boolean	true,false	false	Verify remote hostname/IP in passive mode responses. When true, cross-checks the IP/port returned by PASV/EPSV against the control connection host to detect proxy/misrouting issues.

Config Option	Type	Valid Values	Default Value	Description
ftp.serverSystemKey	string	UNIX,VMS,NETWARE,SYST_L8,etc.	UNIX	Server system type for parsing directory listings (e.g., UNIX, VMS, NETWARE). Configures the FTP client to interpret server-specific listing formats and timestamps accurately, preventing errors in file metadata extraction.
ftp.defaultFileTransferType	enum	ASCII,EBCDIC,BINARY,EBCDIC_IMAGE,LOCAL_EBCDIC	BINARY	Default file transfer mode (ASCII for text with line-ending translation; BINARY for raw bytes to avoid corruption). Set via enum value; applied globally unless overridden per operation. Use BINARY for mixed content safety.
ftp.serverTimezone	string	IANA Time Zone Database identifiers (e.g., "UTC", "America/New_York", "Europe/London")	System default (JVM□s local timezone)	The timezone of the FTP server used for parsing timestamps in directory listings and MDTM commands. If not specified, falls back to the JVM's default timezone for local alignment in file tracking.
ftp.connectTimeout	int	any (>0)	5000	Connection timeout in milliseconds. The maximum time to wait for the initial TCP connection to the FTP server; adjust higher for slow networks or lower for faster failure detection.
ftp.keepAliveInterval	int	any (>0)	60000	Control keep-alive interval in milliseconds. Sends periodic NOOP commands to maintain the control connection during idle periods, preventing timeouts on long-polling scenarios.

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

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
file_mesh_manager.events.connectorInit	boolean	true, false	true	true → send connector init event to File Mesh Manager false → No connector init event
file_mesh_manager.events.connectorStart	boolean	true, false	true	true → send connector start event with all configurations to File Mesh Manager false → No connector start event

Config Option	Type	Valid Values	Default Value	Description
`file_mesh_manager.events.connectorComplete`	`boolean`	`true`, `false`	`true`	true → send connector complete event after all files have been processed to File Mesh Manager false → No connector complete event

License

See the license terms that apply to Self-Managed Micro-Integrations.

Support

See Support for how to get help or report an issue with a Self-Managed Micro-Integration.